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: $CE = \{ce_1, \dots ce_N\}$
1. Initialize $CE$ to be the empty set.
2. Pick a set of m ($> N$) distinct counterexamples $C = \{ce_1, ce_2 \dots ce_m\}$ from $A^i_{CE}$.
3. Build a set of linear programs $\{lp_1, lp_2 \dots lp_m\}$ corresponding to each of $\{ce_1, ce_2 \dots ce_m\}$
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.