# Automated Revision of Legacy Real-Time Programs: Work in Progress[1]

Borzoo Bonakdarpour Sandeep S. Kulkarni
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI 48824, USA
Email: {borzoo,sandeep}@cse.msu.edu

## Abstract

*In this paper, we focus on the problem of automatic revision of legacy real-time programs. We consider this problem in two contexts. First, we investigate the problem of automated addition of properties expressed in Metric Temporal Logic (*MTL*) formulas to existing real-time programs modeled in Alur and Dill timed automata. Then, we consider transformation problems, where we design synthesis methods to add fault-tolerance to existing fault-intolerant real-time programs. While both problems have been addressed in the literature for untimed programs in theory and practice, there is much to be done for real-time programs. To this end, we concentrate on filling the gap between theory and practice of automated methods for synthesizing real-time programs by characterizing the class of real-time programs and properties, where program synthesis is practically feasible.*

**Keywords: Timed automata, Real-time, Bounded liveness, Program synthesis, Program transformation, Fault-tolerance, Formal methods.**

## 1. Introduction

*Automated program synthesis* is the problem of designing an algorithmic method to find a program that satisfies a required set of behaviors. Depending upon the choice of formulation of the problem and expressiveness of specifications and programs, the class of complexity of synthesis methods varies from polynomial-time to undecidability.

In this paper, we describe our ongoing research on automated synthesis methods for revising legacy real-time programs. In this research, we characterize the class of real-time programs and properties, where automated synthesis can be achieved efficiently and the ones that involve exponential complexities. Moreover, we study the problem of revising existing real-time programs in two contexts: (1) adding properties typically used in specifying real-time requirements to existing real-time programs, and (2) adding fault-tolerance to existing fault-intolerant real-time programs. Since we use the timed automata formalism [1], many real-time scheduling problems can be reduced to our problems as well.

In the context of untimed programs, similar problems have been addressed in the literature. In [2], the authors consider the problem of automated revision of legacy UNITY [3] programs. The problem of synthesizing *untimed* fault-tolerant programs has been studied from different perspectives. In [4–6], the authors propose synthesis algorithms for adding fault-tolerance and multitolerance to existing programs in the high (respectively, low) atomicity model, where processes can (respectively, cannot) read and write all the program variables in one atomic step.

Synthesis of real-time programs has mostly been formulated in the context of timed control synthesis from game theoretical perspective [7–12]. There are two drawbacks in the proposed methods that make the synthesis methods either unrealistic or unfeasible: (1) the complexity of such methods are too high to be used in practice (at least EXPTIME-complete), and/or (2) either the given program or the specification must be deterministic.

**Organization of the paper.** In Section 2, we present the preliminary concepts. In Section 3, we present our current results and ongoing research on adding properties to legacy real-time programs. Then, in Section 4, we present our current results and ongoing research on designing synthesis methods to add fault-tolerance to legacy real-time programs. In Section 5, we describe our plan to use our theoretical results in practice. Finally, we make the concluding remarks in Section 6.

## 2. Background

In this section, we give intuitive definitions of the terminology that we use in this paper. A legacy *real-time pro-*

1 This work was partially sponsored by NSF CAREER CCR-0092724, DARPA Grant OSURS01-C-1901, ONR Grant N00014-01-1-0744, NSF grant EIA-0130724, and a grant from Michigan State University.

*gram* is specified in terms of its state space and set of transitions. We use the notion of timed automata [1] to represent real-time programs. A timed automaton is a traditional finite state automaton equipped with clock variables and timing constraints.

We model real-time properties by *Metric Temporal Logic* (MTL) formulas. Linear Temporal Logic (LTL) specifies the *qualitative part* of a program. MTL introduces real time to LTL by constraining temporal operators, so that one can specify the *quantitative part* as well. For instance, the constrained eventually operator $\Diamond_{[1,3]}$ is interpreted as "eventually within 1 to 3 time units both inclusive". A *specification* is a conjunction of a set of MTL properties. A *computation* is an infinite sequence of states. Note that, an MTL property defines a set of computations. We say that a real time program $\mathcal{P}$ *satisfies* specification (or property) $\Sigma$ iff all computations of $\mathcal{P}$ are in $\Sigma$. Otherwise, we say that $\mathcal{P}$ *violates* $\Sigma$.

## 3. Adding Properties to Legacy Real-Time Programs

In this section, we describe the problem of revising legacy real-time programs through adding new properties. Furthermore, we present our current results and ongoing research in this regard.

**Problem statement.** Given are a real-time program $\mathcal{P}$ and an MTL property $\Pi$. Our goal is to transform $\mathcal{P}$ into a new real-time program $\mathcal{P}'$ such that $\mathcal{P}'$ satisfies the given property $\Pi$ and it continues to satisfy its old specification, which is not necessarily given. $\Box$

Since the notion of the given property in the problem statement can be any formula expressed in MTL, we have to deal with a highly expressive set of properties, which in turn (if not undecidable) requires highly complex algorithms. Hence, we focus on properties typically used in specifying real-time systems rather than solving the problem for any arbitrary property. More specifically, as the starting point, we focus on the well-known time-bounded response property. A *time-bounded response property* is of the form $\Pi \equiv \Box(p \rightarrow \Diamond_{\leq\delta} q)$, where $p$ and $q$ are two sets of states, and $\delta$ is a positive real number, i.e., it is always the case that a $p$-state is followed by a $q$-state within $\delta$ time units. One can instantiate the problem statement in the obvious way, i.e., our goal is to add a time-bounded response property $\Pi$ to an existing real-time program $\mathcal{P}$.

### 3.1. Current Results

Our current results fall in two categories: (1) transformation algorithms, and (2) hardness results. In the first category, we identify the classes of MTL properties where program transformation can be achieved efficiently.

**Transformation algorithms.** As the first step, we have developed a sound and complete algorithm that adds a time-bounded response property to a real-time program $\mathcal{P}$. Our algorithm first transforms a real-time program into a detailed *region graph* $R(\mathcal{P})$ [1]. Region graphs abstract the notion of time from timed automata while maintaining the set of computations. Vertices of $R(\mathcal{P})$ are called *regions* and transitions are called *edges*. After generating $R(\mathcal{P})$, we prune the regions of the region graph from where the given time-bounded response property $\Pi$ is violated. Towards this end, we first transform $R(\mathcal{P})$ into an ordinary weighted directed graph $G$ proposed in [13]. In this graph, the longest distance between two vertices is equal to maximum time delay between the corresponding regions. We prune the regions from where the given time-bounded response property is violated using standard shortest and longest path algorithms. More specifically, we find a subgraph of $G$, say $G'$, such that we are assured that the length of the longest path from each $p$-state to a $q$-state in $G'$ is at most $\delta$. To this end, we include the shortest path from each $p$-state to a $q$-state, provided the length of such a path is at most $\delta$. Then, we transform the resulting digraph $G'$ back to a region graph $R(\mathcal{P}')$ and finally to a real-time program $\mathcal{P}'$. Note that, region graphs are time-abstract bisimulations [1].

**Theorem 3.1.** The above algorithm for adding a time-bounded response property to a real-time program is sound and complete [14]. $\Box$

**Theorem 3.2.** The problem of adding a bounded response property to a real-time program is PSPACE-complete [14]. $\Box$

The novelty of the above algorithm is it is sound and complete. In other words, not only the algorithm synthesizes a real-time program that is correct by construction, i.e., it satisfies the constraints of the program statement, but also we are guaranteed that if a solution to the problem statement for a given instance exists, our algorithm finds a solution as well. The completeness of our algorithm is important in sense that if the algorithm fails to find a solution, we are assured that the given real-time program is not *fixable* [2].

Note, however, that although our algorithm is complete, since it may prune some states and transitions unnecessarily, the synthesized program does not have maximum nondeterministic. Maximum nondeterminism is desirable in the sense that it increases the chance of success for further revision of programs, e.g., adding other time-bounded response properties.

**Hardness results.** We now describe our hardness results based on different types of bounded response properties:

- *Adding time-bounded response properties with maximum non-determinism*: We model non-determinism in terms the number of outgoing transitions from states of a program. We have shown that the problem of addition of a bounded response property while maintaining maximum set of program transitions is NP-complete in the size the program's region graph [14].

- *Adding interval bounded response properties*: An interval bounded response property is of the form $\Pi \equiv \Box(p \rightarrow \Diamond_{\leq[\delta_1,\delta_2]} q)$, i.e., a $p$-state should follow by a $q$-state not later than $\delta_2$ time units and at least after $\delta_1$ time units. We have shown that the problem of addition of such a property to an existing real-time program is also NP-complete in the size of program's region graph [14].

- *Adding unbounded response properties*: An unbounded response property (also called *leads-to*) is of the form $\Pi \equiv \Box(p \rightarrow \Diamond q)$. We have shown that the problem of addition of an unbounded response property to a given real-time program is PSPACE-complete [15].

### 3.2. Ongoing Research

We are currently working on identifying the complexity of adding other types of MTL properties to existing real-time programs. In fact, we plan to identify a boundary for the class of properties that can be added to legacy real-time programs with reasonable time and space complexity. In this context, an open problem is identifying the class of complexity of simultaneous addition a set of MTL properties, e.g., two time-bounded eventually properties $(\Diamond_{\leq\delta_1} p \wedge \Diamond_{\leq\delta_2} q)$.

We are also doing research to overcome the state explosion problem. More specifically, we are developing transforation algorithms such that they manipulate a *zone graph* [16] rather than a region graph. Zone graph is a more concise time-abstract representation of real-time programs used in model checking techniques. Zone graphs collapse a set of regions to a single zone such that it contains enough information for finding counter-examples in model checking. However, since it may abstract a part of the necessary information, devising synthesis algorithms on zone graphs becomes a challenging problem.

## 4. Adding Fault-Tolerance to Legacy Real-Time Programs

Another aspect of our research on automated revision of legacy real-time programs is adding fault-tolerance to existing fault-intolerant real-time programs with respect to a given set of faults. In this section, we present our current results and ongoing research in this regard.

In order to analyze fault-tolerance in the context of real-time programs, we define several *levels of fault-tolerance* based on satisfaction of properties and timing constraints in the presence of faults. More specifically, we introduce three levels of fault-tolerance, namely, *failsafe*, *nonmasking*, and *masking* based on satisfaction of safety and liveness specifications in the presence of faults, respectively. For failsafe and masking fault-tolerance we, furthermore, propose two levels, namely, *soft* and *hard* fault-tolerance based on satisfaction of timing constraints in the presence of faults. For instance, we say that a program is *hard-masking* fault-tolerant iff it satisfies its safety and liveness specifications as well as timing constraints in both the absence and presence of faults.

**Problem statement.** Given are a real-time program $\mathcal{P}$, its invariant, the safety specification $\Sigma$, and a set of fault transitions. Our goal is to transform $\mathcal{P}$ into a new real-time program $\mathcal{P}'$ such that $\mathcal{P}'$ continues to satisfy its old specification in the absence of faults and provides the required level of fault-tolerance in the presence of faults. $\Box$

### 4.1. Current Results

Thus far, we have shown that automated addition of (1) nonmasking, (2) soft and hard-failsafe, and (3) soft-masking fault-tolerance, where timing constraints of the given program is limited to one bounded-response property, can be done in polynomial-time in the size of the input program's region graph. More specifically, we have developed sound and complete transformation algorithms for adding the aforementioned levels of fault-tolerance to real-time programs [15]. The algorithms essentially use the transformation algorithm presented in Subsection 3.1 with some modifications to take the fault-tolerance issues into account.

We have also shown a hardness result for the case where the synthesized hard-masking program is required to satisfy more than one bounded-response property in the presence of faults. More specifically, we have proved that the problem of addition of hard-masking fault-tolerance to an existing real-time program where the resulting program is required to satisfy more than one bounded response property, is NP-complete in the size of the program's region graph [15].

### 4.2. Ongoing Research

Our main open question in the context of addition of fault-tolerance to real-time programs is the complexity of automated addition of hard-masking fault-tolerance where the synthesized program is required to satisfy at most one bounded response property in the presence of faults. Intuitively speaking, a hard-masking fault-tolerant program satisfies a bounded response property in the presence of faults and provides bounded-time recovery after faults stop occurring. Note that, bounded-time recovery is in turn a bounded response property (from the states reached by faults to the states in the program invariant). In [2], we show that adding two unbounded response properties to an untimed program in the absence of faults is NP-complete. However, since bounded-time recovery is in fact a bounded eventually property in nature, we cannot directly apply the results presented in [2] to our problem.

## 5. From Theory to Practice

In this section, we describe the practical aspects of our research. In the theoretical part of our research, we characterize the class of properties and real-time programs where automated synthesis is feasible (in terms of time and space complexity). In the practical part, we plan to use the developed theories (e.g., transformation algorithms) in developing tools for synthesizing real-time programs. Since the complexity of the aforementioned algorithms is comparable to that of existing model checking techniques in dense real-time, we expect that it is feasible to use the proposed algorithms in practice.

Another application of the transformation algorithm presented in Subsection 3.1 is in model checking. Let us consider the case where a model checker generates a counterexample to show that a program does not satisfy a bounded response property. Now, in order to fix the problem, we run our algorithm such that it adds the desired bounded response property to the program. As mentioned in Subsection 3.1, since our algorithm is complete, if it fails to synthesize a program, it means that the program is not fixable with respect to the desired bounded response property and, hence, a more comprehensive approach must be applied (e.g., synthesis from specification).

In the context of adding fault-tolerance to existing fault-intolerant real-time programs, we plan to extend our tool FTSyn[2] (which is currently capable of synthesizing fault-tolerant untimed programs), so that it synthesizes fault-tolerant real-time programs as well. FTSyn has been successfully used to synthesize the classic examples of fault-tolerant computing such as Byzantine agreement, triple modular redundancy, Dijkstra's self-stabilizing token ring algorithm, and etc.

## 6. Conclusion

In this paper, we focused on our research on synthesis methods for revising legacy real-time programs in both theory and practice. We proposed two ways to revise legacy real-time programs: (1) revision through adding new properties, and (2) revision through adding fault-tolerance. Our aim in this research is to characterize the class of properties and programs where automated synthesis can be achieved efficiently. We presented our current results and ongoing research for both adding properties and fault-tolerance to real-time programs.

## References

[1] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] A. Ebnenasir, S. S. Kulkarni, and B. Bonakdarpour. Revising UNITY programs: Possibilities and limitations. In *International Conference on Principles of Distributed Systems (OPODIS)*, pages 275–290, 2005.

[3] K. M. Chandy and J. Misra. *Parallel program design: a foundation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[4] S. S. Kulkarni and A. Arora. Automating the addition of fault-tolerance. In *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, volume 1926 of *Lecture Notes in Computer Science*, pages 82–93, Pune, India, 2000. Springer.

[5] S. S. Kulkarni, A. Arora, and A. Chippada. Polynomial time synthesis of Byzantine agreement. In *Symposium on Reliable Distributed Systems (SRDS)*, pages 130–140, 2001.

[6] S. S. Kulkarni and A. Ebnenasir. Automated synthesis of multitolerance. In *International Conference on Dependable Systems and Networks (DSN)*, pages 209–219, 2004.

[7] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *12th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 229–242, 1995.

[8] E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In *Hybrid Systems: Computation and Control (HSCC)*, pages 19–30, 1999.

[9] D. D'Souza and P. Madhusudan. Timed control synthesis for external specifications. In *19th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 571–582, 2002.

[10] P. Bouyer, D. D'Souza, P. Madhusudan, and A. Petit. Timed control with partial observability. In *Computer Aided Verification (CAV)*, pages 180–192, 2003.

[11] M. Faella, S. LaTorre, and A. Murano. Dense real-time games. In *Logic in Computer Science (LICS)*, pages 167–176, 2002.

[12] L. de Alfaro, M. Faella, T. A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *International Conference on Concurrency Theory (CONCUR)*, 2003.

[13] C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. In *Computer-Aided Verificaion (CAV)*, pages 399–409, 1991.

[14] B. Bonakdarpour and S. S. Kulkarni. Automated incremental synthesis of timed automata. In *International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, pages 261–276, 2006.

[15] B. Bonakdarpour and S. S. Kulkarni. Incremental synthesis of fault-tolerant real-time programs. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*, pages 122–136, 2006.

[16] R. Alur, C. Courcoubetis, N. Halbwachs, D. L. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *International Conference on Concurrency Theory (CONCUR)*, pages 340–354, 1992.

2 For more information, visit `http://www.cse.msu.edu/~ebnenasi/research/tools/ftsyn.htm`.