# Automatic Generation of Simulation Monitors from Quantitative Constraint Formula

*Xi Chen, Harry Hsieh*
*University of California, Riverside, CA*
*E-mail: {xichen, harry}@cs.ucr.edu*

*Felice Balarin, Yosinori Watanabe*
*Cadence Berkeley Laboratories, Berkeley, CA*
*E-mail: {felice, watanabe}@cadence.com*

## Abstract

*System design methodology is poised to become the next big enabler for highly sophisticated electronic products. Design verification continues to be a major challenge and simulation will remain an important tool for making sure that implementations perform as they should. In this paper we present algorithms to automatically generate C++ checkers from any formula written in the formal quantitative constraint language, Logic Of Constraints (LOC). The executable can then be used to analyze the simulation traces for constraint violation and output debugging information. Different checkers can be generated for fast analysis under different memory limitations. LOC is particularly suitable for specification of system level quantitative constraints where relative coordination of instances of events, not lower level interaction, is of paramount concern. We illustrate the usefulness and efficiency of our automatic trace analysis methodology with case studies on large simulation traces from various system level designs.*

## 1 Introduction

The increasing complexity of embedded systems today demands more sophisticated design and test methodologies. Systems are becoming more integrated as more and more functionality and features are required for the product to succeed in the marketplace. Designing at the Register Transfer Level (RTL) or sequential C-code level, as is done by embedded hardware and software developers today, is no longer efficient. The next major productivity gain will come in the form of system level design.

In this paper, we propose an efficient automatic approach to analyze simulation traces and check whether they satisfy quantitative properties specified by denotational logic formulas. The property to be verified is written in Logic of Constraints (LOC) [4], a logic particularly suitable for specifying constraints at the abstract system level, where coordination of executions, not the low level interaction, is of paramount concern. We then automatically generate a C++ trace checker from the quantitative LOC formula. The checker analyzes the traces and reports any violations of the LOC formula. Like any other simulation-based approach, the checker can only disprove the LOC formula (if a violation is found), but it can never prove it conclusively, as that would require analyzing infinitely many traces.

In the next section, we review the definition of LOC and compare it with other forms of logic and constraint specification. In section 3, we discuss the algorithm for building a trace checker for any given LOC formula. We demonstrate the usefulness and efficiency with a verification case study in section 4. Finally, in section 5, we conclude this paper.

## 2 Logic Of Constraints (LOC)

Logic Of Constraints [4] is a formalism designed to reason about simulation traces. It consists of all the terms and operators allowed in sentential logic, with additions that make it possible to specify system level quantitative constraints without compromising the ease of analysis. The basic components of an LOC formula are **events**, **event instances**, **event indexes** and **annotations**.

LOC can be used to specify some common real-time constraints:

- **rate:** E.g. "a new $Display$ will be produced every 10 time units": $t(Display[i+1]) - t(Display[i]) = 10$
- **latency:** E.g. "$Display$ is generated no more than 25 time units after $Stimuli$": $t(Display[i]) - t(Stimuli[i]) \leq 25$
- **jitter:** E.g. "every $Display$ is no more than 15 time units away from the corresponding tick of the real-time clock with period 15": $\mid t(Display[i]) - i * 10 \mid \leq 15$

By adding additional index variables and quantifiers, LOC can be extended to be at least as expressive as S1S [3] and Linear Temporal Logic. There are no inherent problem in generating simulation monitor for them.

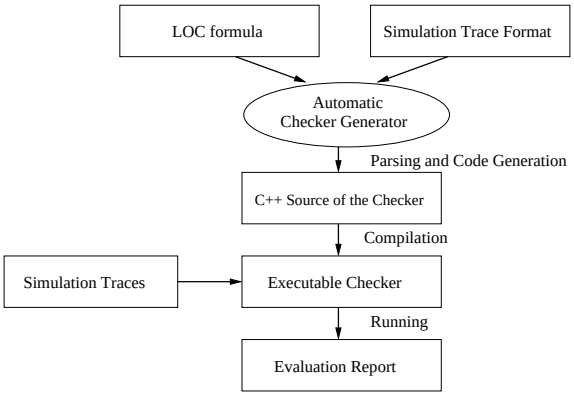


**Figure 1. Trace Analysis Methodology.**

## 3 The LOC Checker

We analyze simulation traces for LOC constraint violation. The methodology for verification with automatically generated LOC checker is illustrated in Figure 1. From the LOC formula and the trace format specification, an automatic tool is used to generate a C++ LOC checker. The checker is compiled into an executable that will take in simulation traces and report any constraint violation. To help the designer to find the point of error easily, the error report will include the value of index $i$ which violates the constraint and the value of each annotation in the formula. The checker is designed to keep checking and reporting any violation until stopped by the user or if the trace terminates.

The algorithm progresses based on index variable $i$. Each LOC formula instance is checked sequentially with the value of $i$ being 0, 1, 2, ...etc. A formula instance is a formula with *i* evaluated to some fixed positive integer number. In addition, a memory recycling algorithm is utilized to minimize the runtime memory usage.

## 4 Case Study

In this section, we apply the methodology discussed in the previous section to an design example. It is a Synchronous Data Flow (SDF) [5] design called Expression originally specified in Ptolemy and is part of the standard PtolemyII [1] distribution. The Expression design is respecified and simulated with SystemC simulator [2].

Figure 2 shows a SDF design. The data generators SLOW and FAST generate data at different rates, and the EXPR process takes one input from each, performs some operation (in this case, multiplication) and outputs the result to DISPLAY. SDF designs have the property that different scheduling will result in the same behavior. A snapshot of the simulation trace is shown in Figure 2.

The following LOC formula must be satisfied for any

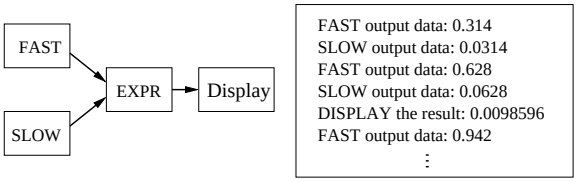


**Figure 2. Expression Design and Simulation Trace.**

correct simulation of the given SDF design:

$$SLOW[i] * FAST[i] = DISPLAY[i] \quad (1)$$

We use the automatically generated checkers to show that the traces from SystemC simulation adhere to the property. The analysis time is linear to the size of the trace file and the maximum memory usage is constant regardless of the trace file size(see table 1. The platform for the experiment is a dual 1.5GHz Athlon system with 1GB of memory.

**Table 1. Result of Constraint (1) on EXPR**

| Lines of Trace | $10^4$ | $10^5$ | $10^6$ | $10^7$ |
|---|---|---|---|---|
| Time Used(s) | <1 | 1 | 12 | 130 |
| Memory Usage | 8KB | 8KB | 8KB | 8KB |

## 5 Conclusion

In this paper, we have presented a methodology for system-level verification through automatic trace analysis. We have demonstrated how we take any formula written in the formal quantitative constraint language, Logic Of Constraint, and automatically generate a trace checker that will efficiently analyze the simulation traces for constraint violations. The analyzer is fast even under memory limitation. We applied the methodology to many case studies and demonstrate that automatic LOC trace analysis can be very useful.

## References

[1] http://www.ptolemy.eecs.berkeley.edu.
[2] http://www.systemc.org.
[3] A. Aziz, F. Balarin, R. Brayton, and A. Sangiovanni-Vincentelli. Sequential synthesis using s1s. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(10).
[4] F. Balarin, Y. Watanabe, J. Burch, L. Lavagno, R. Passerone, and A. Sangiovanni-Vincentelli. Constraints specification at higher levels of abstraction. *International Workshop on High Level Design Validation and Test - HLDVT01*, Sept. 2001.
[5] E. Lee and D. Messerschmitt. Synchronous data flow. *Proceedings of IEEE*, Sept. 1987.