d-IRA : A Distributed Reachability Algorithm for Analysis of Linear Hybrid Automata

Sumit Kumar Jha

Computer Science Department, Carnegie Mellon University, Pittsburgh PA 15213

Abstract. This paper presents the design of a novel distributed algorithm d-IRA for the reachability analysis of linear hybrid automata. Recent work on *iterative relaxation abstraction* (IRA) is leveraged to distribute the reachability problem among multiple computational nodes in a non-redundant manner by performing careful infeasibility analysis of linear programs corresponding to spurious counterexamples. The d-IRA algorithm is resistant to failure of multiple computational nodes. The experimental results provide promising evidence for the possible successful application of this technique.

1 Introduction

The verification of linear hybrid automata is a computationally expensive procedure and is efficient only for systems with few continuous variables. Linear hybrid automata (LHA) are an important class of hybrid systems which can approximate nonlinear hybrid systems in an asymptotically complete fashion [3]. We extend our earlier work [5] on applying counterexample guided abstraction refinement (CEGAR) based model checking algorithms [1] to the analysis of linear hybrid automata and present a distributed algorithm for their reachability analysis.

This paper makes the following three novel contributions:

1. We present the first fault-tolerant distributed algorithm for the reachability analysis of linear hybrid automata.

2. On the theoretical side, we establish a partial-order among counterexamples and relaxations of linear hybrid automata. We find counterexamples not related by the partial order and build relaxations to refute each of them in a distributed manner.

3. The global state which needs to be preserved for failure-tolerance of the distributed system is only a discrete finite state machine. We also illustrate the potential for efficient online back-ups of the global state.

2 The Distributed Algorithm (d-IRA)

The distributed algorithm assumes one master computation node and (N-1) other computational (slave) nodes. Initially, the master node initialises a counter i to zero, chooses the empty set as an initial set of variables $\mathcal{I}_0$ and learns the deterministic finite automata corresponding to Σ^* (where Σ is the alphabet of the linear hybrid automata) as the initial discrete over-approximate *global abstraction* of the language of the LHA H. Now, we explain the distributed algorithm.

1. During the i^{th} iteration, the j^{th} computational node constructs its own relaxation H_i^j of the linear hybrid automata H using the set of

variables I_i^j. This step could involve invoking the Fourier-Motzkin elimination routine. Each computational node then constructs a discrete abstraction $Temp^j$ corresponding to the relaxed linear hybrid automata H_i^j. This step involves making calls to the underlying reachability engine like PHAVer [2]. Both the above steps are identical to the corresponding steps in the IRA algorithm [5] and are not discussed here for brevity.
2. Each computational node sends to the master the discrete abstraction $Temp^j$ which it learnt from the relaxed linear hybrid automata H_i^j. The master node updates the discrete *global abstraction* A_{CE}^{i+1} by taking the intersection of the previous discrete *global abstraction* A_{CE}^{i} with all the newly learnt discrete abstractions $Temp^j$.
3. Then, the master uses partial order relation among the counter-examples A_{CE}^{i+1} to pick a set CE of N *non-redundant* counterexamples. The construction of partial order relation is detailed in Section 3.
4. The master node checks if the set of counterexamples CE is empty. If A_{CE}^{i+1} has no counterexamples, then no bad states are reachable in the system [5] and hence, it is declared to be safe. Otherwise, the master computational node forms a set of linear programs $\mathcal{C}$, where each linear program corresponds to one of the counterexamples in CE_{i+1}. This step is similar to the corresponding step in the IRA algorithm [5] and is discussed in [6].
5. The master node checks if any of the linear programs in $\mathcal{C}$ is feasible. In any of them, say C, is feasible, it stops and reports that the bad state is reachable [6] and reports the corresponding counterexample. If none of the linear programs is feasible, the master node finds the *irreducible infeasible subsets* (IIS) for each of the linear programs. The master node uses the support of the IIS as the choice for the next set of variables $\mathcal{I}_{i+1}$ which will be used to construct the relaxations. The master node communicates the set I_{i+1}^j to the j^{th} client.

3 A Partial Order for Counterexamples and Relaxations

In order to make the distributed computation effective, it is essential that the various computational nodes do not solve equivalent reachability sub-problems. In particular, we want to make sure that the relaxed linear hybrid automata for the i^{th} iteration H_i^j and H_i^k are different. We achieve this goal by making a suitable choice of counterexamples from the global abstraction A_{CE}^{i+1}. Before we present our algorithmic methods, we define some related notions. Our definitions of linear hybrid automata, relaxations and counterexamples are identical to those in literature [3, 5]. Given a path ρ in a linear hybrid automata H, we can derive a set of corresponding linear constraints $Constraints(H, \rho)$ which is feasible if and only if the path is feasible. This construction [5, 6] is omitted here.

Definition 1. Minimal Explanation for Infeasible Counterexamples *: Given a counterexample path ρ which is infeasible in a linear hybrid automata H but feasible in a relaxation H' of H, (i.e. $H' \sqsubseteq H$), a set of linear constraints $IIS(\rho)$ is said to be an irreducible infeasible subset (IIS) for ρ if and only if:*

– $IIS(\rho) \subseteq Constraints(H, \rho)$ *and* $IIS(\rho)$ *is not feasible.*
– *for any set* S *s.t.* $S \subset IIS(\rho)$, S *is feasible.*

The special basis [5] Var *of the IIS of* ρ *is called a* minimal explanation for the infeasible counterexample *and we write it as* $Var(\rho, IIS(\rho))$.

In the following, we assume that there exists a function $\mathcal{IIS}$ which maps each counterexample to a unique IIS.

Definition 2. Dominance of Counterexamples *: A counterexample ce is said to dominate a counterexample* ce' *if and only if* $Var(ce, \mathcal{IIS}(ce)) \subseteq Var(ce', \mathcal{IIS}(ce'))$. *We write* $ce \succeq ce'$.

Definition 3. *Two counterexamples ce and* ce' *are said to be equivalent iff* $Var(ce, \mathcal{IIS}(ce)) = Var(ce', \mathcal{IIS}(ce'))$. *Then, we say* $ce \approx ce'$.

The relaxations of hybrid automata form a partial order. We summarize our results based on this key observation in the following theorems. The proofs are presented in [4].

Theorem 1. *The dominance relation* $\succeq$ *among counterexamples is a partial order relation.*

Theorem 2. *Let* H_{ce} *be the relaxation of* H *w.r.t.* $Var(ce, \mathcal{IIS}(ce))$ *and* $H_{ce'}$ *be the relaxation of* H *w.r.t.* $Var(ce', \mathcal{IIS}(ce'))$. *If the counterexample ce dominates the counterexample* ce' *i.e.* $ce \succeq ce'$, *then* H_{ce} *is a relaxation of* $H_{ce'}$ *i.e.* $H_{ce} \sqsubseteq H_{ce'}$.

The algorithm *Select_CE* presented below for selecting N counterexamples is based on the above results.

Algorithm *Select_CE*
Input: Global Abstraction Automata A^i_{CE}, LHA H, a timer TIMEOUT.
Output: N counterexamples: $CE = \{ce_1, \ldots ce_N\}$
1. Initialize CE to be the empty set.
2. Pick a set of m ($> N$) distinct counterexamples $C = \{ce_1, ce_2 \ldots ce_m\}$ from A^i_{CE}.
3. Build a set of linear programs $\{lp_1, lp_2 \ldots lp_m\}$ corresponding to each of $\{ce_1, ce_2 \ldots ce_m\}$
4. For each (infeasible) linear program lp_i, obtain an IIS and remember it as $\mathcal{IIS}(lp_i)$
5. For each counterexample $ce_i \in C$,
   a. Check whether there exists a counterexample $ce_j \in C$ such that $ce_j \succeq ce_i$ $(i \neq j)$.
   b. If no such counterexample ce_j exists, add ce_i to CE.
   c. Remove ce_i from C.
6. If ($|CE| < N$ *and* $!TIMEOUT$) , $m = m \times 2$; goto step 2.
7. RETURN the first N members of CE as a set.

4 Failure Tolerance of d-IRA

Resistance to failures and restarts of slave nodes: This is possible because the slave nodes do not store any global state information during

the distributed computation and hence, the overall distributed reachability computation is robust to failure of slave nodes. If the i^{th} slave node fails during the j^{th} iteration, then the d-IRA algorithm can still proceed by making the assumption that $L(Temp_i) = \Sigma^*$.

Tolerance to failure of master node: The current state of the distributed computation is really captured completely by the global abstraction A^i_{CE} after the i^{th} iteration. It is hence desirable to back-up the global abstraction to a group of *shadow masters* during periods of low communication activity.

5 Experimental Results and Conclusion

We implemented a version of our distributed algorithm using the IRA infrastructure which parallelized only the relaxation step. We found up to a 4-X improvement runtime on our four processor machine[1] with this implementation on a set of parameterized adaptive cruise control examples [5]. .

Table 1. Distributed IRA vs IRA

Example	♯-Variables	Time for d-IRA [s]	Time for IRA [s]	Speedup
ACC-4	4	11	15	1.36
ACC-8	8	100	192	1.92
ACC-16	16	1057	3839	3.63
ACC-19	19	2438	9752	**4.0**

References

1. J. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.
2. G. Frehse. PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. In *HSCC*, pages 258–273, 2005.
3. P.-H. Ho. Automatic Analysis of Hybrid Systems, Ph.D. thesis, technical report CSD-TR95-1536, Cornell University, August 1995, 188 pages, 1995.
4. S. K. Jha. Design of a distributed reachability algorithm for analysis of linear hybrid automata. *CoRR*, abs/0710.3764, 2007.
5. S. K. Jha, B. H. Krogh, J. E. Weimer, and E. M. Clarke. Reachability for linear hybrid automata using iterative relaxation abstraction. In *HSCC*, pages 287–300, 2007.
6. X. Li, S. K. Jha, and L. Bu. Towards an Efficient Path-Oriented Tool for Bounded Reachability analysis of Linear Hybrid Systems using Linear Programming. In *BMC*, 2006.

[1] We ran our experiments on a four processor 64-bit AMD Opteron(tm) 844 SMP machine running Red Hat Linux version 2.6.19.1-001-K8.

A Counterexample-Guided Approach to Parameter Synthesis for Linear Hybrid Automata

Goran Frehse[1], Sumit Kumar Jha[2], and Bruce H. Krogh[3]

[1] Verimag (UJF-CNRS-INPG), 2, av. de Vignate, 38610 Gières, France
goran.frehse@imag.fr
[2] Computer Science Department, Carnegie Mellon University
[3] ECE Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
jha@cs.cmu.edu, krogh@ece.cmu.edu

Abstract. Our goal is to find the set of parameters for which a given linear hybrid automaton does not reach a given set of bad states. The problem is known to be semi-solvable (if the algorithm terminates the result is correct) by introducing the parameters as state variables and computing the set of reachable states. This is usually too expensive, however, and in our experiments only possible for very simple systems with few parameters. We propose an adaptation of counterexample-guided abstraction refinement (CEGAR) with which one can obtain an underapproximation of the set of good parameters using linear programming. The adaptation is generic and can be applied on top of any CEGAR method where the counterexamples correspond to paths in the concrete system. For each counterexample, the cost incurred by underapproximating the parameters is polynomial in the number of variables, parameters, and the length of counterexample. We identify a syntactic condition for which the approach is complete in the sense that the underapproximation is empty only if the problem has no solution. Experimental results are provided for two CEGAR methods, a simple discrete version and iterative relaxation abstraction (IRA), both of which show a drastic improvement in performance compared to standard reachability.

1 Introduction

The admissible behaviors of linear hybrid automata (LHA) are determined by sets of linear constraints. The parameters in these constraints represent either physical constants or values chosen by the designer. When the LHA does not satisfy the design specifications, the latter constraints can be adjusted to eliminate the undesirable behaviors. This paper concerns this design problem in the context of reachability specifications: Given a parameterized LHA, determine the set of design parameters, called *good parameters*, for which no bad locations can be reached.

The parameter design problem for LHA was formulated and solved by Henzinger et al. [1], but the proposed solution is tractable for only very simple

systems with few parameters. This paper concerns the extension of verification techniques to solve the LHA parameter design problem. Our approach leverages the fact that the feasibility of a given counterexample path corresponds to the satisfiability of a set of linear constraints over instantiations of the initial and final values of the continuous variables in each location along the path, along with variables representing the duration of the continuous state trajectory in each location. Although this observation does not make the parameter design problem tractable, because projection of these constraints into the parameter space is computationally complex and there can be in general an infinite number of counterexample paths, it does lead to a set of heuristics that make it possible to efficiently compute underapproximations of the set of good parameters.

The heuristics we propose are integrated into *counterexample guided abstraction refinement* (CEGAR) [2]. In a standard CEGAR loop, a discrete abstraction of the system is used to find a counterexample, which is a path from the initial states to states considered bad. In a feasibility check, it is then verified whether this path corresponds to a behavior of the concrete hybrid system or whether it was a spurious product of the abstraction. If it is spurious, the abstraction is refined and the loop repeats. If the counterexample corresponds to a concrete behavior, the system is unsafe. Our adaptation consists of replacing the feasibility check with an operator that obtains constraints on the parameters that *make* the counterexample infeasible. If all counterexamples have been eliminated, the resulting constraints describe a set of parameters for which the system is safe. The make-infeasible operator can be implemented approximatively using linear programming, e.g., obtaining rectangular or octagonal underapproximations of the good parameters. Its complexity for each counterexample is polynomial in the number of variables, parameters and the length of the counterexample, compared to exponential complexity of an exact solution. Depending on whether the number of counterexamples or the number of parameters is the dominating cost factor, we apply the underapproximation to each path individually or collectively on sets of paths.

In the general case, the underapproximation may produce an empty set even though good parameters exist. We identify a condition we call *parameter-monotonicity* (intuitively, when parameters function either as lower or upper bounds but not both), under which octagonal approximations are sufficient to prevent this from happening.

The entire approach is generic in the sense that it can be applied to any CEGAR loop in which the counterexamples correspond to paths in the concrete system (as opposed to sets of paths or transitions). It suffices to replace the feasibility check with the make-infeasible operator. We provide experimental results for two different CEGAR implementations: a simple variant of standard discrete CEGAR, and iterative relaxation abstraction (IRA) [3]. Compared to the traditional way of synthesizing parameters using reachability as in [1], we observe a dramatic improvement in speed.

The following section defines the class of LHA with parameters studied in this paper. Section 3 describes the role of counterexamples in defining the set

of good parameters. Section 4 discusses the special case when the parameters are *monotonic*, which means each parameter serves as either an upper or lower bound throughout the LHA. Section 5 presents a general counterexample-guided procedure for computing sets of good parameters and Section 6 presents experimental results for two implementations of the procedure. The concluding section discusses directions for further research.

2 Linear Hybrid Automata with Parameters

We consider *linear hybrid automata* (LHA) [4] with explicit parameter variables. An LHA $H = (Var, Lab, Loc, Inv, Flow, Trans, ini)$ consists of:

- A finite set of real-valued *variables* $Var = \mathtt{X} \cup \mathtt{P}$, where $\mathtt{X} = \{\mathtt{x_1}, \ldots, \mathtt{x_n}\}$ are the *continuous state variables* and $\mathtt{P} = \{\mathtt{p_1}, \ldots, \mathtt{p_m}\}$ are the *parameters*, which remain constant. We denote the values of variables with $x = (x_1, \ldots, x_n)^\intercal$ and $p = (p_1, \ldots, p_m)^\intercal$.
- A finite set of *labels* Lab.
- A finite set of *locations* Loc. A *state* (l, x, p) of the automaton consists of a location $l \in Loc$ and real values $(x, p) \in \mathbb{R}^{n+m}$ for each of the variables.
- For each location l, $Inv(l) \subseteq \mathbb{R}^{n+m}$ is the set of admissible values of the variables in the location.
- $Flow(l) \subseteq \mathbb{R}^n$ is the set of possible time derivatives $(\dot{x}_1, \ldots, \dot{x}_n)^\intercal$; the derivatives of the parameters are implicitly zero.
- A finite set of *transitions* $Trans \subseteq Loc \times Lab \times 2^{\mathbb{R}^{2n+m}} \times Loc$. A transition indicates that the system state may jump instantaneously from the transition *source state* (l, x, p) to the transition *target state* (l', x', p) if $(x, p, x') \in \mu$, where $\mu \subseteq \mathbb{R}^{2n+m}$ is the transition's *jump relation*. We desire a unique correspondence between sequences of transitions and sequences of labels, so we require that any location l has at most one outgoing transition for each label (the general case can be brought to this form by adding labels and renaming).
- A location $ini \in Loc$ is designated as *initial location* from which all behaviors must start.

The sets $Inv(l)$ and $Flow(l)$ are specified by conjunctions of linear constraints

$$a^\intercal x + e^\intercal p \leq b, \quad \text{respectively} \quad a^\intercal \dot{x} \leq b, \tag{1}$$

where a, e are vectors of integer coefficients and b is an integer. The jump relation μ of a transition is specified by a conjunction of linear constraints of the form

$$a^\intercal x + e^\intercal p + a'^\intercal x' \leq b, \tag{2}$$

where x denotes the values of the variables before the jump, and x' denotes the values after; the values of the parameters do not change. Given a conjunction C of linear constraints over $\mathtt{X}$ and $\mathtt{P}$, we write $[\![C]\!]$ to denote the set of values of (x, p) (a polyhedron) for which all of the constraints are satisfied. We write

$C(p')$ to denote the constraints obtained by substituting p with the values in p', and call $C(p')$ *infeasible* if $[\![C(p')]\!] = \emptyset$.

We define the semantics of a LHA in terms of *feasible paths* for a given parameter value p. This is consistent with the semantics in [4] but reformulated to simplify the use of linear programming. A *path* $\pi = \alpha_0\alpha_1 \ldots \alpha_{z-1}$ is a finite sequence of labels α_i such that the sequence of transitions $(l_i, \alpha_i, \mu_i, l'_i)$ satisfies $l_0 = ini$ and $l'_i = l_{i+1}$ for $i = 0, \ldots, z-1$ (this defines l_z to be the target state of the last transition). The path is *feasible* for a given p if there exist vectors $x_j^{in}, x_j^{out} \in \mathbb{R}^n$ and scalars $\delta_j \in \mathbb{R}$ for $j = 0, \ldots, z$ such that

- $(x_j^{in}, p), (x_j^{out}, p) \in Inv(l_j)$,
- $(x_j^{out} - x_j^{in})/\delta_j \in Flow(l_j)$,
- for $j < z$, $(x_j^{out}, p, x_{j+1}^{in}) \in \mu_j$.

In the above sequence, (l_j, x_j^{in}, p) is the state in which the automaton enters location l_j, and (l_j, x_j^{out}, p) is the state after letting time elapse for δ_j units. If $j < z$, the automaton leaves l_j via the transition identified by α_j, and jumps to the state $(l_{j+1}, x_{j+1}^{in}, p)$. For a given path the above constraints can be written as linear constraints over the $2z(n+1)+m$ variables of x_j^{in}, x_j^{out}, δ_j and p. We call these *path constraints* and denote them by $PathCon(\pi, p)$. Expressed in terms of the path constraints, a counterexample is feasible if $[\![PathCon(\pi, p)]\!] \neq \emptyset$, which can be decided using efficient linear programming techniques [5].

In this paper, we consider reachability problems for LHA. A location l is said to be *reachable* if there is a feasible path $\pi = \alpha_0\alpha_1 \ldots \alpha_{z-1}$ with $l_z = l$. An LHA H is said to be *safe* if none of the locations in a given set of *bad locations* L_B is reachable. We call a path to a bad location a *counterexample*, and write $CE(H, p)$ for the (possibly infinite) set of counterexamples in H for the parameter value p. Let $FCE(H, p)$ denote the set of *feasible counterexamples* in H. The system is safe if and only if $FCE(H, p)$ is empty, i.e., there are no counterexamples or none of the existing counterexamples is feasible. The extension of $CE(H, p)$ and $FCE(H, p)$ to sets of parameter values is straightforward. While we have made some restrictions to our LHA (unique labels) and the reachability problem (unsafe locations), it is straightforward to bring the general problem (unsafe states) to this form by relabeling transitions and introducing an error location reachable by transitions from the unsafe states. We will use the following example throughout the paper:

Example 1. Consider a buffer tank with steady inflow and with a controllable outlet valve resulting in a net level increase $\dot{x} = r$ if the valve is closed, and a net decrease $\dot{x} = -r$ if it is open. A controller is supposed to keep the level between x_{min} and x_{max}. The controller never waits longer than time T to check the level x, and opens (closes) the valve when $x > M$ ($x < m$).

The LHA model H_{tank} of the controlled system is shown in Fig. 1, where for simplicity jump constraints of the form $x' = x$ have been omitted. H_{tank} has the parameters m, M, T, x_{min}, and x_{max}. We assume r to be a given constant (parameters are not allowed in the flows). The forbidden location is *error*.

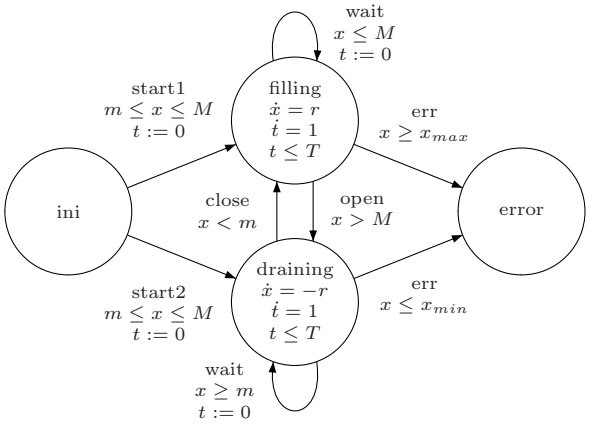


Fig. 1. LHA H_{tank} for the controlled tank example with parameters m, M, T, x_{min}, and x_{max}; r is a given constant since LHA-parameters cannot bound derivatives

H_{tank} has infinitely many counterexamples, a shortest of them being $\pi = \mathit{start1}, \mathit{err}$, which covers the locations ini, $\mathit{filling}$ and error, and has the following path constraints (some irrelevant ones are omitted):

$$\begin{array}{rrl} m \leq x_0^{out} \leq M, & x_0^{out} = x_1^{in},\ t_1^{in} = 0, & \text{(jump relation)} \\ x_1^{out} - x_1^{in} = r\delta_0, & t_1^{out} - t_1^{in} = \delta_0, & \text{(flow)} \\ t_1^{out} \leq T, & & \text{(invariant)} \\ x_1^{out} \geq x_{max}. & & \text{(jump relation)} \end{array} \tag{3}$$

■

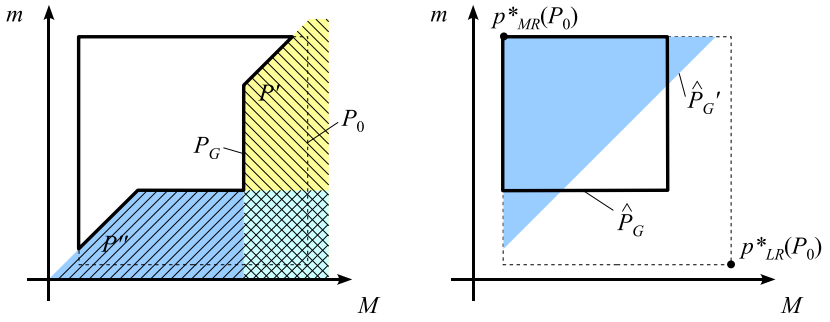
3 Parameter Synthesis Using Counterexamples

We consider the following *good parameters problem*: Given an LHA H and a rectangular parameter domain $P_0 \subseteq \mathbb{R}^m$, what is the largest set of parameter values $P_G \subseteq P_0$ for which the hybrid automaton is safe? Recalling that for a given parameter value p', the set of feasible counterexamples $\mathit{FCE}(H, p')$ is empty exactly if the system is safe, the goal is to compute

$$P_G = \{p' \in P_0 \mid \mathit{FCE}(H, p') = \emptyset\}. \tag{4}$$

We refer to P_G as the *good parameters* and to $P_B := P_0 \setminus P_G$ as the *bad parameters*. A straightforward solution to (4) is via reachability [1]. The set of reachable states $\mathit{Reach}(H)$ is obtained by computing successor states until a fixpoint is reached. Denoting projection onto the parameters (existential quantification over $\mathtt{X}$) with $\downarrow_{\mathtt{P}}$, the set of good parameters is

$$P_G = P_0 \setminus \big(\mathit{Reach}(H) \cap L_B \times \mathbb{R}^{n+m}\big)\downarrow_{\mathtt{P}} . \tag{5}$$



(a) good and bad values (b) underapproximations

Fig. 2. Parameters m and M with feasible paths (for fixed x_{max} and T)

If $Reach(H)$ is obtained as a finite boolean combination of linear constraints for each location, P_G can be computed exactly using Fourier-Motzkin elimination. In all but the most simple cases, however, this is prohibitively expensive for three reasons. Firstly, the reachability computation taking into account all parameter values is relatively expensive, since it includes behaviors that will later be excluded in the final solution. Secondly, the projection operation can be very expensive if there are many variables. Thirdly, the difference operation is very expensive if the projection operation produces a disjunction consisting of a large number of convex sets. In this paper, we try to find good parameters by checking individual counterexamples and removing from P_0 those parameters for which the counterexamples are feasible. Using projection (Fourier-Motzkin elimination) onto the parameters, and recalling the definition of FCE, (4) becomes

$$\begin{aligned} P_G &= P_0 \setminus \{p' \mid \exists \pi \in CE(H, P_0) : [\![PathCon(\pi, p')]\!] \neq \emptyset\} \\ &= P_0 \setminus \textstyle\bigcup_{\pi \in CE(H,P_0)} [\![PathCon(\pi, p)]\!]\downarrow_{\mathbb{P}} . \end{aligned} \tag{6}$$

Example 2. Recall the buffer tank from Fig. 1 and the counterexample $\pi = start1, err$ with path constraints (3). We can eliminate x_i^{in}, x_i^{out}, t_i^{in}, t_i^{out} from the path constraints to obtain $m \leq M \wedge M + rT \geq x_{max}$. For values of m, M, T, and x_{max} that satisfy these inequalities, shown as the shaded region P' in Fig. 2(a), the path constraints are feasible and the path is feasible. If we want the system to be safe, we must choose parameter values that violate these inequalities, i.e., make the path constraints infeasible, for the above as well as all other counterexamples (there are infinitely many). For $\pi = start2, err$ the path constraints yield $m \leq M \wedge m - rT \leq x_{min}$, shown as P'' in Fig. 2(a). We finally obtain $P_G = [\![m > M \vee (M + rT < x_{max} \wedge m - rT > x_{min})]\!]$, shown as a solid outline in Fig. 2(a), when taking into account all counterexamples. ■

The method for computing P_G suggested by (6) is conceptually similar to (5), and shares its problems: there may be lots of paths in $CE(H, P_0)$ (possibly infinitely many) and projection is very expensive when there are more than a few

variables. Recall that the dimension of the linear program as well as the number of constraints of $PathCon(\pi, p)$ increase linearly with the length of the counterexample. So the projection entails a cost that is exponential in the number of variables and the length of the counterexample. The difference operation also incurs a cost exponential in the number of parameters.

The exact solution being clearly too expensive, we use rectangular or octagonal overapproximations of the bad parameters, and carry out the difference operation on the overapproximation. For a set S of values for the variables of the path constraints, we write $OverAppr_{\mathtt{P}}(S)$ to denote one of the following overapproximations of $S{\downarrow_{\mathtt{P}}}$:

$$Box_{\mathtt{P}}(S) = \bigcap_{i=1,\ldots,m} \{p \mid \min_{(x,p')\in S} p'_i \leq p_i \leq \max_{(x,p'')\in S} p''_i\}, \tag{7}$$

$$Oct_{\mathtt{P}}(S) = \bigcap_{i,j=1,\ldots,m} \{p \mid \min_{(x,p')\in S} p'_i - p'_j \leq p_i - p_j \leq \max_{(x,p'')\in S} p''_i - p''_j\} \cap \bigcap_{i,j=1,\ldots,m} \{p \mid \min_{(x,p')\in S} p'_i + p'_j \leq p_i + p_j \leq \max_{(x,p'')\in S} p''_i + p''_j\} \tag{8}$$

These overapproximations are obtained by solving a linear program for each constraint: in total $2m$ programs for rectangular, and $2m^2$ for octagonal overapproximations, where m is the number of parameters. The cost of linear programming is polynomial in the number of variables of the path constraints, which is $2(|\pi|+1)n+m$. In total, the cost for obtaining an overapproximation for a single counterexample is still polynomial.

How these overapproximations should be applied in (6) depends on the dominating cost factor: lots of counterexamples to be checked, or lots of parameters. If we expect few counterexamples, we overapproximate the projection operation, but carry out the difference operation faithfully. We refer to this as *individual overapproximation*:

$$\hat{P}_G = P_0 \setminus \bigcup_{\pi \in CE(H,P_0)} OverAppr_{\mathtt{P}}([\![PathCon(\pi, p)]\!]). \tag{9}$$

For each counterexample, the cost of obtaining the overapproximation is polynomial in the number of parameters, since one linear program needs to be solved for each constraint in the overapproximation. The difference operation incurrs a cost that is exponential in the number of counterexamples, although we did not encounter such a worst case in practice. Consequently, this variant is suitable mainly when a small number of counterexamples is checked.

If the number of counterexamples is excessive, we overapproximate the union operation, which we refer to as *collective overapproximation*:

$$\hat{P}_G = P_0 \setminus OverAppr_{\mathtt{P}}(\bigcup_{\pi \in CE(H,P_0)} [\![PathCon(\pi, p)]\!]). \tag{10}$$

Here $OverAppr_{\mathtt{P}}$ is extended to unions of convex sets as follows. For each counterexample, we compute the bounds of the rectangular or octagonal overapproximation and take the worst case, so there is no explicit union operation. Again,

we need to solve a low number of linear programs, but here the complexity of the difference operation depends only on the number of parameters, not the number of counterexamples.

Example 3. Applying rectangular individual overapproximation to our buffer tank example, we obtain $\hat{P}_G$ as shown in Fig. 2(b). For rectangular collective overapproximation we get $\hat{P}_G = \emptyset$, i.e., we fail to find any good parameter at all. With octagonal individual overapproximation we incidentally obtain P_G exactly, for collective overapproximation $\hat{P}'_G$ as shown in Fig. 2(b).

In the general case, neither (9) nor (10) is *complete*, that is, (9) or (10) may return empty sets even though P_G is not empty. This makes the chances of finding a sufficiently good set of parameters look pretty slim. But in practice, many systems do not have an arbitrarily complex set of good parameters, but one with a particularly simple structure, where octagonal overapproximations turn out to be complete. We examine this special case closer in the following section.

4 Monotonic Parameters

It turns out that the set of parameters for which a given counterexample is feasible has a special form if the parameters occur only with one sign (either positive or negative) in the linear constraints defining the automaton. We show that the good parameters have a point that is most restrictive, and that any good point in the parameter space can be relaxed (tightened) toward that point. As a consequence, octagonal constraints are complete when counterexamples are overapproximated individually or collectively.

We call a constraint of the form (1) or (2) *positive in* $\mathtt{p_i}$ if $e_i > 0$, *negative in* $\mathtt{p_i}$ if $e_i < 0$, and *independent of* $\mathtt{p_i}$ if $e_i = 0$, where e_i is the coefficient of p_i from the vector e . We call a parameter $\mathtt{p_i}$ positive (negative) if for all $\pi \in CE(H, P_0)$ with $[\![PathCon(\pi, p)]\!]{\downarrow_{\mathtt{P}}} \neq \emptyset$, all active constraints in $PathCon(\pi, p)$ that are not independent of the parameter are positive (negative) in the parameter.[4] Intuitively, a parameter is negative if it is only relevant as an upper bound, and a positive parameter only as a lower bound. Let *psgn* be a function over the parameters with $psgn(\mathtt{p_i}) = 1$ (-1) if the LHA is positive (negative) in $\mathtt{p_i}$. We call the parameters *monotonic* and the LHA *parameter-monotonic* if all parameters are positive or negative. A syntactic sufficient condition for a parameter $\mathtt{p_i}$ being positive (negative) is that all constraints of invariants and jump relations are positive (negative) in $\mathtt{p_i}$. This is easy to see since the path constraints are made up of simple instantiations of the constraints of invariants, flows and jump relations, with the same coefficients for the parameters. Often, a parameter can be monotonic even though the syntactic condition for monotonicity is not fulfilled because, although the signs of the coefficients differ for some constraints, they are the same for all of the active constraints. In general, we may assume parameter-monotonicity and check for each counterexample whether this assumption is

[4] We call a constraint *active* if removing the constraint leads to a strictly larger set.

true. If it is not, we are no longer guaranteed to find a good parameter using overapproximations, although we may of course still try.

Example 4. In our buffer tank example, the parameters x_{max}, T are syntactically positive, x_{min} is syntactically negative, and m and M are neither. In the counterexample $\pi = \mathit{start1}, \mathit{err}$, whose path constraints are given in (3), m is positive and M is negative. It turns out that this is a useful assumption, even though m and M also occur with opposite sign on the transitions switching between $\mathit{filling}$ and $\mathit{draining}$. ■

The following results formalize our discussion of parameter-monotonicity. Let $\mathit{InfeasibleCone}_C(p')$ be defined for a set of constraints C as the set of p'' with $p''_i \geq p'_i$ if C is positive in $\mathtt{p_i}$, and $p''_i \leq p'_i$ if C is negative in $\mathtt{p_i}$. Let $\mathit{FeasibleCone}_C(p')$ be the set of p'' with $p''_i \leq p'_i$ if C is positive in $\mathtt{p_i}$, and $p''_i \geq p'_i$ if C is negative in $\mathtt{p_i}$.

Lemma 1. *Given a set of linear constraints C monotonic in all parameters, $[\![C(p')]\!] = \emptyset$ implies $[\![C(p'')]\!] = \emptyset$ for all $p'' \in \mathit{InfeasibleCone}_C(p')$. Symmetrically, $[\![C(p')]\!] \neq \emptyset$ implies $[\![C(p'')]\!] \neq \emptyset$ for all $p'' \in \mathit{FeasibleCone}_C(p')$.*

Proof. We give the proof for $\mathit{InfeasibleCone}_C(p')$; the proof for $\mathit{FeasibleCone}(p')$ is symmetric. Assume $[\![C(p')]\!] = \emptyset$, and the constraints in C are negative in $\mathtt{p_i}$, i.e., of the form $a^\intercal x + e^\intercal p \leq b$ with $e_i \leq 0$. Consider any $p'' \in \mathit{InfeasibleCone}_C(p')$. If $[\![C(p'')]\!] \neq \emptyset$, there exists some x' such that $a^\intercal x' + e^\intercal p'' \leq b$ for all constraints in C. We show that this contradicts the hypothesis. According to the definition of $\mathit{InfeasibleCone}_C(p')$, $e^\intercal p' \leq e^\intercal p''$, since $p''_i \leq p'_i$ and $e_i \leq 0$. Therefore $a^\intercal x' + e^\intercal p' \leq a^\intercal x' + e^\intercal p'' \leq b$, and (x', p') satisfies all constraints in C, which means $[\![C(p')]\!] \neq \emptyset$. □

Under the assumption that the LHA H is parameter-monotonic, a parameter $\mathtt{p_i}$ has the same sign, $\mathit{psgn}(\mathtt{p_i})$, in all path constraints that may occur in (6). Since $\mathit{InfeasibleCone}_C(p')$ is identical for all C with the same parameter sign, we may simply define $\mathit{InfeasibleCone}_H(p')$ over H using $\mathit{psgn}(\mathtt{p_i})$, and similarly with $\mathit{FeasibleCone}_H(p')$. Using the above result, we now show that if the LHA is parameter-monotonic, there is a *most restrictive* and a *least restrictive* combination of parameters out of any rectangular domain P, defined component-wise for parameter $\mathtt{p_i}$ as

$$p^*_{MR,i}(P) = \max_{p' \in P} \mathit{psgn}(\mathtt{p_i}) p'_i, \qquad p^*_{LR,i}(P) = \min_{p' \in P} \mathit{psgn}(\mathtt{p_i}) p'_i. \tag{11}$$

Example 5. Assuming that in the buffer tank example m is a positive and M a negative parameter, we get $p^*_{MR}(P_0)$ and $p^*_{LR}(P_0)$ shown in Fig. 2(b). ■

Proposition 1. *If an LHA H is parameter-monotonic, then for any $p \in P_0$*

$$\mathit{FCE}(H, p^*_{MR}(P_0)) \subseteq \mathit{FCE}(H, p) \subseteq \mathit{FCE}(H, p^*_{LR}(P_0)).$$

Proof. Consider any $\pi \in FCE(H, p^*_{MR}(P_0))$. By definition of FCE, π is a counterexample for which $[\![PathCon(\pi, p^*_{MR}(P_0))]\!] \neq \emptyset$. According to the definition of $p^*_{MR}(P_0)$, any $p \in P_0$ is in $FeasibleCone_H(p^*_{MR}(P_0))$. With Lemma 1 we get $[\![PathCon(\pi, p)]\!] \neq \emptyset$, and consequently $\pi \in FCE(H, p)$. The argument for p^*_{LR} is dual. □

It immediately follows from Prop. 1 that if we substitute the parameters in H with the value $p^*_{LR}(P)$ to obtain H', an LHA without parameters, $FCE(H, P) \subseteq FCE(H')$, i.e., H' is an overapproximation of H. Working with H' may be cheaper since it has less variables than H'.

Proposition 2. *If the parameters are monotonic and $p' \in P_G$, $p'' \in P_B$, then*

$$InfeasibleCone_H(p') \cap P_0 \subseteq P_G \quad \text{and} \quad FeasibleCone_H(p'') \cap P_0 \subseteq P_B.$$

Proof. Since $p' \in P_G$, $[\![PathCon(\pi, p')]\!] = \emptyset$ for all $\pi \in CE(H, P_0)$. According to Lemma 1, the same holds for any $p'' \in InfeasibleCone_H(p')$. Therefore $InfeasibleCone_H(p') \cap P_0 \subseteq P_G$. The argument for $FeasibleCone$ is symmetric. □

As a straightforward application of Prop. 2, we can use any feasible counterexamples to obtain an overapproximation of P_G. Similarly, we can use any infeasible counterexamples to obtain an underapproximation of P_G:

Proposition 3. *Given a parameter-monotonic LHA H, a set of z_{feas} parameter valuations $p^1, \ldots, p^{z_{feas}}$ such that there exists a $\pi^i \in FCE(H, p^i)$ for $i = 1, \ldots, z_{feas}$, and a set of z_{infeas} parameter valuations $\bar{p}^1, \ldots, \bar{p}^{z_{infeas}}$ such that $FCE(H, \bar{p}^j) = \emptyset$ for $j = 1, \ldots, z_{infeas}$, then*

$$P_0 \cap \bigcup_j InfeasibleCone_H(\bar{p}^j) \subseteq P_G \subseteq P_0 \setminus \bigcup_i FeasibleCone_H(p^i).$$

We now show that for a parameter-monotonic LHA with rectangular P_0 and a finite number of counterexamples, collective (and therefore also individual) overapproximation with octagonal constraints is complete, i.e., the overapproximation is empty iff $P_G = \emptyset$. This observation follows from the following:

Proposition 4. *If H is parameter-monotonic, $|CE(H)|$ is finite and P_0 is rectangular, $p^*_{MR}(P_0) \in Oct_{\mathtt{P}}(\bigcup_{\pi \in CE(H,P_0)} [\![PathCon(\pi, p) \wedge p \in P_0]\!])$ iff $p^*_{MR}(P_0) \in P_B$.*

Proof. (Sketch) If $p^*_{MR}(P_0) \in P_B$, it follows from Lemma 1, the definition of p^*_{MR} and Prop. 2 that $P_B = P_0$. Since P_0 is rectangular, the octagonal overapproximation of P_B is identical to P_0 and therefore to P_B, and contains $p^*_{MR}(P_0)$. If $p^*_{MR}(P_0) \notin P_B$ and there is a finite number of refining counterexamples, there exists at least one pair of parameters $\mathtt{p_i}, \mathtt{p_j}$ such that for all $\pi \in CE(H, P_0)$

$$max_{p' \in [\![PathCon(\pi,p)]\!]_{\downarrow \mathtt{P}} \cap P_0} psgn(\mathtt{p_i}) p'_i + psgn(\mathtt{p_j}) p'_j < p^*_{MR,i}(P_0) + p^*_{MR,j}(P_0),$$

since otherwise $p^*_{MR}(P_0)$ would be feasible for some path π. From (8) it follows that $p^*_{MR}(P_0) \notin Oct_{\mathtt{P}}(\bigcup_{\pi \in CE(H,P_0)} [\![PathCon(\pi) \wedge \pi \in CE(H, P_0))]\!]$. □

5 Counterexample-Guided Parameter Synthesis

We adapt the familiar *counterexample guided abstraction refinement* (CEGAR) loop [2] to parameter synthesis based on the results in the previous section. We define a CEGAR loop with the following operators:

- $\Pi' := \mathit{IniAbstr}(H, P)$ constructs a set of paths Π' with $\mathit{FCE}(H, P) \subseteq \Pi'$,
- $\pi := \mathit{SelectCE}(\Pi)$ selects a path in Π, given $\Pi \neq \emptyset$,
- $\Pi' := \mathit{RefineWithSpuriousCE}(\Pi, H, P, \pi)$ refines the set of paths Π using the system H and the spurious counterexample π, i.e., it produces a set of paths Π' such that $\Pi' \subseteq \Pi \setminus \{\pi\}$, and $\mathit{FCE}(H, P) \subseteq \Pi'$ (ensuring that the refinement never removes feasible counterexamples).

A number of CEGAR algorithms, including iterative relaxation abstraction [3], can be brought to this form. Note in general CEGAR constructs and refines a finite abstraction, which may take on various forms. We represent this abstraction as a set of paths Π to simplify and generalize the theoretical discussion, but assume that implementations model this set implicitly, say with an LHA or finite state machine. Algorithm 1 shows our adapted CEGAR loop. Its inputs are the system H and the initial parameter domain $\hat{P}_0$, from which the resulting good parameters are chosen. If it terminates, it outputs an underapproximation of the good parameters $\hat{P}_G$. A conventional CEGAR loop would terminate as soon as the feasibility test in line 6 evaluates to true, reporting π_i as a feasible counterexample. In our adaptation, we instead restrict the parameters such that the counterexample is infeasible and continue to search for other feasible counterexamples. This is accomplished by two additional operators:

- $P' := \mathit{MakeInfeasible}(C, P)$ returns a set $P' \subseteq P$ such that $[\![C]\!]{\downarrow_{\mathtt{P}}} \cap P' = \emptyset$. Note that the only such set might be $P' = \emptyset$,
- $P' := \mathit{AnalysisParameters}(P)$ returns a set of parameter values P' such that $\mathit{FCE}(H, P) \subseteq \mathit{FCE}(H, P')$.

The exact implementation of $\mathit{MakeInfeasible}(C, P)$ is $P \setminus [\![C]\!]{\downarrow_{\mathtt{P}}}$, but as discussed in the previous sections, underapproximations similar to (9) and (10) may be advisable. The operator *AnalysisParameters* is used to simplify the set of parameters used in the analysis of H. The condition is that this simplification does not drop any feasible counterexamples. As shown in Sect. 4, one may select a single point in P if the parameters are monotonic, i.e., use $\mathit{AnalysisParameters}(P) := p^*_{LR}(P)$. In that case, the parameters in H can be substituted by constants, thus reducing the number of variables in H. For the general case, a valid implementation is simply $\mathit{AnalysisParameters}(P) := P$.

On the basis of (6) it is straightforward to show that when Alg. 1 terminates, $H(p)$ is safe for any $p \in \hat{P}_G$:

Proposition 5. *If $\hat{P}_0 \supseteq P_0$ and Alg. 1 terminates, $\hat{P}_G \subseteq P_G$.*

If the initial abstraction contains all counterexamples and the implementation of $\mathit{MakeInfeasible}(C, P)$ is exact, $\hat{P}_G$ is exact as well:

Algorithm 1: Counterexample-Guided Parameter Synthesis

Input: LHA H with bad locations L_B, parameter domain $\hat{P}_0$
Output: $\hat{P}_G$ such that $\hat{P}_G \subseteq P_G$

1 $i := 0$;
2 $\tilde{P}_0 := \mathit{AnalysisParameters}(\hat{P}_0)$;
3 $\hat{\Pi}_0 := \mathit{IniAbstr}(H, \tilde{P}_0)$;
4 **while** $\hat{\Pi}_i \neq \emptyset \wedge \hat{P}_i \neq \emptyset$ **do**
5 $\pi_i := \mathit{SelectCE}(\hat{\Pi}_i)$;
6 **if** $[\![\mathit{PathCon}(\pi_i, p) \wedge p \in \hat{P}_i]\!] \neq \emptyset$ **then**
7 $\hat{P}_{i+1} := \mathit{MakeInfeasible}(\mathit{PathCon}(\pi_i, p), \hat{P}_i)$;
8 $\tilde{P}_{i+1} := \mathit{AnalysisParameters}(\hat{P}_{i+1})$;
9 **else**
10 $\hat{P}_{i+1} := \hat{P}_i$; $\tilde{P}_{i+1} := \tilde{P}_i$;
11 **end**
12 $\hat{\Pi}_{i+1} := \mathit{RefineWithSpuriousCE}(\hat{\Pi}_i, H, \tilde{P}_{i+1}, \pi_i)$;
13 $i := i + 1$;
14 **end**
15 $\hat{P}_G := \hat{P}_i$;

Proposition 6. *If $\hat{P}_0 \supseteq P_0$, Alg. 1 terminates, $\mathit{IniAbstr}(H, P_0) = \mathit{CE}(H, P_0)$, and $\mathit{MakeInfeasible}(C, P) = P \setminus [\![C]\!]\downarrow_{\mathbb{P}}$, $\hat{P}_G = P_G$.*

We now briefly present a way to check less paths for parameter-monotonic LHA. According to Prop. 1, we may substitute the parameters with the value $p^*_{LR}(P_0)$ to produce an overapproximation of $\mathit{CE}(H, P_0)$. Usually, there is a prohibitively large number of such paths. Instead, we wish to start with a small set of paths, and iteratively add further paths only when necessary. We run Alg. 1 with $\hat{P}_0 = P_0$ and $\mathit{AnalysisParameters}(P) := p^*_{MR}(P)$, to obtain with $\hat{P}_G$ an initial set of parameters checking the least number of paths possible. To check the remaining paths, we run Alg. 1 with $\hat{P}_0 = \hat{P}_G$ and $\mathit{AnalysisParameters}(P) := p^*_{LR}(P)$. Note that we can skip the initialization of $\hat{\Pi}_0$ in line 3 when in running Alg. 1 in step 2 above, and instead continue with $\hat{\Pi}_i$ from step 1.

We use the following two CEGAR implementations, and show experimental results in the next section. *Simple Discrete CEGAR* is a straightforward CEGAR algorithm based on a discrete abstraction of the hybrid system, somewhat similar to [6]. A set of paths is represented by a finite state machine (FSM). $\mathit{IniAbstr}(H, P)$ constructs a FSM $A_0 = (\mathit{Locs}, \Sigma, \rightarrow, \mathit{ini}, \mathit{final})$, where $\Sigma = \mathit{Locs} \times \mathit{Lab}$, and $\rightarrow = \{(l, (l, \alpha), l') \mid \exists \mu : (l, \alpha, \mu, l') \in \mathit{Trans}\}$. $\mathit{SelectCE}(A_i)$ returns a shortest word in the language of A_i. *RefineWithSpurious* works as follows. First, it reduces the counterexample $\pi = \alpha_0, \ldots, \alpha_{z-1}$ by finding the largest k and the smallest j such that the path constraints for $\alpha_k, \ldots, \alpha_j$ are infeasible (starting in the location l_k reached by $\alpha_0, \ldots, \alpha_{k-1}$). For FSM A and language L let $A - L$ denote removing the language L from the language of A. This is a standard automaton operation, implemented by first coding L by an automaton.

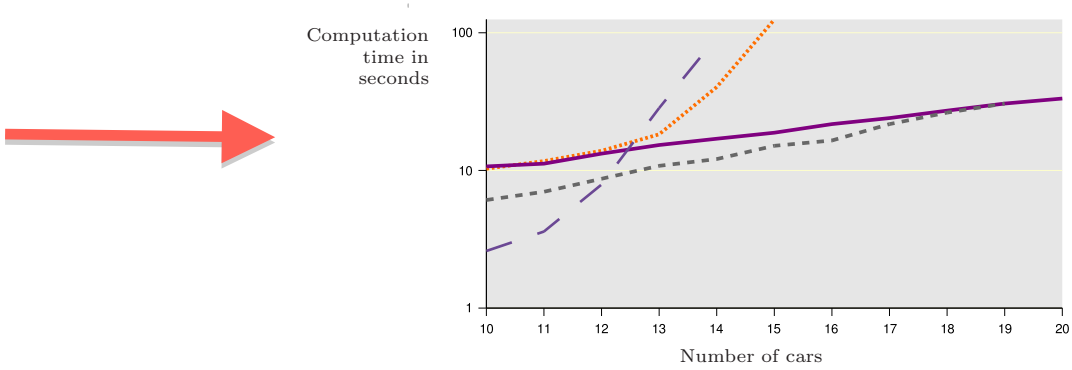


Fig. 3. Parameter synthesis for automated highway controller using standard reachability (wide dashed), simple discrete CEGAR with projection as in (6) (dotted) and with individual octagonal overapproximations as in (9) (solid), and IRA with individual octagonal overapproximations as in (9) (fine dashed)

Let $L_{pre} = \varepsilon$ if $k = 0$ and $L_{pre} = \Sigma^*$ otherwise, and $L_{post} = \varepsilon$ if $j = z - 1$ and $L_{post} = \Sigma^*$ otherwise, Then $A_{i+1} = A_i - L_{pre}(l_k, \alpha_k) \dots (l_j, \alpha_j) L_{post}$.

Iterative Relaxation Abstraction[3] is similar to Simple Discrete CEGAR, with $A_{i+1} = \mathit{RefineWithSpurious}(A_i, H, P, \pi)$ constructed as follows: First, find an irreducible infeasible subset (IIS) of $\mathit{PathCon}(\pi, p)$, say C. Let V be the variables (including parameters) with nonzero coefficients in C. Let $\mathit{localize}(H, V)$ be the automaton obtained from H by removing all constraints involving variables not in V. Let A' be the language of $\mathit{localize}(H, V)$, which is semi-computable using reachability techniques. Then $A_{i+1} = A_i \cap A' \setminus \pi$.

6 Experimental Results

We consider the model of a central arbiter for an automated highway, roughly similar to the one in [7], and verify that no two vehicles on the automated highway collide with each other. The arbiter provides an allowed range $[a, b]$ for the velocity for each vehicle. When two vehicles come within a distance d_{std} of each other, the arbiter asks the faster car and all behind it to reduce the speed to a' and the slower car and all in front of it to increase the speed to b'. When the distance between the two vehicles involved exceeds d_{normal}, the arbiter goes back to normal. The cars are considered to have crashed if their distance is below c. The LHA model for n cars has n continuous state variables and the parameters d_{std} and d_{normal}. The number of counterexamples is infinite, since the controller can cycle infinitely between normal and recovery mode before a crash occurs.

We consider the constants $a = 40$, $b = 60$, $a' = 41$, $b' = 80$, $c = 0.002$. Using CEGAR and octagonal overapproximations, we synthesize the solution $d_{std} \leq d_{normal} \leq 10 \wedge d_{std} > 0.002$. The plot of the log of the time taken vs. the number of cars in Fig. 3 shows that the cost of standard reachability is double-exponential, while using CEGAR (simple discrete or IRA) it is exponential with

a low factor (actually due to the time it takes to compose the system, not the analysis itself). For illustration, we also include the time it takes to obtain the exact solution as in (6) using CEGAR and Fourier-Motzkin elimination – it is also double exponential. These results were obtained on a 1.8 GHz AMD Opteron processor with 16 GB RAM running 32 bit code under Linux. For linear programming we use GLPK [8] with exact arithmetic for simple discrete CEGAR, and CPLEX [9] with floating point arithmetic for IRA.

7 Conclusions

This paper proposes a method for using counterexamples to guide the construction of a set of good parameters for parameterized LHA. The proposed procedure extends the philosophy of CEGAR for verification to a class of design problems. The method is complete when the parameters are monotonic. The effectiveness of the approach is illustrated for an example of an automatic highway controller.

The implications of parameter-monotonicity merit further investigation. If an LHA is not monotonic in the parameters, it can be brought to monotonic form by replacing each parameter $\mathtt{p_i}$ that occurs with both signs with $\mathtt{p_i^+}$ where it occurs with positive sign and $\mathtt{p_i^-}$ where it occurs with negative sign. If P'_G is the set of good parameters for the modified LHA, the set of good parameters for the original system is given by $P_G = P'_G \cap [\![p_i^+ = p_i^-]\!]$. Further research is needed to determine whether or not this leads to any computational advantage for LHA that are not parameter-monotonic.

References

1. Thomas A. Henzinger and Howard Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In *Formal Methods for Industrial Applications*, number 1165 in LNCS, pages 265–282. Springer Verlag, 1996.
2. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV'00*, pages 154–169, London, UK, 2000. Springer.
3. S.K. Jha, B.H. Krogh, J.E. Weimer, and E.M. Clarke. Reachability for linear hybrid automata using iterative relaxation abstraction. In A. Bemporad, A. Bicchi, and G.C. Buttazzo, editors, *HSCC*, volume 4416 of *LNCS*. Springer, 2007.
4. Thomas A. Henzinger. The theory of hybrid automata. In *Proc. 11th Annual IEEE Symposium on Logic in Computer Science, LICS'96, New Brunswick, New Jersey, 27-30 July 1996*, pages 278–292. IEEE Computer Society Press, 1996.
5. X. Li, S. K. Jha, and L. Bu. Towards an Efficient Path-Oriented Tool for Bounded Reachability analysis of Linear Hybrid Systems using Linear Programming. In *BMC'06: Proceedings of the Workshop on Bounded Model Checking*, 2006.
6. Marc Segelken. Abstraction and counterexample-guided construction of ω-automata for model checking of step-discrete linear hybrid models. In W. Damm and H. Hermanns, editors, *CAV*, volume 4590 of *LNCS*, pages 433–448. Springer, 2007.
7. R. Horowitz and P. Varaiya. Control design of an automated highway system. *Proc. IEEE*, 88:913–925, July 2000.
8. GNU Linear Programming Kit, v.4.17, 2007. `http://www.gnu.org/software/glpk`.
9. ILOG. `http://www.ilog.com/products/cplex/product/simplex.cfm`, 2007.

A Counterexample-Guided Approach to Parameter Synthesis for Linear Hybrid Automata

Goran Frehse[1], Sumit Kumar Jha[2], and Bruce H. Krogh[3]

[1] Verimag (UJF-CNRS-INPG), 2, av. de Vignate, 38610 Gières, France
goran.frehse@imag.fr
[2] Computer Science Department, Carnegie Mellon University
[3] ECE Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
jha@cs.cmu.edu, krogh@ece.cmu.edu

Abstract. Our goal is to find the set of parameters for which a given linear hybrid automaton does not reach a given set of bad states. The problem is known to be semi-solvable (if the algorithm terminates the result is correct) by introducing the parameters as state variables and computing the set of reachable states. This is usually too expensive, however, and in our experiments only possible for very simple systems with few parameters. We propose an adaptation of counterexample-guided abstraction refinement (CEGAR) with which one can obtain an underapproximation of the set of good parameters using linear programming. The adaptation is generic and can be applied on top of any CEGAR method where the counterexamples correspond to paths in the concrete system. For each counterexample, the cost incurred by underapproximating the parameters is polynomial in the number of variables, parameters, and the length of counterexample. We identify a syntactic condition for which the approach is complete in the sense that the underapproximation is empty only if the problem has no solution. Experimental results are provided for two CEGAR methods, a simple discrete version and iterative relaxation abstraction (IRA), both of which show a drastic improvement in performance compared to standard reachability.

1 Introduction

The admissible behaviors of linear hybrid automata (LHA) are determined by sets of linear constraints. The parameters in these constraints represent either physical constants or values chosen by the designer. When the LHA does not satisfy the design specifications, the latter constraints can be adjusted to eliminate the undesirable behaviors. This paper concerns this design problem in the context of reachability specifications: Given a parameterized LHA, determine the set of design parameters, called *good parameters*, for which no bad locations can be reached.

The parameter design problem for LHA was formulated and solved by Henzinger et al. [1], but the proposed solution is tractable for only very simple

systems with few parameters. This paper concerns the extension of verification techniques to solve the LHA parameter design problem. Our approach leverages the fact that the feasibility of a given counterexample path corresponds to the satisfiability of a set of linear constraints over instantiations of the initial and final values of the continuous variables in each location along the path, along with variables representing the duration of the continuous state trajectory in each location. Although this observation does not make the parameter design problem tractable, because projection of these constraints into the parameter space is computationally complex and there can be in general an infinite number of counterexample paths, it does lead to a set of heuristics that make it possible to efficiently compute underapproximations of the set of good parameters.

The heuristics we propose are integrated into *counterexample guided abstraction refinement* (CEGAR) [2]. In a standard CEGAR loop, a discrete abstraction of the system is used to find a counterexample, which is a path from the initial states to states considered bad. In a feasibility check, it is then verified whether this path corresponds to a behavior of the concrete hybrid system or whether it was a spurious product of the abstraction. If it is spurious, the abstraction is refined and the loop repeats. If the counterexample corresponds to a concrete behavior, the system is unsafe. Our adaptation consists of replacing the feasibility check with an operator that obtains constraints on the parameters that *make* the counterexample infeasible. If all counterexamples have been eliminated, the resulting constraints describe a set of parameters for which the system is safe. The make-infeasible operator can be implemented approximatively using linear programming, e.g., obtaining rectangular or octagonal underapproximations of the good parameters. Its complexity for each counterexample is polynomial in the number of variables, parameters and the length of the counterexample, compared to exponential complexity of an exact solution. Depending on whether the number of counterexamples or the number of parameters is the dominating cost factor, we apply the underapproximation to each path individually or collectively on sets of paths.

In the general case, the underapproximation may produce an empty set even though good parameters exist. We identify a condition we call *parameter-monotonicity* (intuitively, when parameters function either as lower or upper bounds but not both), under which octagonal approximations are sufficient to prevent this from happening.

The entire approach is generic in the sense that it can be applied to any CEGAR loop in which the counterexamples correspond to paths in the concrete system (as opposed to sets of paths or transitions). It suffices to replace the feasibility check with the make-infeasible operator. We provide experimental results for two different CEGAR implementations: a simple variant of standard discrete CEGAR, and iterative relaxation abstraction (IRA) [3]. Compared to the traditional way of synthesizing parameters using reachability as in [1], we observe a dramatic improvement in speed.

The following section defines the class of LHA with parameters studied in this paper. Section 3 describes the role of counterexamples in defining the set

of good parameters. Section 4 discusses the special case when the parameters are *monotonic*, which means each parameter serves as either an upper or lower bound throughout the LHA. Section 5 presents a general counterexample-guided procedure for computing sets of good parameters and Section 6 presents experimental results for two implementations of the procedure. The concluding section discusses directions for further research.

2 Linear Hybrid Automata with Parameters

We consider *linear hybrid automata* (LHA) [4] with explicit parameter variables. An LHA $H = (Var, Lab, Loc, Inv, Flow, Trans, ini)$ consists of:

- A finite set of real-valued *variables* $Var = \mathtt{X} \cup \mathtt{P}$, where $\mathtt{X} = \{\mathtt{x_1}, \ldots, \mathtt{x_n}\}$ are the *continuous state variables* and $\mathtt{P} = \{\mathtt{p_1}, \ldots, \mathtt{p_m}\}$ are the *parameters*, which remain constant. We denote the values of variables with $x = (x_1, \ldots, x_n)^\intercal$ and $p = (p_1, \ldots, p_m)^\intercal$.
- A finite set of *labels* Lab.
- A finite set of *locations* Loc. A *state* (l, x, p) of the automaton consists of a location $l \in Loc$ and real values $(x, p) \in \mathbb{R}^{n+m}$ for each of the variables.
- For each location l, $Inv(l) \subseteq \mathbb{R}^{n+m}$ is the set of admissible values of the variables in the location.
- $Flow(l) \subseteq \mathbb{R}^n$ is the set of possible time derivatives $(\dot{x}_1, \ldots, \dot{x}_n)^\intercal$; the derivatives of the parameters are implicitly zero.
- A finite set of *transitions* $Trans \subseteq Loc \times Lab \times 2^{\mathbb{R}^{2n+m}} \times Loc$. A transition indicates that the system state may jump instantaneously from the transition *source state* (l, x, p) to the transition *target state* (l', x', p) if $(x, p, x') \in \mu$, where $\mu \subseteq \mathbb{R}^{2n+m}$ is the transition's *jump relation*. We desire a unique correspondence between sequences of transitions and sequences of labels, so we require that any location l has at most one outgoing transition for each label (the general case can be brought to this form by adding labels and renaming).
- A location $ini \in Loc$ is designated as *initial location* from which all behaviors must start.

The sets $Inv(l)$ and $Flow(l)$ are specified by conjunctions of linear constraints

$$a^\intercal x + e^\intercal p \leq b, \quad \text{respectively} \quad a^\intercal \dot{x} \leq b, \tag{1}$$

where a, e are vectors of integer coefficients and b is an integer. The jump relation μ of a transition is specified by a conjunction of linear constraints of the form

$$a^\intercal x + e^\intercal p + a'^\intercal x' \leq b, \tag{2}$$

where x denotes the values of the variables before the jump, and x' denotes the values after; the values of the parameters do not change. Given a conjunction C of linear constraints over $\mathtt{X}$ and $\mathtt{P}$, we write $[\![C]\!]$ to denote the set of values of (x, p) (a polyhedron) for which all of the constraints are satisfied. We write

$C(p')$ to denote the constraints obtained by substituting p with the values in p', and call $C(p')$ *infeasible* if $[\![C(p')]\!] = \emptyset$.

We define the semantics of a LHA in terms of *feasible paths* for a given parameter value p. This is consistent with the semantics in [4] but reformulated to simplify the use of linear programming. A *path* $\pi = \alpha_0\alpha_1 \ldots \alpha_{z-1}$ is a finite sequence of labels α_i such that the sequence of transitions $(l_i, \alpha_i, \mu_i, l'_i)$ satisfies $l_0 = ini$ and $l'_i = l_{i+1}$ for $i = 0, \ldots, z-1$ (this defines l_z to be the target state of the last transition). The path is *feasible* for a given p if there exist vectors $x_j^{in}, x_j^{out} \in \mathbb{R}^n$ and scalars $\delta_j \in \mathbb{R}$ for $j = 0, \ldots, z$ such that

- $(x_j^{in}, p), (x_j^{out}, p) \in Inv(l_j)$,
- $(x_j^{out} - x_j^{in})/\delta_j \in Flow(l_j)$,
- for $j < z$, $(x_j^{out}, p, x_{j+1}^{in}) \in \mu_j$.

In the above sequence, (l_j, x_j^{in}, p) is the state in which the automaton enters location l_j, and (l_j, x_j^{out}, p) is the state after letting time elapse for δ_j units. If $j < z$, the automaton leaves l_j via the transition identified by α_j, and jumps to the state $(l_{j+1}, x_{j+1}^{in}, p)$. For a given path the above constraints can be written as linear constraints over the $2z(n+1)+m$ variables of x_j^{in}, x_j^{out}, δ_j and p. We call these *path constraints* and denote them by $PathCon(\pi, p)$. Expressed in terms of the path constraints, a counterexample is feasible if $[\![PathCon(\pi, p)]\!] \neq \emptyset$, which can be decided using efficient linear programming techniques [5].

In this paper, we consider reachability problems for LHA. A location l is said to be *reachable* if there is a feasible path $\pi = \alpha_0\alpha_1 \ldots \alpha_{z-1}$ with $l_z = l$. An LHA H is said to be *safe* if none of the locations in a given set of *bad locations* L_B is reachable. We call a path to a bad location a *counterexample*, and write $CE(H, p)$ for the (possibly infinite) set of counterexamples in H for the parameter value p. Let $FCE(H, p)$ denote the set of *feasible counterexamples* in H. The system is safe if and only if $FCE(H, p)$ is empty, i.e., there are no counterexamples or none of the existing counterexamples is feasible. The extension of $CE(H, p)$ and $FCE(H, p)$ to sets of parameter values is straightforward. While we have made some restrictions to our LHA (unique labels) and the reachability problem (unsafe locations), it is straightforward to bring the general problem (unsafe states) to this form by relabeling transitions and introducing an error location reachable by transitions from the unsafe states. We will use the following example throughout the paper:

Example 1. Consider a buffer tank with steady inflow and with a controllable outlet valve resulting in a net level increase $\dot{x} = r$ if the valve is closed, and a net decrease $\dot{x} = -r$ if it is open. A controller is supposed to keep the level between x_{min} and x_{max}. The controller never waits longer than time T to check the level x, and opens (closes) the valve when $x > M$ ($x < m$).

The LHA model H_{tank} of the controlled system is shown in Fig. 1, where for simplicity jump constraints of the form $x' = x$ have been omitted. H_{tank} has the parameters m, M, T, x_{min}, and x_{max}. We assume r to be a given constant (parameters are not allowed in the flows). The forbidden location is *error*.

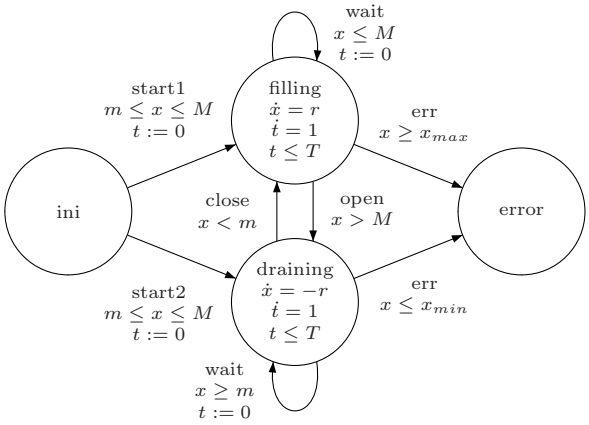


Fig. 1. LHA H_{tank} for the controlled tank example with parameters m, M, T, x_{min}, and x_{max}; r is a given constant since LHA-parameters cannot bound derivatives

H_{tank} has infinitely many counterexamples, a shortest of them being $\pi = \mathit{start1}, \mathit{err}$, which covers the locations ini, $\mathit{filling}$ and error, and has the following path constraints (some irrelevant ones are omitted):

$$\begin{array}{rrl} m \leq x_0^{out} \leq M, & x_0^{out} = x_1^{in},\ t_1^{in} = 0, & \text{(jump relation)} \\ x_1^{out} - x_1^{in} = r\delta_0, & t_1^{out} - t_1^{in} = \delta_0, & \text{(flow)} \\ t_1^{out} \leq T, & & \text{(invariant)} \\ x_1^{out} \geq x_{max}. & & \text{(jump relation)} \end{array} \tag{3}$$

■

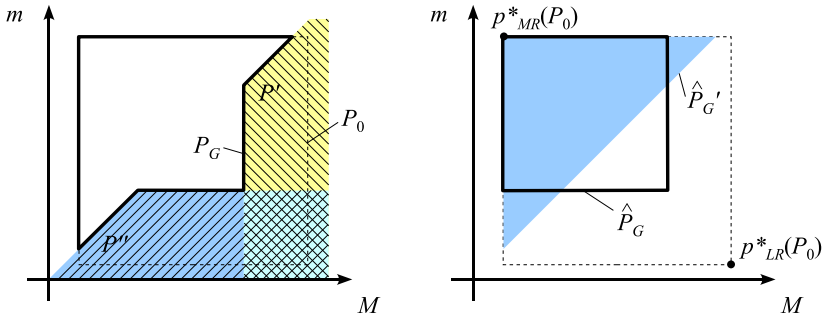
3 Parameter Synthesis Using Counterexamples

We consider the following *good parameters problem*: Given an LHA H and a rectangular parameter domain $P_0 \subseteq \mathbb{R}^m$, what is the largest set of parameter values $P_G \subseteq P_0$ for which the hybrid automaton is safe? Recalling that for a given parameter value p', the set of feasible counterexamples $\mathit{FCE}(H, p')$ is empty exactly if the system is safe, the goal is to compute

$$P_G = \{p' \in P_0 \mid \mathit{FCE}(H, p') = \emptyset\}. \tag{4}$$

We refer to P_G as the *good parameters* and to $P_B := P_0 \setminus P_G$ as the *bad parameters*. A straightforward solution to (4) is via reachability [1]. The set of reachable states $\mathit{Reach}(H)$ is obtained by computing successor states until a fixpoint is reached. Denoting projection onto the parameters (existential quantification over $\mathtt{X}$) with $\downarrow_{\mathtt{P}}$, the set of good parameters is

$$P_G = P_0 \setminus \big(\mathit{Reach}(H) \cap L_B \times \mathbb{R}^{n+m}\big)\downarrow_{\mathtt{P}} . \tag{5}$$



(a) good and bad values (b) underapproximations

Fig. 2. Parameters m and M with feasible paths (for fixed x_{max} and T)

If $Reach(H)$ is obtained as a finite boolean combination of linear constraints for each location, P_G can be computed exactly using Fourier-Motzkin elimination. In all but the most simple cases, however, this is prohibitively expensive for three reasons. Firstly, the reachability computation taking into account all parameter values is relatively expensive, since it includes behaviors that will later be excluded in the final solution. Secondly, the projection operation can be very expensive if there are many variables. Thirdly, the difference operation is very expensive if the projection operation produces a disjunction consisting of a large number of convex sets. In this paper, we try to find good parameters by checking individual counterexamples and removing from P_0 those parameters for which the counterexamples are feasible. Using projection (Fourier-Motzkin elimination) onto the parameters, and recalling the definition of FCE, (4) becomes

$$\begin{aligned} P_G &= P_0 \setminus \{p' \mid \exists \pi \in CE(H, P_0) : [\![PathCon(\pi, p')]\!] \neq \emptyset\} \\ &= P_0 \setminus \textstyle\bigcup_{\pi \in CE(H,P_0)} [\![PathCon(\pi, p)]\!]\downarrow_{\mathbb{P}} . \end{aligned} \tag{6}$$

Example 2. Recall the buffer tank from Fig. 1 and the counterexample $\pi = start1, err$ with path constraints (3). We can eliminate x_i^{in}, x_i^{out}, t_i^{in}, t_i^{out} from the path constraints to obtain $m \leq M \wedge M + rT \geq x_{max}$. For values of m, M, T, and x_{max} that satisfy these inequalities, shown as the shaded region P' in Fig. 2(a), the path constraints are feasible and the path is feasible. If we want the system to be safe, we must choose parameter values that violate these inequalities, i.e., make the path constraints infeasible, for the above as well as all other counterexamples (there are infinitely many). For $\pi = start2, err$ the path constraints yield $m \leq M \wedge m - rT \leq x_{min}$, shown as P'' in Fig. 2(a). We finally obtain $P_G = [\![m > M \vee (M + rT < x_{max} \wedge m - rT > x_{min})]\!]$, shown as a solid outline in Fig. 2(a), when taking into account all counterexamples. ■

The method for computing P_G suggested by (6) is conceptually similar to (5), and shares its problems: there may be lots of paths in $CE(H, P_0)$ (possibly infinitely many) and projection is very expensive when there are more than a few

variables. Recall that the dimension of the linear program as well as the number of constraints of $PathCon(\pi, p)$ increase linearly with the length of the counterexample. So the projection entails a cost that is exponential in the number of variables and the length of the counterexample. The difference operation also incurs a cost exponential in the number of parameters.

The exact solution being clearly too expensive, we use rectangular or octagonal overapproximations of the bad parameters, and carry out the difference operation on the overapproximation. For a set S of values for the variables of the path constraints, we write $OverAppr_{\mathtt{P}}(S)$ to denote one of the following overapproximations of $S{\downarrow_{\mathtt{P}}}$:

$$Box_{\mathtt{P}}(S) = \bigcap_{i=1,\ldots,m} \{p \mid \min_{(x,p')\in S} p'_i \leq p_i \leq \max_{(x,p'')\in S} p''_i\}, \tag{7}$$

$$Oct_{\mathtt{P}}(S) = \bigcap_{i,j=1,\ldots,m} \{p \mid \min_{(x,p')\in S} p'_i - p'_j \leq p_i - p_j \leq \max_{(x,p'')\in S} p''_i - p''_j\} \cap \bigcap_{i,j=1,\ldots,m} \{p \mid \min_{(x,p')\in S} p'_i + p'_j \leq p_i + p_j \leq \max_{(x,p'')\in S} p''_i + p''_j\} \tag{8}$$

These overapproximations are obtained by solving a linear program for each constraint: in total $2m$ programs for rectangular, and $2m^2$ for octagonal overapproximations, where m is the number of parameters. The cost of linear programming is polynomial in the number of variables of the path constraints, which is $2(|\pi|+1)n+m$. In total, the cost for obtaining an overapproximation for a single counterexample is still polynomial.

How these overapproximations should be applied in (6) depends on the dominating cost factor: lots of counterexamples to be checked, or lots of parameters. If we expect few counterexamples, we overapproximate the projection operation, but carry out the difference operation faithfully. We refer to this as *individual overapproximation*:

$$\hat{P}_G = P_0 \setminus \bigcup_{\pi \in CE(H,P_0)} OverAppr_{\mathtt{P}}([\![PathCon(\pi, p)]\!]). \tag{9}$$

For each counterexample, the cost of obtaining the overapproximation is polynomial in the number of parameters, since one linear program needs to be solved for each constraint in the overapproximation. The difference operation incurrs a cost that is exponential in the number of counterexamples, although we did not encounter such a worst case in practice. Consequently, this variant is suitable mainly when a small number of counterexamples is checked.

If the number of counterexamples is excessive, we overapproximate the union operation, which we refer to as *collective overapproximation*:

$$\hat{P}_G = P_0 \setminus OverAppr_{\mathtt{P}}(\bigcup_{\pi \in CE(H,P_0)} [\![PathCon(\pi, p)]\!]). \tag{10}$$

Here $OverAppr_{\mathtt{P}}$ is extended to unions of convex sets as follows. For each counterexample, we compute the bounds of the rectangular or octagonal overapproximation and take the worst case, so there is no explicit union operation. Again,

we need to solve a low number of linear programs, but here the complexity of the difference operation depends only on the number of parameters, not the number of counterexamples.

Example 3. Applying rectangular individual overapproximation to our buffer tank example, we obtain $\hat{P}_G$ as shown in Fig. 2(b). For rectangular collective overapproximation we get $\hat{P}_G = \emptyset$, i.e., we fail to find any good parameter at all. With octagonal individual overapproximation we incidentally obtain P_G exactly, for collective overapproximation $\hat{P}'_G$ as shown in Fig. 2(b).

In the general case, neither (9) nor (10) is *complete*, that is, (9) or (10) may return empty sets even though P_G is not empty. This makes the chances of finding a sufficiently good set of parameters look pretty slim. But in practice, many systems do not have an arbitrarily complex set of good parameters, but one with a particularly simple structure, where octagonal overapproximations turn out to be complete. We examine this special case closer in the following section.

4 Monotonic Parameters

It turns out that the set of parameters for which a given counterexample is feasible has a special form if the parameters occur only with one sign (either positive or negative) in the linear constraints defining the automaton. We show that the good parameters have a point that is most restrictive, and that any good point in the parameter space can be relaxed (tightened) toward that point. As a consequence, octagonal constraints are complete when counterexamples are overapproximated individually or collectively.

We call a constraint of the form (1) or (2) *positive in* $\mathtt{p_i}$ if $e_i > 0$, *negative in* $\mathtt{p_i}$ if $e_i < 0$, and *independent of* $\mathtt{p_i}$ if $e_i = 0$, where e_i is the coefficient of p_i from the vector e . We call a parameter $\mathtt{p_i}$ positive (negative) if for all $\pi \in CE(H, P_0)$ with $[\![PathCon(\pi, p)]\!]{\downarrow_{\mathtt{P}}} \neq \emptyset$, all active constraints in $PathCon(\pi, p)$ that are not independent of the parameter are positive (negative) in the parameter.[4] Intuitively, a parameter is negative if it is only relevant as an upper bound, and a positive parameter only as a lower bound. Let *psgn* be a function over the parameters with $psgn(\mathtt{p_i}) = 1$ (-1) if the LHA is positive (negative) in $\mathtt{p_i}$. We call the parameters *monotonic* and the LHA *parameter-monotonic* if all parameters are positive or negative. A syntactic sufficient condition for a parameter $\mathtt{p_i}$ being positive (negative) is that all constraints of invariants and jump relations are positive (negative) in $\mathtt{p_i}$. This is easy to see since the path constraints are made up of simple instantiations of the constraints of invariants, flows and jump relations, with the same coefficients for the parameters. Often, a parameter can be monotonic even though the syntactic condition for monotonicity is not fulfilled because, although the signs of the coefficients differ for some constraints, they are the same for all of the active constraints. In general, we may assume parameter-monotonicity and check for each counterexample whether this assumption is

[4] We call a constraint *active* if removing the constraint leads to a strictly larger set.

true. If it is not, we are no longer guaranteed to find a good parameter using overapproximations, although we may of course still try.

Example 4. In our buffer tank example, the parameters x_{max}, T are syntactically positive, x_{min} is syntactically negative, and m and M are neither. In the counterexample $\pi = \mathit{start1}, \mathit{err}$, whose path constraints are given in (3), m is positive and M is negative. It turns out that this is a useful assumption, even though m and M also occur with opposite sign on the transitions switching between $\mathit{filling}$ and $\mathit{draining}$. ■

The following results formalize our discussion of parameter-monotonicity. Let $\mathit{InfeasibleCone}_C(p')$ be defined for a set of constraints C as the set of p'' with $p''_i \geq p'_i$ if C is positive in $\mathtt{p_i}$, and $p''_i \leq p'_i$ if C is negative in $\mathtt{p_i}$. Let $\mathit{FeasibleCone}_C(p')$ be the set of p'' with $p''_i \leq p'_i$ if C is positive in $\mathtt{p_i}$, and $p''_i \geq p'_i$ if C is negative in $\mathtt{p_i}$.

Lemma 1. *Given a set of linear constraints C monotonic in all parameters, $[\![C(p')]\!] = \emptyset$ implies $[\![C(p'')]\!] = \emptyset$ for all $p'' \in \mathit{InfeasibleCone}_C(p')$. Symmetrically, $[\![C(p')]\!] \neq \emptyset$ implies $[\![C(p'')]\!] \neq \emptyset$ for all $p'' \in \mathit{FeasibleCone}_C(p')$.*

Proof. We give the proof for $\mathit{InfeasibleCone}_C(p')$; the proof for $\mathit{FeasibleCone}(p')$ is symmetric. Assume $[\![C(p')]\!] = \emptyset$, and the constraints in C are negative in $\mathtt{p_i}$, i.e., of the form $a^\intercal x + e^\intercal p \leq b$ with $e_i \leq 0$. Consider any $p'' \in \mathit{InfeasibleCone}_C(p')$. If $[\![C(p'')]\!] \neq \emptyset$, there exists some x' such that $a^\intercal x' + e^\intercal p'' \leq b$ for all constraints in C. We show that this contradicts the hypothesis. According to the definition of $\mathit{InfeasibleCone}_C(p')$, $e^\intercal p' \leq e^\intercal p''$, since $p''_i \leq p'_i$ and $e_i \leq 0$. Therefore $a^\intercal x' + e^\intercal p' \leq a^\intercal x' + e^\intercal p'' \leq b$, and (x', p') satisfies all constraints in C, which means $[\![C(p')]\!] \neq \emptyset$. □

Under the assumption that the LHA H is parameter-monotonic, a parameter $\mathtt{p_i}$ has the same sign, $\mathit{psgn}(\mathtt{p_i})$, in all path constraints that may occur in (6). Since $\mathit{InfeasibleCone}_C(p')$ is identical for all C with the same parameter sign, we may simply define $\mathit{InfeasibleCone}_H(p')$ over H using $\mathit{psgn}(\mathtt{p_i})$, and similarly with $\mathit{FeasibleCone}_H(p')$. Using the above result, we now show that if the LHA is parameter-monotonic, there is a *most restrictive* and a *least restrictive* combination of parameters out of any rectangular domain P, defined component-wise for parameter $\mathtt{p_i}$ as

$$p^*_{MR,i}(P) = \max_{p' \in P} \mathit{psgn}(\mathtt{p_i}) p'_i, \qquad p^*_{LR,i}(P) = \min_{p' \in P} \mathit{psgn}(\mathtt{p_i}) p'_i. \tag{11}$$

Example 5. Assuming that in the buffer tank example m is a positive and M a negative parameter, we get $p^*_{MR}(P_0)$ and $p^*_{LR}(P_0)$ shown in Fig. 2(b). ■

Proposition 1. *If an LHA H is parameter-monotonic, then for any $p \in P_0$*

$$\mathit{FCE}(H, p^*_{MR}(P_0)) \subseteq \mathit{FCE}(H, p) \subseteq \mathit{FCE}(H, p^*_{LR}(P_0)).$$

Proof. Consider any $\pi \in FCE(H, p^*_{MR}(P_0))$. By definition of FCE, π is a counterexample for which $[\![PathCon(\pi, p^*_{MR}(P_0))]\!] \neq \emptyset$. According to the definition of $p^*_{MR}(P_0)$, any $p \in P_0$ is in $FeasibleCone_H(p^*_{MR}(P_0))$. With Lemma 1 we get $[\![PathCon(\pi, p)]\!] \neq \emptyset$, and consequently $\pi \in FCE(H, p)$. The argument for p^*_{LR} is dual. □

It immediately follows from Prop. 1 that if we substitute the parameters in H with the value $p^*_{LR}(P)$ to obtain H', an LHA without parameters, $FCE(H, P) \subseteq FCE(H')$, i.e., H' is an overapproximation of H. Working with H' may be cheaper since it has less variables than H'.

Proposition 2. *If the parameters are monotonic and $p' \in P_G$, $p'' \in P_B$, then*

$$InfeasibleCone_H(p') \cap P_0 \subseteq P_G \quad \text{and} \quad FeasibleCone_H(p'') \cap P_0 \subseteq P_B.$$

Proof. Since $p' \in P_G$, $[\![PathCon(\pi, p')]\!] = \emptyset$ for all $\pi \in CE(H, P_0)$. According to Lemma 1, the same holds for any $p'' \in InfeasibleCone_H(p')$. Therefore $InfeasibleCone_H(p') \cap P_0 \subseteq P_G$. The argument for $FeasibleCone$ is symmetric. □

As a straightforward application of Prop. 2, we can use any feasible counterexamples to obtain an overapproximation of P_G. Similarly, we can use any infeasible counterexamples to obtain an underapproximation of P_G:

Proposition 3. *Given a parameter-monotonic LHA H, a set of z_{feas} parameter valuations $p^1, \ldots, p^{z_{feas}}$ such that there exists a $\pi^i \in FCE(H, p^i)$ for $i = 1, \ldots, z_{feas}$, and a set of z_{infeas} parameter valuations $\bar{p}^1, \ldots, \bar{p}^{z_{infeas}}$ such that $FCE(H, \bar{p}^j) = \emptyset$ for $j = 1, \ldots, z_{infeas}$, then*

$$P_0 \cap \bigcup_j InfeasibleCone_H(\bar{p}^j) \subseteq P_G \subseteq P_0 \setminus \bigcup_i FeasibleCone_H(p^i).$$

We now show that for a parameter-monotonic LHA with rectangular P_0 and a finite number of counterexamples, collective (and therefore also individual) overapproximation with octagonal constraints is complete, i.e., the overapproximation is empty iff $P_G = \emptyset$. This observation follows from the following:

Proposition 4. *If H is parameter-monotonic, $|CE(H)|$ is finite and P_0 is rectangular, $p^*_{MR}(P_0) \in Oct_{\mathtt{P}}(\bigcup_{\pi \in CE(H,P_0)} [\![PathCon(\pi, p) \wedge p \in P_0]\!])$ iff $p^*_{MR}(P_0) \in P_B$.*

Proof. (Sketch) If $p^*_{MR}(P_0) \in P_B$, it follows from Lemma 1, the definition of p^*_{MR} and Prop. 2 that $P_B = P_0$. Since P_0 is rectangular, the octagonal overapproximation of P_B is identical to P_0 and therefore to P_B, and contains $p^*_{MR}(P_0)$. If $p^*_{MR}(P_0) \notin P_B$ and there is a finite number of refining counterexamples, there exists at least one pair of parameters $\mathtt{p_i}, \mathtt{p_j}$ such that for all $\pi \in CE(H, P_0)$

$$max_{p' \in [\![PathCon(\pi,p)]\!]_{\downarrow \mathtt{P}} \cap P_0} psgn(\mathtt{p_i}) p'_i + psgn(\mathtt{p_j}) p'_j < p^*_{MR,i}(P_0) + p^*_{MR,j}(P_0),$$

since otherwise $p^*_{MR}(P_0)$ would be feasible for some path π. From (8) it follows that $p^*_{MR}(P_0) \notin Oct_{\mathtt{P}}(\bigcup_{\pi \in CE(H,P_0)} [\![PathCon(\pi) \wedge \pi \in CE(H, P_0))]\!]$. □

5 Counterexample-Guided Parameter Synthesis

We adapt the familiar *counterexample guided abstraction refinement* (CEGAR) loop [2] to parameter synthesis based on the results in the previous section. We define a CEGAR loop with the following operators:

- $\Pi' := \mathit{IniAbstr}(H, P)$ constructs a set of paths Π' with $\mathit{FCE}(H, P) \subseteq \Pi'$,
- $\pi := \mathit{SelectCE}(\Pi)$ selects a path in Π, given $\Pi \neq \emptyset$,
- $\Pi' := \mathit{RefineWithSpuriousCE}(\Pi, H, P, \pi)$ refines the set of paths Π using the system H and the spurious counterexample π, i.e., it produces a set of paths Π' such that $\Pi' \subseteq \Pi \setminus \{\pi\}$, and $\mathit{FCE}(H, P) \subseteq \Pi'$ (ensuring that the refinement never removes feasible counterexamples).

A number of CEGAR algorithms, including iterative relaxation abstraction [3], can be brought to this form. Note in general CEGAR constructs and refines a finite abstraction, which may take on various forms. We represent this abstraction as a set of paths Π to simplify and generalize the theoretical discussion, but assume that implementations model this set implicitly, say with an LHA or finite state machine. Algorithm 1 shows our adapted CEGAR loop. Its inputs are the system H and the initial parameter domain $\hat{P}_0$, from which the resulting good parameters are chosen. If it terminates, it outputs an underapproximation of the good parameters $\hat{P}_G$. A conventional CEGAR loop would terminate as soon as the feasibility test in line 6 evaluates to true, reporting π_i as a feasible counterexample. In our adaptation, we instead restrict the parameters such that the counterexample is infeasible and continue to search for other feasible counterexamples. This is accomplished by two additional operators:

- $P' := \mathit{MakeInfeasible}(C, P)$ returns a set $P' \subseteq P$ such that $[\![C]\!]{\downarrow_{\mathtt{P}}} \cap P' = \emptyset$. Note that the only such set might be $P' = \emptyset$,
- $P' := \mathit{AnalysisParameters}(P)$ returns a set of parameter values P' such that $\mathit{FCE}(H, P) \subseteq \mathit{FCE}(H, P')$.

The exact implementation of $\mathit{MakeInfeasible}(C, P)$ is $P \setminus [\![C]\!]{\downarrow_{\mathtt{P}}}$, but as discussed in the previous sections, underapproximations similar to (9) and (10) may be advisable. The operator *AnalysisParameters* is used to simplify the set of parameters used in the analysis of H. The condition is that this simplification does not drop any feasible counterexamples. As shown in Sect. 4, one may select a single point in P if the parameters are monotonic, i.e., use $\mathit{AnalysisParameters}(P) := p^*_{LR}(P)$. In that case, the parameters in H can be substituted by constants, thus reducing the number of variables in H. For the general case, a valid implementation is simply $\mathit{AnalysisParameters}(P) := P$.

On the basis of (6) it is straightforward to show that when Alg. 1 terminates, $H(p)$ is safe for any $p \in \hat{P}_G$:

Proposition 5. *If $\hat{P}_0 \supseteq P_0$ and Alg. 1 terminates, $\hat{P}_G \subseteq P_G$.*

If the initial abstraction contains all counterexamples and the implementation of $\mathit{MakeInfeasible}(C, P)$ is exact, $\hat{P}_G$ is exact as well:

Algorithm 1: Counterexample-Guided Parameter Synthesis

Input: LHA H with bad locations L_B, parameter domain $\hat{P}_0$
Output: $\hat{P}_G$ such that $\hat{P}_G \subseteq P_G$

1 $i := 0$;
2 $\tilde{P}_0 := \mathit{AnalysisParameters}(\hat{P}_0)$;
3 $\hat{\Pi}_0 := \mathit{IniAbstr}(H, \tilde{P}_0)$;
4 **while** $\hat{\Pi}_i \neq \emptyset \wedge \hat{P}_i \neq \emptyset$ **do**
5 $\pi_i := \mathit{SelectCE}(\hat{\Pi}_i)$;
6 **if** $[\![\mathit{PathCon}(\pi_i, p) \wedge p \in \hat{P}_i]\!] \neq \emptyset$ **then**
7 $\hat{P}_{i+1} := \mathit{MakeInfeasible}(\mathit{PathCon}(\pi_i, p), \hat{P}_i)$;
8 $\tilde{P}_{i+1} := \mathit{AnalysisParameters}(\hat{P}_{i+1})$;
9 **else**
10 $\hat{P}_{i+1} := \hat{P}_i$; $\tilde{P}_{i+1} := \tilde{P}_i$;
11 **end**
12 $\hat{\Pi}_{i+1} := \mathit{RefineWithSpuriousCE}(\hat{\Pi}_i, H, \tilde{P}_{i+1}, \pi_i)$;
13 $i := i + 1$;
14 **end**
15 $\hat{P}_G := \hat{P}_i$;

Proposition 6. *If $\hat{P}_0 \supseteq P_0$, Alg. 1 terminates, $\mathit{IniAbstr}(H, P_0) = \mathit{CE}(H, P_0)$, and $\mathit{MakeInfeasible}(C, P) = P \setminus [\![C]\!]\downarrow_{\mathbb{P}}$, $\hat{P}_G = P_G$.*

We now briefly present a way to check less paths for parameter-monotonic LHA. According to Prop. 1, we may substitute the parameters with the value $p^*_{LR}(P_0)$ to produce an overapproximation of $\mathit{CE}(H, P_0)$. Usually, there is a prohibitively large number of such paths. Instead, we wish to start with a small set of paths, and iteratively add further paths only when necessary. We run Alg. 1 with $\hat{P}_0 = P_0$ and $\mathit{AnalysisParameters}(P) := p^*_{MR}(P)$, to obtain with $\hat{P}_G$ an initial set of parameters checking the least number of paths possible. To check the remaining paths, we run Alg. 1 with $\hat{P}_0 = \hat{P}_G$ and $\mathit{AnalysisParameters}(P) := p^*_{LR}(P)$. Note that we can skip the initialization of $\hat{\Pi}_0$ in line 3 when in running Alg. 1 in step 2 above, and instead continue with $\hat{\Pi}_i$ from step 1.

We use the following two CEGAR implementations, and show experimental results in the next section. *Simple Discrete CEGAR* is a straightforward CEGAR algorithm based on a discrete abstraction of the hybrid system, somewhat similar to [6]. A set of paths is represented by a finite state machine (FSM). $\mathit{IniAbstr}(H, P)$ constructs a FSM $A_0 = (\mathit{Locs}, \Sigma, \rightarrow, \mathit{ini}, \mathit{final})$, where $\Sigma = \mathit{Locs} \times \mathit{Lab}$, and $\rightarrow = \{(l, (l, \alpha), l') \mid \exists \mu : (l, \alpha, \mu, l') \in \mathit{Trans}\}$. $\mathit{SelectCE}(A_i)$ returns a shortest word in the language of A_i. *RefineWithSpurious* works as follows. First, it reduces the counterexample $\pi = \alpha_0, \ldots, \alpha_{z-1}$ by finding the largest k and the smallest j such that the path constraints for $\alpha_k, \ldots, \alpha_j$ are infeasible (starting in the location l_k reached by $\alpha_0, \ldots, \alpha_{k-1}$). For FSM A and language L let $A - L$ denote removing the language L from the language of A. This is a standard automaton operation, implemented by first coding L by an automaton.

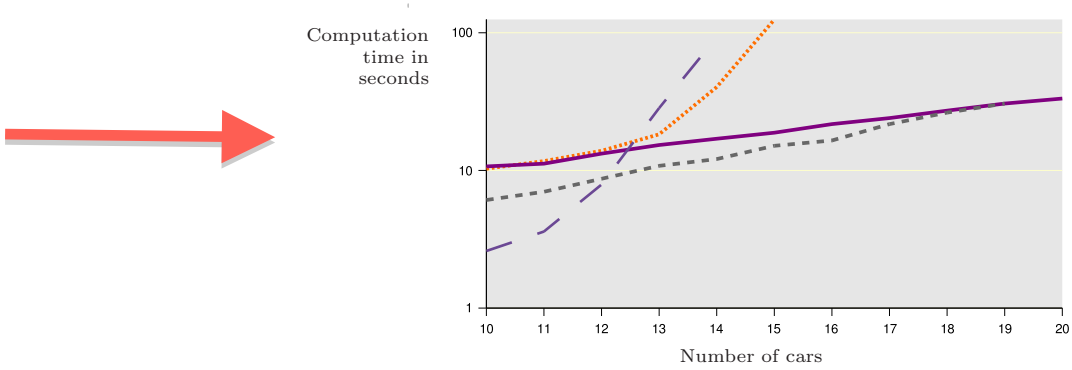


Fig. 3. Parameter synthesis for automated highway controller using standard reachability (wide dashed), simple discrete CEGAR with projection as in (6) (dotted) and with individual octagonal overapproximations as in (9) (solid), and IRA with individual octagonal overapproximations as in (9) (fine dashed)

Let $L_{pre} = \varepsilon$ if $k = 0$ and $L_{pre} = \Sigma^*$ otherwise, and $L_{post} = \varepsilon$ if $j = z - 1$ and $L_{post} = \Sigma^*$ otherwise, Then $A_{i+1} = A_i - L_{pre}(l_k, \alpha_k) \dots (l_j, \alpha_j) L_{post}$.

Iterative Relaxation Abstraction[3] is similar to Simple Discrete CEGAR, with $A_{i+1} = \mathit{RefineWithSpurious}(A_i, H, P, \pi)$ constructed as follows: First, find an irreducible infeasible subset (IIS) of $\mathit{PathCon}(\pi, p)$, say C. Let V be the variables (including parameters) with nonzero coefficients in C. Let $\mathit{localize}(H, V)$ be the automaton obtained from H by removing all constraints involving variables not in V. Let A' be the language of $\mathit{localize}(H, V)$, which is semi-computable using reachability techniques. Then $A_{i+1} = A_i \cap A' \setminus \pi$.

6 Experimental Results

We consider the model of a central arbiter for an automated highway, roughly similar to the one in [7], and verify that no two vehicles on the automated highway collide with each other. The arbiter provides an allowed range $[a, b]$ for the velocity for each vehicle. When two vehicles come within a distance d_{std} of each other, the arbiter asks the faster car and all behind it to reduce the speed to a' and the slower car and all in front of it to increase the speed to b'. When the distance between the two vehicles involved exceeds d_{normal}, the arbiter goes back to normal. The cars are considered to have crashed if their distance is below c. The LHA model for n cars has n continuous state variables and the parameters d_{std} and d_{normal}. The number of counterexamples is infinite, since the controller can cycle infinitely between normal and recovery mode before a crash occurs.

We consider the constants $a = 40$, $b = 60$, $a' = 41$, $b' = 80$, $c = 0.002$. Using CEGAR and octagonal overapproximations, we synthesize the solution $d_{std} \leq d_{normal} \leq 10 \wedge d_{std} > 0.002$. The plot of the log of the time taken vs. the number of cars in Fig. 3 shows that the cost of standard reachability is double-exponential, while using CEGAR (simple discrete or IRA) it is exponential with

a low factor (actually due to the time it takes to compose the system, not the analysis itself). For illustration, we also include the time it takes to obtain the exact solution as in (6) using CEGAR and Fourier-Motzkin elimination – it is also double exponential. These results were obtained on a 1.8 GHz AMD Opteron processor with 16 GB RAM running 32 bit code under Linux. For linear programming we use GLPK [8] with exact arithmetic for simple discrete CEGAR, and CPLEX [9] with floating point arithmetic for IRA.

7 Conclusions

This paper proposes a method for using counterexamples to guide the construction of a set of good parameters for parameterized LHA. The proposed procedure extends the philosophy of CEGAR for verification to a class of design problems. The method is complete when the parameters are monotonic. The effectiveness of the approach is illustrated for an example of an automatic highway controller.

The implications of parameter-monotonicity merit further investigation. If an LHA is not monotonic in the parameters, it can be brought to monotonic form by replacing each parameter $\mathtt{p_i}$ that occurs with both signs with $\mathtt{p_i^+}$ where it occurs with positive sign and $\mathtt{p_i^-}$ where it occurs with negative sign. If P'_G is the set of good parameters for the modified LHA, the set of good parameters for the original system is given by $P_G = P'_G \cap [\![p_i^+ = p_i^-]\!]$. Further research is needed to determine whether or not this leads to any computational advantage for LHA that are not parameter-monotonic.

References

1. Thomas A. Henzinger and Howard Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In *Formal Methods for Industrial Applications*, number 1165 in LNCS, pages 265–282. Springer Verlag, 1996.
2. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV'00*, pages 154–169, London, UK, 2000. Springer.
3. S.K. Jha, B.H. Krogh, J.E. Weimer, and E.M. Clarke. Reachability for linear hybrid automata using iterative relaxation abstraction. In A. Bemporad, A. Bicchi, and G.C. Buttazzo, editors, *HSCC*, volume 4416 of *LNCS*. Springer, 2007.
4. Thomas A. Henzinger. The theory of hybrid automata. In *Proc. 11th Annual IEEE Symposium on Logic in Computer Science, LICS'96, New Brunswick, New Jersey, 27-30 July 1996*, pages 278–292. IEEE Computer Society Press, 1996.
5. X. Li, S. K. Jha, and L. Bu. Towards an Efficient Path-Oriented Tool for Bounded Reachability analysis of Linear Hybrid Systems using Linear Programming. In *BMC'06: Proceedings of the Workshop on Bounded Model Checking*, 2006.
6. Marc Segelken. Abstraction and counterexample-guided construction of ω-automata for model checking of step-discrete linear hybrid models. In W. Damm and H. Hermanns, editors, *CAV*, volume 4590 of *LNCS*, pages 433–448. Springer, 2007.
7. R. Horowitz and P. Varaiya. Control design of an automated highway system. *Proc. IEEE*, 88:913–925, July 2000.
8. GNU Linear Programming Kit, v.4.17, 2007. `http://www.gnu.org/software/glpk`.
9. ILOG. `http://www.ilog.com/products/cplex/product/simplex.cfm`, 2007.