Optimization of Rule-Based Expert Systems Via State Transition System Construction[*]

Blaž Zupan[†] Albert Mo Kim Cheng
Department of Computer Science
University of Houston
Houston, Texas 77204-3475
(blaz,cheng)@cs.uh.edu

Abstract

Embedded rule-based expert systems must satisfy stringent timing constraints when applied to real-time environments. The paper describes a novel approach to reduce the response time of rule-based expert systems. Our optimization method is based on a construction of the reduced cycle-free finite state transition system corresponding to the input rule-based system. The method makes use of rule-base system decomposition, concurrency and state-equivalency. The new and optimized system is synthesized from the derived transition system. Compared with the original system, the synthesized system has (1) fewer number of rule firings to reach the fixed point, (2) is inherently stable and (3) has no redundant rules. The synthesis method also determines the tight response time bound of the new system. The optimized system is guaranteed to compute correct results independent of the scheduling strategy and execution environment.

1 Introduction

Validation and verification is becoming a very important phase in the life cycle of an expert system [7]. This is particularly true for real-time expert systems. Apart from functional correctness, a real-time expert system must also satisfy stringent timing constraints.

Rule-based real-time expert systems are investigated herein. These embedded AI systems are increasingly used in different industrial applications, such as airplane avionics, smart robots, space vehicles and other safety critical applications. The result of missing a deadline in these systems may be fatal. The verification task is to prove that the system can deliver an adequate performance in bounded time. If this is not the case or if the real-time expert system is too complex to analyze, the system has to be resynthesized. In this paper, we propose a formal framework to such an optimization.

With respect to the analysis, there has been little work in the field of synthesis of rule-based real-time expert systems. The method presented in this paper has been developed for the EQL rule-based language introduced by Browne et al. [2] and Cheng et al. [6]. They define the response time of a rule-based program as a fixed point convergence time. Efficient algorithms for analyzing the time bounds for rule sets with bounded response time have been developed [5] for this class of programs.

Our method is based on the identification of the finite state transition system for the original rule-based program. Due to the nontrivial nature of expert systems, the combinatorial explosion of states may threaten the implementability of such an approach. Bouajjani et al. [1] proposes an algorithm for minimal model generation that can be used for building a state graph of transition systems. He uses a notion of equivalency of states and instead of reducing a transition system from its entirely generated version, introduces a unique approach of reducing the system during its generation. The final states in such a graph are called *stable states* and may represent more than one basic state of an entirely generated graph. A *stubborn attack on state explosion* by Valmari is another method for state reduction. In [9] he argues that concurrency is the major contributor to state explosion. It introduces a large number of execution sequences which lead from a common start state to a common end state by the same transition, but the transitions occur in different order causing the sequences to go

[*]This material is based upon work supported in part by the National Science Foundation under Award No. CCR-9111563 and by the Texas Advanced Research Program under Grant No. 3652270.

[†]On leave from "Jožef Stefan" Institute, Ljubljana, Slovenia.

through different states. He claims that it would be sufficient to simulate the process in one arbitrarily chosen order and thus reducing the number of states.

Cheng [3] uses parallel rule execution to achieve higher execution speed in rule-based systems. He defines *inter-* and *intra-rule-set parallelism*. The rule-based program is decomposed into inter-dependent sets, rule-sets which do not depend on each other can be executed in parallel. Within each inter-dependent rule-set the concurrency is present due to intra-rule-set parallelism.

The above approaches provide the basis for the method introduced here. In accordance with Cheng we use the decomposition techniques so that we do not have to construct the transition system for the entire program at once. As proposed by Bouajjani et al., we construct the transition system with reduced states. Valmari's approach is combined with Cheng's concept of intra-rule-set parallelism and by using these two ideas, the number of states is further reduced. The method uses also the concept of mutual exclusion of states and builds the transition system so that it contains no cycles.

The new rule-based expert system is derived from the constructed transition system. Unlike the methods that use only parallel execution of rules as a means of increasing the performance [3], we also guarantee stability of the synthesized system. Another advantage of the method is that by using the transition system, it can determine the exact upper bound of rule firings. Thus, the analysis of the constructed rule based system by tools described in [5] is no longer needed.

The rest of the paper is organized as follows. The next section briefly describes the syntax and semantics of the EQL rule-based language, the concept of finite state transition system, and EQL program decomposition techniques. The method for EQL program optimization is then introduced in Section 3 and an attempt to evaluate its performance is given by Section 4. Section 5 is the conclusion.

2 Introductory Definitions

The class of real-time programs considered in this paper are called *equational rule-based* EQL *programs* [2]. Here we define the syntax of EQL programs, describe the decomposition techniques to break the program into inter-dependent sets, and formally introduce the state transition system of an EQL program.

2.1 EQL program

A real-time expert system given in the form of M EQL rules is defined as:

$$\begin{aligned} r_1 :&\quad A_1 \text{ IF } \mathrm{EC_1} \\ r_2 :&\quad A_2 \text{ IF } \mathrm{EC_2} \\ \dots& \\ r_M :&\quad A_M \text{ IF } \mathrm{EC_M} \end{aligned}$$

where $k = 1 \dots M$, EC_k is the enabling condition of rule k and A_k is an action. An action can be a primitive action or a composite of M_K^S subactions:

$$A_k \equiv L_{k,1} := R_{k,1} \; ! \; \dots \; ! \; L_{k,M_k^S} := R_{k,M_k^S}$$

The subactions are interpreted from left to right. Thus, the subaction with index i can change the value of variable used by the action with index j, where $j > i$.

We will restrict ourselves to the use of two-valued variables only. In this paper, the right-hand-side of the assignments and enabling conditions are thus boolean expressions and can be viewed as predicates on the variables in the program.

Here (Fig.1) we give an example of an EQL program, which will be used throughout the whole paper. For clarity, the example program is intentionally kept very simple. In practice, our method can be used for the systems of much bigger complexity, possibly consisting of several hundreds of rules.

```
PROGRAM eql_program;
VAR
  a, b, c, d : BOOLEAN;
RULES
(* 1 *)    c:=TRUE IF NOT a AND NOT b AND NOT c
(* 2 *) [] b:=TRUE IF NOT b
(* 3 *) [] a:=TRUE IF NOT a
(* 4 *) [] b:=FALSE ! c:=FALSE IF NOT a AND b AND c
(* 5 *) [] a:=FALSE ! b:=TRUE ! c:=TRUE IF a AND
             NOT b AND c
(* 6 *) [] d:=TRUE IF a AND b AND c
(* 7 *) [] d:=FALSE IF a AND b AND NOT c
END.
```

Figure 1: An example of the EQL rule-based expert system.

For ease of discussion, we define three sets of variables for an EQL program:

1. $L_k = \{v \mid v$ is a variable appearing in the left-hand-side of the multiple assignment statement of rule $k\}$ (i.e. for the above EQL program $L_5 = \{a, b, c\}$),

2. $R_k = \{v \mid v$ is a variable appearing in the right-hand-side of the multiple assignment statement of rule $k\}$ ($R_2 = \emptyset$) and

3. $T_k = \{v \mid v$ is a variable appearing in the enabling condition of rule $k\}$ ($T_2 = \{b\}$).

2.2 Finite State Transition System

To develop an optimization method we view an EQL rule-based real-time expert system as a *transition system* $\mathcal{T}$ with a finite set of states. $\mathcal{T}$ is a triple $(\mathcal{S}, \mathcal{R}, \rightarrow)$ where

1. $\mathcal{S}$ is a finite set of states. Assuming a finite set $\mathcal{V}$ of variables $v_1, v_2, \ldots, v_N$ and an ordering of $\mathcal{V}$, $\mathcal{S}$ is the set of all possible Cartesian products of the values of variables;

2. $\mathcal{R}$ is a set of rules $r_1, r_2, \ldots, r_M$ in the rule base of the expert system;

3. $\rightarrow$ is a mapping associated with each $r \in \mathcal{R}$, i.e. a transition relation $\xrightarrow{r} \subseteq \mathcal{S} \times \mathcal{S}$, so that if r is enabled at $s_1 \in \mathcal{S}$ and firing of r at that state s_1 would result in the new state $s_2 \in \mathcal{S}$, we can write $s_1 \xrightarrow{r} s_2$.

A transition system for the program given in Fig.1 is shown in Fig.2.

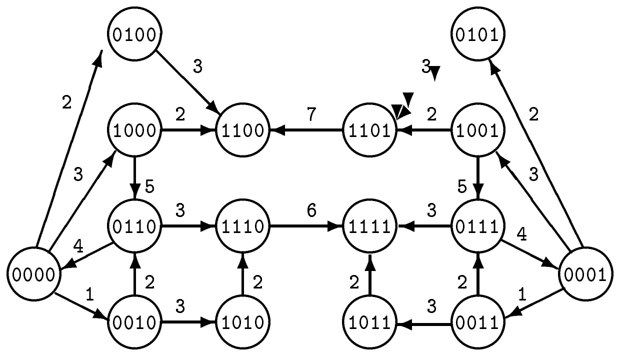

Figure 2: The transition system for the EQL program in Fig.1. Values in each vertex (4 binary digits) correspond to the values of the variables `a`, `b`, `c` and `d`. There are two fixed points: `1100` and `1111`.

The cardinality of $\mathcal{S}$ grows exponentially with the number of variables. Even for a small EQL program the complexity of transition system is quite big and this reduces the possibility for time and space-efficient analysis of the system. For this reason we introduce the *assertion based finite state transition system* $\mathcal{T}^R$.

$\mathcal{T}^R$ is a triple $(\mathcal{S}^R, \mathcal{R}, \Rightarrow)$ where R is as defined above and

1. $\mathcal{S}^R$ is a set of assertions about states in S, so that for $s^R \in \mathcal{S}^R$ and $s \in \mathcal{S}$ $s^R(s)$ holds if state s in T is mapped to state s^R in T^R;

2. $\Rightarrow$ is a mapping associated with each $r \in \mathcal{R}$, i.e. a transition relation $\overset{r}{\Rightarrow} \subseteq \mathcal{S}^R \times \mathcal{S}^R$. Having $s_1 \in \mathcal{S}$, $s_2 \in \mathcal{S}$ and $r \in \mathcal{R}$ so that $s_1 \xrightarrow{r} s_2$, there are corresponding states $s_1^R \in \mathcal{S}^R$ and $s_2^R \in \mathcal{S}^R$ so that both $s_1^R(s_1)$ and $s_2^R(s_2)$ hold and $s_1^R \overset{r}{\Rightarrow} s_2^R$.

When $EC_k(s^R)$ holds, $s^R \in \mathcal{S}^R$, rule r_k is enabled for all states s for which $s^R(s)$ holds.

In contrast with the states in $\mathcal{T}^R$, the states in $\mathcal{T}$ describe a specific (and only one) state of a system. According to this we will also refer to them as *primitive states*. The states in $\mathcal{T}^R$ may include one or more primitive states and we will refer to them as *aggregate states*.

$\mathcal{T}$ is a special case of $\mathcal{T}^R$, where each state in $\mathcal{S}^R$ would be an assertion for a single state in $\mathcal{S}$. $\mathcal{T}^R$ may have as many states as $\mathcal{T}$. However, using the method described in the next section we can, depending on the given EQL system, construct $\mathcal{T}^R$ of much smaller complexity.

For a transition system $\mathcal{T}$ we define a set of *fixed points* as:

$$\mathcal{F} = \{f : \ f \in \mathcal{S} \wedge s^* \in \mathcal{S} \wedge s \neq s^* \wedge \nexists r \in \mathcal{R} \wedge s_1 \xrightarrow{r} s_2\}$$

$\mathcal{F}^R$ is a set of assertions about fixed points in $\mathcal{T}$, so that for every $f^R \subset \mathcal{F}^R$ and every $f \in \mathcal{F}$ $f^R(f)$ holds. Similarly, for every $f \in \mathcal{F}$ there exists $f^R \in \mathcal{F}^R$ so that $f^R(f)$ holds. In Fig.2 and 6, fixed points can be easily identified and are represented by those vertices that are without outgoing edges.

A launch state is either an initial state of the system or a state obtained from a fixed-point by replacing the input variable components with any combination of input variable values (note that for simplicity, we do not distinguish between program and input variables, as is the usual case with an EQL program).

We say that s_j is reachable from s_i if there exists a path from s_i to s_j. We denote the reachability with $s_i \xrightarrow{*} s_j$ ($s_i \overset{*}{\Rightarrow} s_j$ for reachability in $\mathcal{T}^R$).

Note that in general the transition system of an EQL program can contain cycles and redundant rules, and the fixed point may be unreachable from some of the launch states.

2.3 Decomposition of an EQL Program

We refer to decomposition techniques for EQL programs as given in [3], which are based on *rule independency*. Rule b is said to be independent from rule a if the following conditions hold:

I1. $L_a \cap L_b = \emptyset$.
I2. rule a does not potentially enable rule b.
I3. $R_b \cap L_a = \emptyset$.

The independent rule-sets are determined from the high-level dependency (HLD) graph. This is constructed from the rule-dependency graph which consists of vertices (one for every rule) and directed edges (i.e. connecting vertices a to b if rule b is not independent from rule a). All vertices belonging to the same strongly connected component are then grouped into a single vertex. The graph thus obtained is called a *high-level dependency graph* and each of its vertices is called a *forward independent rule-set*.

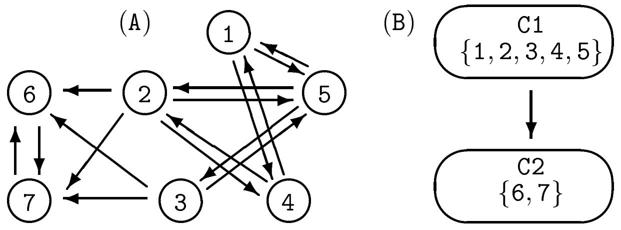

Figure 3: Rule-dependency graph (A) and HLD graph constructed from it (B) for EQL program in Fig.1.

Rules can now be fired by following the topological ordering of the vertices (rule sets) and firing the rules in each vertex until a fixed point is reached. If the EQL program is guaranteed to reach the fixed point from every launch state, the above uniprocessor rule schedule will guarantee the program will reach a fixed point as well [3].

If the optimization technique maintains the assertion about fixed-point reachability for every independent rule-set, each rule-set can be optimized independently. The above decomposition method was evaluated in [3] and the results encourage us to use this method to substantially reduce the complexity of the optimization process.

3 Method

We designed an optimization method (see [2]) which takes an *original* rule-based expert system as input and derives a new *optimized* expert system. The method ensures the correctness of the derived expert system in terms of reaching the single accurate fixed point for each launch state and faster response time in terms of the number of rule firings. The semantics of the original system is altered since the method guarantees that a launch state will have just a single corresponding fixed point, which would be arbitrarily chosen from the set of original fixed point.

Our optimization method consists of two main steps: construction of an assertion-based finite state transition system $\mathcal{T}^R$ and synthesis of a new EQL rule-based expert system from the derived $\mathcal{T}^R$ (Fig.4). The complexity of these two phases is reduced by optimizing only one independent rule-set at a time.

```
1:  read in the original EQL program $\mathcal{P}$
2:  construct HLD graph and identify independent
    rule-sets for $\mathcal{P}$
3:  **forall** independent rule-sets in $\mathcal{P}$ **do**
4:      construct $\mathcal{T}^R$
5:      synthesize EQL program $\mathcal{P}^R$ from $\mathcal{T}^R$
6:      output $\mathcal{P}^R$
7:  **end forall**
```

Figure 4: Derivation of the optimized EQL program.

3.1 Construction of reduced assertion-based transition system $\mathcal{T}^R$

$\mathcal{T}^R$ is derived directly from the original rule-set of the EQL program. The derivation algorithm (Fig.5) combines bottom-up and breadth-first search strategy. It starts at the assertion state for the fixed points and gradually expands $\mathcal{T}^R$ until all states that can reach fixed points are found.

The algorithm uses the variable S, which stores a set of all possible expansion states for the current transition system. S is constructed by considering all aggregate states in *ToExpand* and finding all rules that their (single) firing can result in that state. For each such tuple (s, r), where expanded state $s \in ToExpand$ and $r \in \mathcal{R}$, the expansion state $s'_{s,r}$ is found so that $s'_{s,r} \overset{r}{\Rightarrow} s,\ r \in \mathcal{R}$.

Note that the aggregate state $s'_{s,r}$ should be mutual exclusive with all of the states currently in *States* ($\forall s^* \in S \,.\, s^* \wedge s'_{s,r} \equiv$ FALSE). This constraint ensures the constructed system to be cycle-free.

In each iteration the *most general* aggregate state from *States* is chosen. The generality of a state is assumed to be proportional to the number of primitive states in the aggregate state. The algorithm terminates when no more expandable states are found, i.e. when the transition system includes all the states that can reach the fixed point.

The above algorithm will minimize the number of states grouping equivalent primitive states into a single aggregate states. Each transition corresponds to the firing of a single rule.

To further reduce the number of the states in the transition system, we use the technique that originates from the notion of *intra-rule-set parallelism* [3] and from the idea of utilizing the concurrency for preventing the state explosion [9]. We will allow transitions to be labeled with a set of rules c rather than with single rule r. For every state $s \in States$ we find all possible rule-sets, so that for particular set c each of the rules $r_i \in c$ can be used for expansion of s. Furthermore, for every pair of rules $r_i, r_j \in c$, $r_i \neq r_j$, the following conditions should hold:

D1. rules r_i and r_j do not potentially disable each other.

D2. $L_{r_i} \cap L_{r_j} = \emptyset$ or $L_{r_i} \cap L_{r_j} \neq \emptyset$ and the same expression is assigned to the variables of the subset.

D3. $L_{r_i} \cap R_{r_j} = \emptyset$ and $L_{r_j} \cap R_{r_i} = \emptyset$.

D1 follows the idea of intra-ruleset parallelism [3]. **D2** and **D3** guarantee the cycle-free firing of the two rules at primitive states in the expansion state $s'_{s,c}$ (for a detailed proof, see [10]).

While the introduction of multiple-rule transitions minimizes the number of states in the transition system, we can not determine the tight response time (maximum number of rules to be fired to reach the fixed point) for such system directly. Namely, considering the transition $s_1 \overset{c}{\Rightarrow} s_2$, it is not trivial to determine the lower bound for the rules in c to fire at s_1. For this we have to derive a single-transition system rooting at s_2 and including all the primitive states from s_1. All the transitions in such system would be labeled with the rules from c and the longest path to s_2 would correspond to the tight number of rules to be fired for the transition $s_1 \overset{c}{\Rightarrow} s_2$.

Different firing sequences of rules that are enabled in the same state when the multiple-rule transitions are used can lead to different fixed points. Due to the mutual-exclusivity of states, this is not true for a single-rule transition system, since no two rules can be enabled at the same primitive state.

The constructed transition diagrams for the EQL program from Fig.1 are shown in Fig.6. The assertion-based transition diagrams in Fig.6 are considerably

```
procedure $Construct\_\mathcal{T}^R()$
1:  $States$ := FixedPoints(); $Edges$ := $\emptyset$
2:  $ToExpand$ := $State$; $NextToExpand$ := $\emptyset$
3:  repeat
4:     repeat
5:        $S := \{s'_{s,r} \mid s \in States,\ s'_{s,r} \cap States \neq \emptyset,$
                 $s'_{s,r} \overset{r}{\Rightarrow} s,\ r \in \mathcal{R}\}$
6:        if $S \neq \emptyset$ then
7:           choose most general state $|s'_{s,r}| > 0$ from $S$
8:           $Edges := Edges \cup s'_{s,r} \overset{c}{\Rightarrow} s$
9:           $States := States \cup \{s'_{s,r}\}$
10:          $NextToExpand := NextToExpand \cup s'_{s,r}$
11:       end if
12:       remove states from $ToExpand$ that
          can not be expanded
13:    until $|s'_{s,r}| = 0$
14:    ToExpand := NextToExpand
15: until $ToExpand = \emptyset$
```

Figure 5: Construction of $\mathcal{T}^R$ with single-rule transitions using breadth-first bottom-up derivation algorithm.

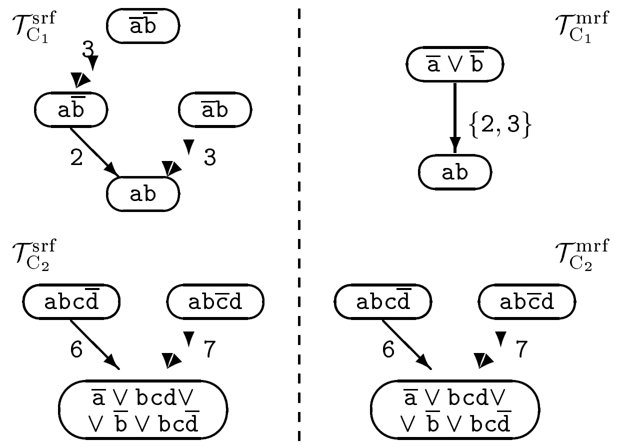
$\mathcal{T}^{\mathrm{srf}}_{\mathrm{C}_1}$
$\overline{\mathtt{a}}\overline{\mathtt{b}}$
3
$\mathtt{a}\overline{\mathtt{b}}$
$\overline{\mathtt{a}}\mathtt{b}$
2
3
ab
$\mathcal{T}^{\mathrm{mrf}}_{\mathrm{C}_1}$
$\overline{\mathtt{a}} \vee \overline{\mathtt{b}}$
{2, 3}
ab
$\mathcal{T}^{\mathrm{srf}}_{\mathrm{C}_2}$
$\mathtt{abc}\overline{\mathtt{d}}$
$\mathtt{ab}\overline{\mathtt{c}}\mathtt{d}$
6
7
$\overline{\mathtt{a}} \vee \mathtt{bcd} \vee$
$\vee\ \overline{\mathtt{b}} \vee \mathtt{bc}\overline{\mathtt{d}}$
$\mathcal{T}^{\mathrm{mrf}}_{\mathrm{C}_2}$
$\mathtt{abc}\overline{\mathtt{d}}$
$\mathtt{ab}\overline{\mathtt{c}}\mathtt{d}$
6
7
$\overline{\mathtt{a}} \vee \mathtt{bcd} \vee$
$\vee\ \overline{\mathtt{b}} \vee \mathtt{bc}\overline{\mathtt{d}}$

Figure 6: Assertion-based transition diagrams constructed for the example EQL program (Fig.1). Both single- and multiple-rule (srf and mrf) solutions are shown for both independent rule-sets (C_1 and C_2). Fixed points (both in the cases of srf and mrf) are ab and $\overline{\mathtt{a}} \vee \mathtt{bcd} \vee \overline{\mathtt{b}} \vee \mathtt{bc}\overline{\mathtt{d}}$.

simpler than the one shown in Fig.2. This may not always be the case, nonetheless the experiments performed lead us to believe that for standard rule-based expert systems, the corresponding constructed $\mathcal{T}^R$s would not be exhaustibly complex in terms of space requirements.

3.2 Synthesis of an optimized EQL program

A new EQL program is synthesized from the constructed $\mathcal{T}^R$s. For each rule in the independent rule-set, the new enabling condition is determined by scanning through $\mathcal{T}^R$ so that for rule i and multiple-rule transitions system:

$$EC_i^{New} = (\bigvee_{s_o \overset{i}{\Rightarrow} s_d} s_o) \wedge EC_i$$

In the case of single-rule transitions system, the conjunction term EC_i can be omitted in the above expression.

The new rules are then formed with the same assignment parts as the original rules and the new enabling conditions. Rules not included in any of constructed $\mathcal{T}^R$s are redundant and will not be added to the new rule-base.

For our example, the optimized EQL program constructed from $\mathcal{T}^R$s with multiple-rule transitions from Fig.6 is shown in Fig.7. Note that redundant rules 1, 4 and 5 are not included in the constructed transition system.

```
PROGRAM optimized_eql_program;
VAR
  a, b, c, d : BOOLEAN;
RULES
  (* 2 *) [] b:=TRUE IF NOT b
  (* 3 *) [] a:=TRUE IF NOT a
  (* 6 *) [] d:=TRUE IF a AND b AND c
  (* 7 *) [] d:=FALSE IF a AND b AND NOT c
END.
```

Figure 7: An example of the EQL rule-based expert system.

Concluding the example we have to point out that the synthesized EQL program has fewer rules than the original and it is easy to determine that the maximum number of rule firings to reach a fixed point is 3 (since enabling conditions for rule 6 and 7 are mutually exclusive). The resulting program does not have any cycles.

It is not always the case where the synthesized enabling conditions are as simple as the one in our example. In general, they are more complex than the original one due to the greater specificity of the synthesized rules.

The problem of enabling condition complexity can be solved by iteratively changing the generated $\mathcal{T}^R$s and observing the impact of the changes by synthesizing the output EQL program. We are considering simulated annealing [8] to be an appropriate method to be used for the optimization.

4 Experimental Results

To demonstrate the applicability of our method, we used it to optimize several EQL programs. The metrics used are the number of rules fired to reach the fixed point and the complexity of the rules (enabling conditions) before and after the optimization. To evaluate the performance of the non-optimized EQL program, we used the Estella - General Analysis Tool[4]. Estella determines the upper-bound on the number of rule firings for a cycle-free (stable) EQL program. The complexity of an enabling condition was determined by counting the number of operations (conjunctions, disjunctions and negations) that are to be performed to evaluate the conditions. The optimization does not alter the action part of the rules. For this reason, the complexity of the action part has not been considered in this analysis.

In [6], Estella was used to analyze the Integrated Status Assessment Expert System (ISA) and the Fuel Cell Monitoring Expert System (FCE). For both expert systems Estella identifies the cycles of rule-firings and therefore is not able to determine the number of rules to be fired to reach a fixed point. We have used our optimization program on both mentioned expert systems. Due to the rule-firing cycles, besides obtaining an equivalent stable cycle-free system, we could not evaluate the improvement in the time-response.

To be able to compare the time-response of unoptimized and optimized systems, we have developed a tool for randomized generation of EQL programs. Given the input parameters, the tool generates the cycle-free EQL programs that could be analyzed with Estella. We evaluated seven such different programs. Since the independent rule-set is the basic unit for both the Estella and our method, we have generated the program with a single independent rule-set. Table 1 shows the results of the evaluation of the optimization.

Two different assumption based transition system techniques were investigated: (1) single-rule transition case allowed us to determine the tight response

	#rules			#vars	#fired		av. #operations/rule			av. #vars/rule			#states in $\mathcal{T}^R$	
	U	OS	OM	all	U	OS	U	OS	OM	U	OS	OM	OS	OM
P1	5	3	3	5	5	2	1.80	13.0	14.5	1.80	4.333	5.0	6	3
P2	5	4	4	5	4	3	7.20	11.50	12.5	3.80	4.000	4.25	11	3
P3	8	6	7	5	5	2	1.50	9.167	14.2	1.625	5.000	4.714	10	3
P4	12	6	10	5	5	3	1.833	11.50	13.5	1.833	4.833	4.600	15	3
P5	10	7	8	5	5	3	2.30	10.0	10.7	2.100	5.00	5.000	15	2
P6	10	8	9	10	7	4	7.20	121.2	229.8	4.500	10.00	9.88	39	2
P7	15	11	13	10	7	4	6.733	179.3	157.3	4.266	9.272	9.615	53	3

Table 1: Results of the experiments given for the unoptimized (U) and optimized programs synthesized from transition systems with single (OS) and multiple-rule (OM) transitions. #rules and #vars are respectively the total number of rules and variables in the program. The number of operations and variables per rule are averaged for all of the rules in the generated program. #fired is the maximum number of rules to be fired to reach a fixed point.

bound, and (2) the multiple-rule transition case highly reduced the size of the assumption based transition system.

The optimization always resulted in a reduction of the number of rules to be fired to reach a fixed point in respect to the one determined by Estella from the unoptimized program. It was expected that the optimized programs will have more complex enabling conditions. For most of these programs, the total number of rules was changed, indicating that some rules are redundant (compare this to our example in the previous section).

A surprisingly low number of states in the assertion-based transition diagram for programs P6 and P7 have been observed. Note that even for a small program with 10 variables, there are potentially $2^{10} = 1024$ states in the non-reduced transition diagram.

In general, based on the analysis of the systems with single-rule transitions, the performance was improved from 25% to 50% with regard to the number of rule firings while the enabling condition complexity was increased. Although the complexity of the enabling conditions was in general increased, the fact the number of rules in the new rule-base was in all cases reduced leads us to believe that this would not have a major impact to the performance. Besides, the performance is enhanced due to the cycle freedom of the synthesized programs. Furthermore, we have to stress that all optimized programs are stable and thus are guaranteed to reach a fixed point in a determined bounded number of rule firings.

5 Conclusion

In this paper, we have described a novel approach to the optimization of rule-based expert systems. Our method is based on a construction of the reduced finite state transition system corresponding to the input rule-based system. Our focus is on the optimization of the rules' enabling conditions while leaving the rules' assignments unchanged.

The new and optimized rule-based expert system is synthesized from the derived transition system. The states in the transition system are mutually exclusive. This, together with the cycle-free nature of the transition system's state diagram, contributes to the special properties of the rule-based system constructed from it.

In comparison with the original system, the synthesized rule-based systems have fewer number of rule firings to reach the fixed point. The rule-based systems constructed by the proposed method contain no cycles and thus are inherently stable. Redundant rules present in the original systems are either changed or removed.

For programs generated from systems with single-rule transitions, each launch state of such rule-based systems has exactly one fixed point which makes the systems deterministic. The result of the execution of such systems no longer depends on the order of the sequence of rule firings for the rules enabled at the same time. This makes the system tolerant to the different scheduling techniques and, once tested, it can guarantee the same results under different execution environments.

We have shown the evaluation of the method through its use for the optimization of several sample rule-based programs. The development of our systematic approach to tackle the optimization problem opens up new avenues to further enhance the runtime performance of rule-based systems in time-critical environments. Ongoing work is also done to apply the proposed techniques to a variety of rule-based systems.

References

[1] A. Bouajjani, J.-C. Fernandez, and N. Halbwachs. Minimal model generation. In E. M. Clarke and R. P. Kurshan, editors, *Proc. 2nd Int'l Conference on on Computer-Aided Verification*, pages 197–203, New Brunswick, NJ, June 1991.

[2] J. C. Browne, A. M. K. Cheng, and A. K. Mok. Computer-aided design of real-time rule-based decision systems. Technical report, Department of Computer Science, University of Texas at Austin, 1988. Also to appear in *IEEE Transactions on Software Engineering*.

[3] A. M. K. Cheng. Parallel execution of real-time rule-based systems. In *Proc. IEEE Int. Parallel Processing Symposium*, pages 779–789, Newport Beach, CA, April 1993.

[4] A. M. K. Cheng, J. C. Browne, A. K. Mok, and R.-H. Wang. Analysis of real-time rule-based system with behavioral constraint assertions specified in Estella. *IEEE Transactions on Software Engineering*, 19(19):863–885, September 1993.

[5] A. M. K. Cheng and C.-H. Chen. Efficient response time bound analysis of real-time rule-based systems. In *Proc. 7th Annual IEEE Conf. on Computer Assurance*, pages 63–76, NIST, Gaithersburg, Maryland, 1992.

[6] A. M. K. Cheng and C.-K. Wang. Fast static analysis of real-time rule-based systems to verify their fixed point convergence. In *Proc. 5th Annual IEEE Conf. on Computer Assurance*, pages 197–203, NIST, Gaithersburg, Maryland, June 1990.

[7] L. B. Eliot. If it works, is it good? *AI Expert*, 7(6):9–11, June 1992.

[8] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[9] A. A. Valmari. A stubborn attack on state explosion. In E. M. Clarke and R. P. Kurshan, editors, *Proc. 2nd Int'l Conference on on Computer-Aided Verification*, pages 156–165, New Brunswick, NJ, June 1991.

[10] B. Zupan. Optimization of real-time rule-based systems using state-space diagrams. Master's thesis, University of Houston, Department of Computer Science, 1993.