Implementing a Tool for Timing Analysis of Real-Time Production Systems

Albert Mo Kim Cheng
Department of Computer Science
University of Houston--University Park
Houston, Texas 77204-3475

Abstract

The Estella-General Analysis Tool E-GAT is a computer-aided software engineering tool for performing response time analysis of real-time production systems written in the EQL rule-based language. E-GAT detects potential timing errors statically, making it a powerful aid for the rapid prototyping and development of expert systems with guaranteed response time. It is based on a powerful analysis methodology [4] which exploits the identification of rule sets satisfying certain general behavioral constraint assertions. This paper describes the implementation of E-GAT with efficient algorithms.

1. Introduction

There has been little work in the formal verification and validation of production systems in the real-time environment (see [3] for a bibliography). Determining how fast an expert system can respond under all possible situations is a difficult and in general an undecidable problem [1]. In [4], we have developed a powerful and efficient analysis methodology for analyzing a large class of rule-based EQL programs to determine whether a program in this class has bounded response time. In particular, we have identified several sets of general behavioral constraint assertions called special forms of rules with the following property: an EQL program which satisfies all constraints in one of these sets of constraint assertions is guaranteed to have bounded response time. An EQL program consists of a finite set of rules each of which is of the form:

\[ a_1 := b_1 \land a_2 := b_2 \land \cdots \land a_n := b_n \text{ IF test} \]

The language Estella is used to specify application-specific behavioral constraint assertions which are compiled by the Estella compiler to produce recognition routines and then applied to the research initiation award from the National Science Foundation under award No. CCR-9111563 and by a research initiation grant under account number 1-1-27491 from the University of Houston--University Park.

2. The Estella-General Analysis Tool

The facility for specifying behavioral constraint assertions on EQL rules and analyzing EQL programs consists of the following major components:

(1) the recognition procedure generator for special forms,
(2) the EQL program information extractor, and
(3) the general analyzer.

The recognition procedure generator serves as a compiler for the specification written in Estella and generates the corresponding procedure for recognizing the specified special forms of rules with bounded response times. The EQL program information extractor is a collection of procedures that extract relevant information from the EQL program to be analyzed. These procedures provide information in the form of objects that can be used in the Estella specification. The general analyzer takes as input the information provided by the information extractor and the recognition procedures generated by the recognition procedure generator, and produces timing analysis results regarding the EQL program.

3. Implementation of the Estella-General Analysis Tool

The general analysis tool allows the rule-based programmer to select a subset of the rules in a program for analysis. The subset may contain either a contiguous list of rules or separated rules. This provision reduces analysis time by directing the analyzer to focus on 'trouble spots' which the rule-based programmer considers as possible sources of timing violations. In the following section, we shall show how the analyzer uses decomposition techniques to break the program into independent sets so that it does not have to analyze the entire program at once. Timing requirements of these algorithms shall be reported in another paper.

Algorithm A. General Analyzer.

Input: An EQL program or a set of EQL rules; a list of special forms and exceptions, if any, specified in Estella.

Output: If the program will always reach a fixed point in bounded time, output 'yes'. If the program may not always reach a fixed point in bounded time according to the analysis, output 'no' and the rules involved in the possible timing violations.

(1) Parse the special form specifications and exceptions; then generate the corresponding special form recognition procedures.
(2) Construct the high-level dependency graph corresponding to the program by using algorithm B.
(3) If there are more rules for analysis, identify forward-independent sets of rules (defined below) which are in special forms using algorithm C. If at least one rule set in special form is found and there are more rules to be analyzed, go to step 5. If there are no more rules for analysis, output 'yes' (the
EQL rules have bounded response time).

(4a) If no independent set of rules is in a special form catalogued but the variables in some rule set have finite domains, check whether this rule set can always reach a fixed point in bounded time by using a state-based model checking algorithm. If the rule set is determined to be able to always reach a fixed point in bounded time, go to step 5. If the rule set is determined to be unable to always reach a fixed point in bounded time, report the rules involved in the timing violations; go to step (4b).

(4b) Prompt the user for new special forms or exceptions. If new special forms or exceptions are entered, go to step 3. If none are entered, stop; output 'no' (the EQL rules do not have bounded response time).

(5) Mark those forward-independent sets identified to be in special forms as checked (thus removing those rules from further analysis). Rewrite remaining rules to take advantage of the fact that some variables can be treated as constants because of the special form; go to step (3).

3.1. Selecting Independent Special Form Set

Let $L_0$ and $R_0$ be the sets of variables appearing in LHS and in RHS of rule set $A$ respectively.

**Theorem 1. (Sufficient conditions for independence)**

Let $S$ and $Q$ be two disjoint sets of rules. $S$ is independent from $Q$ if the following conditions hold:

1. $L_S \cap L_Q = \emptyset$.
2. The rules in $Q$ do not potentially enable rules in $S$.
3. $R_S \cap L_Q = \emptyset$.

Owing to space limitations, the proofs for the theorems stated in this paper are reported in [2].

3.1.1. Constructing and Checking the Dependency Graph

Before checking to determine if a subset of the rules is in a special form, the analysis algorithm first constructs a high-level dependency graph based on the above conditions for establishing independence of one set of rules from another set.

**Algorithm B. High-level dependency graph construction.**

Input: An EQL program.

Output: A high-level dependency graph corresponding to the input EQL program.

1. For each rule $i$ in the EQL program, create a vertex labeled $i$.
2. Let $S$ contain rule $i$ and let $Q$ contain rule $j$. If one of the conditions 1, 2 or 3 is not satisfied, create a directed edge from vertex $i$ to vertex $j$.
3. Find every strongly connected component in the dependency graph $G(V, E)$ constructed by step 1 and step 2.
4. Let $C_1, C_2, \ldots, C_m$ be the strongly connected components of this graph $G(V, E)$. Define $G(V, E)$ as follows:

$$V = \{C_1, C_2, \ldots, C_m\}$$

$$E = \{(C_i, C_j) | i \neq j, (x, y) \in E, x \in C_i \text{ and } y \in C_j\}$$

We call $G$ the high-level dependency graph of the input EQL program. Each of the vertices $C_i$ in this high-level dependency graph is called a forward-independent set.

**Theorem 2.** The high-level dependency graph of any EQL program is a directed acyclic graph.

3.1.2. Identifying Special Form Sets

The brute-force approach to identify sets of rules in a special form would be to generate all combinations of the rules in a program and then check the rules in each combination to see if they are in one of the special forms catalogued. However, this approach does not take into account the syntactic and semantic structure of an EQL program and it has exponential time complexity. We shall present an algorithm for identifying special form sets by checking the high-level dependency graph constructed by algorithm B.

**Algorithm C. Identification of Special Form Set.**

Input: A high-level dependency graph $G$ corresponding to an EQL program.

Output: Sets of rules, if any, identified to be in some predefined special forms or in some user-defined special forms.

1. Sort the vertices in the high-level dependency graph $G$ to obtain a reverse topological ordering of the vertices. Starting with the first vertex with no out-edge, label the vertices $i$, $i = 1, 2, \ldots, m$, where $m$ is the total number of vertices in $G$. This can be achieved by using a recursive depth-first search algorithm which labels the vertex just visited as described above just before exiting from each call.

2. For the set of rules contained in each vertex $i$, such that $i$ does not have any outgoing edge, determine if the rule set is in one of the special forms catalogued. Report rules which are involved in the violation of specific special form conditions.

References


