Response Time Optimization of Rule-Based Expert Systems*

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

Abstract

Real-time rule-based decision systems are embedded AI systems and must make critical decisions within stringent timing constraints. In the case the response time of the rule-based system is not acceptable, it has to be optimized to meet both timing and integrity constraints. This paper describes a novel approach to reduce the response time of rule-based expert systems. Our optimization method is twofold: the first phase constructs the reduced cycle-free finite state transition system corresponding to the input rule-based system, and the second phase further refines the constructed transition system using the simulated annealing approach. 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.

1 Introduction

Modern, in particular computer based, technology is increasingly used in the environments where events are not predictable, the state of the world is not easily characterized in advance, and the time-response requirements are critical. Real-time systems belong to this class of applications and our main interest is in rule-based real-time systems and the problems posed by stringent time requirements these systems have to satisfy.

The rise of the use of rule-based, real-time expert systems as embedded AI systems in different industrial applications, such as airplane avionics, smart robots, space vehicles and other safety critical applications increases the importance of the validation and verification phase of their life cycle. Apart from functional correctness, real-time expert systems must also satisfy stringent timing constraints. The result of missing a deadline in these systems may be fatal and the verification task is to prove that the system can deliver an adequate performance in a 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.

The definition of the synthesis problem given by Browne et al. in [2] is: "Given an equational rule-based program P that always reaches a safe fixed point in finite time but is not fast enough to meet the timing constraints under a fair scheduler, determine whether there exists an extension of P that meets both the timing and integrity constraints under some scheduler."

We propose a formal framework to a response-time optimization of the rule-based expert system given in the form of rule-based equational logic programs (EQL) as introduced by Browne et al. in [2] and Cheng and Wang in [6]. Our method is based on the identification of the finite state transition system for the original rule-based program. Because of the nontrivial nature of expert systems, the combinatorial explosion of states may threaten the implementability of such an approach. To tackle the complexity problem, we use several different approaches, the most important of them being: (1) the minimal model generation method by Bouajjani et al. [1], (2) the stubborn attack on state explosion using the notion of concurrency by Valmari [10], and (3) the equational program decomposition method and concurrent rule execution algorithm by Cheng [4].

Bouajjani et al. [1] proposes an algorithm for *minimal model generation* that can be used for building a state space graph of transition systems. The algorithm

combines generation and reduction methods. He uses a notion of the equivalency of states and instead of reducing a transition system from its non-reduced version introduces a unique approach of reducing the system during its generation. A *stubborn attack on state explosion* by Valmari is another method for state reduction. In [10] he argues that concurrency is the major contributor to state explosion. It introduces a large number of execution sequences that lead from a common start state to a common end state by the same transition, but the transition occurs in different order causing the sequences to go through different states. We use both approaches in combination with Cheng's [4] parallel execution model. The later is used for rule-base system decomposition and reduction of complexity of optimization, and in combination with Valmari's approach to reduce the number of states in the transition system.

Since the proposed optimization usually increases the complexity of rules (enabling conditions), we introduce an additional optimization phase. It is based on the known method of simulated annealing [8] and attempts to reduce both the number of rules to be fired and the complexity of enabling conditions.

The optimized rule-based system is synthesized from the derived transition system. The system is cycle-free and has optimal performance in terms of number of rules to be fired to reach the fixed point.

The rest of the paper is organized as follows. In Section 2 we outline the syntax of EQL rule-based programs, introduce the transition system representation and describe the rule-based programs decomposition techniques. Section 3 describes the proposed optimization technique, which is ahen discussed in Section 4. The Additional optimization phase is described in Section 5. Section 6 is the conclusion.

2 Prerequisites

The real-time decision system investigated here is described by a model introduced by Browne et al. [2].The system interacts with the external environment by taking sensor readings and computing control decisions based on sensor readings and stored state information. This model is depicted in Fig.1.

The decision system is represented as an equational **EQL** rule- based program [2, 3]. This consists of a finite set of M rules each of which has three parts:

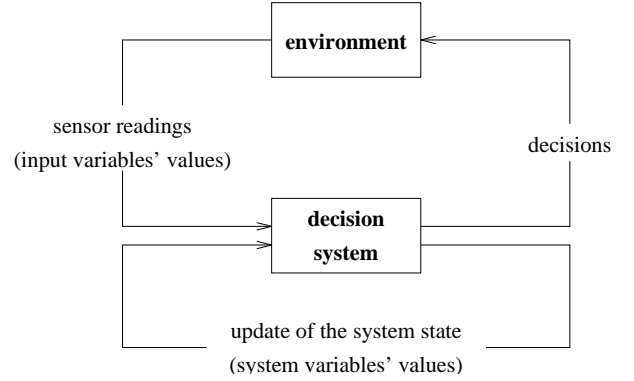1. LHS: the left-hand-side of a multiple assignment statement,

Figure 1: The real-time decision system.

2. RHS: the right-hand-side of a multiple assignment statement, and

3. EC: the enabling condition.

An enabling condition is a predicate on the variables in the program. For ease of discussion, we define three sets of variables for a rule r_k, $1 \leq k \leq M$ of an EQL program:

1. $L_k = \{v | v$ is a variable appearing in LHS of $r_k\}$,

2. $R_k = \{v | v$ is a variable appearing in RHS of $r_k\}$, and

3. $T_k = \{v | v$ is a variable appearing in EC of $r_k\}$.

An example of an EQL program is shown below (Fig.2. We will restrict ourselves to the use of two-valued variables only. Note that EQL programs that use multivalued variables instead can be easily (automatically) converted to it. The assignment statements are interpreted from left to right.

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

Figure 2: An example of the EQL rule-based expert system.

The transition system of an EQL program, as defined in [2] is a labeled graph $G = (V, E)$. V is a set of vertices each of which is labeled by a tuple: $(x_1, ..., x_n, s_1, ...s_p)$ where x_i is a value in the domain of ith input sensor variable and s_j is a value in the domain of the jth program variable. E is a set of edges each of which denotes the firing of a rule. An edge (i, j) connects vertex i to vertex j iff there is a rule r which is enabled at vertex i and firing r will modify the program variables to have the same values as the tuple at vertex j. We will denote this with $i \xrightarrow{r} j$.

Fig.3 shows a transition system for a program in Fig.2. There are two vertices of special type (`001` and `010`) that have no outgoing edges and are said to be *fixed points*. Fixed points are states in the system where no rule is enabled or, if so, firing one will not change the state. Note that in our transition system we do not mark the self loops, i.e. cycles of type $i \xrightarrow{r} i$.
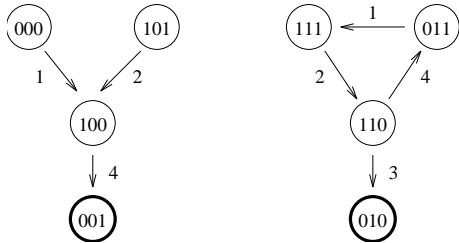

Figure 3: A transition system for the EQL program in Fig.2.

An invocation of an equational rule-based program can be thought of as tracing a path in the transition system. Every time the values of the input variables are changed (at that point the system is in a so-called *launch state*) the values of the system variables have to be determined through iterative firings of rules. The *fixed point* is reached when no variable's value changes. The required property of hard real-time rule-based system is reaching the fixed point before the next change of the values of input variables. In accordance to this, Browne et al. [2] define a timing constraint as an upper bound on the length of the paths from a launch state to a fixed point, and this corresponds to the *response time* of the EQL program.

The vertices of the above defined transition system describe a specific (and only one) state of the system. According to this we will also refer to them as *primitive states*. In the next section we will introduce the notion of equivalent states. These states will group equivalent primitive states into an aggregate states.

I.e. having two aggregate states $I = \{i_1, ..., i_m\}$ and $J = \{j_1, ..., j_m\}$, where $i_1, ..., i_m, j_1, ..., j_m$ are primitive states, a relation $I \xrightarrow{r} J$ would mean that for every primitive state $i_x, x \in 1..m$ in I there exists primitive state $j_y, y \in 1..n$ so that $i_x \xrightarrow{r} j_y$.

The parallelization of the rule-based system is interesting from the viewpoint of speeding-up the execution as well as for the decomposition of the system. Our main interest is in the later, since it makes the *divide and conquer* approach feasible to lower the complexity of the analysis and synthesis of the system. We refer to the parallel execution model proposed by Cheng [4]. He uses the notion of *inter-ruleset parallelism* to decompose the EQL program into inter-dependent rule-sets. We use his model to optimize the rules in each such rule-set separately and thus decrease the complexity of the optimization phase.

3 Method

Traditional methods [2] construct an expanded transition system that includes all the states and transitions and then use various reduction and optimization methods to construct a reduced transition system. They would then use the model checker for the temporal logic CTL [7] to check the integrity and timing constraints. This approach is potentially very complex because of the state explosion problem. Note that the system with N two-valued variables would have 2^N different states.

We here present an optimization method that derives the reduced transition system directly from the rule-based program and can thus avoid the state explosion. The new and optimized equational rule-based program is then synthesized from the derived transition system.

Despite the fact that the reduced transition system can be used also for analysis purposes, our main goal is to use it for the synthesis of the new EQL program. The derived EQL program will have:

- improved response time: if there would be two or more different paths from a launch state to the fixed point in the original non-reduced transition system, only the shortest one will be present in the reduced transition system,

- no cycles: all states of the system are stable, and

- no redundant rules/transitions.

The method ensures the correctness of the derived expert system in terms of reaching the single accurate
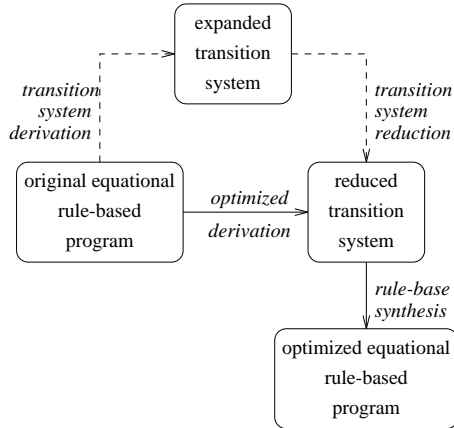
Figure 4: Optimization of equational rule-base program. The steps taken by our optimization method are marked with solid line, while traditional and more complex optimization method is denoted by a dashed line.

fixed point for each launch state. 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 points.

To construct a reduced transition system, several techniques are combined. We first use a bottom-up derivation method, which starts from the fixed points and then gradually adds other states and transitions to the system. To decrease the complexity of the transition system, we identify the equivalent states while building the transition system. The third technique employs further reduction in the number of states and transitions combining those rules that can be fired in parallel. This section presents each of the above techniques and discusses the synthesis method.

3.1 Bottom-Up Derivation of Transition System

Our bottom-up derivation starts at fixed points of the system and gradually builds the transition system iteratively adding new states. Each state is added to the system only once, and the resulting transition system will thus have states/vertices with at most one outgoing transition/edge. The algorithm is depicted in Fig.5.

In the above algorithm, the states used for the expansion of the current transition system are arbitrarily chosen. Each original (expanded) transition system

procedure $BottomUpDerivation()$
1:     $\mathcal{T} = \{ f \mid f \text{ is a fixed point } \}$
2:     **while** ($\exists s \notin \mathcal{T}$ and $\exists s' \in \mathcal{T}$ so that $s \xrightarrow{r} s'$)
3:       add state s and transition $s \xrightarrow{r} s'$) to $\mathcal{T}$
4:     **end while**

Figure 5: Bottom-up derivation of transition system.

may thus have different corresponding bottom-up constructed transition systems. We do not construct the transition system from the existent (expanded) one but rather use the rule-based program and a set of corresponding fixed points. Therefore, we have to apply the rules "in reverse manner", meaning that given the state s' and rule r we have to find a state s so that $s \xrightarrow{r} s'$). We have developed a set of functions that allow us to do so. Interested reader should refer to [11] for a detailed description.

Fig.6 shows an example of the original and corresponding bottom-up transition system, which has, in respect to the original one, the following properties:

- no cycles (all the cycles from the original transition system are removed),

- determinism: each state has only one corresponding fixed point,

- the states that do not have a path to any of the fixed points are eliminated, and

- substantial number of transitions are eliminated.

To reduce the response time we can combine the bottom-up derivation with the breadth-first search approach. We assume the fixed points to be at level 0 and other states to be at level l, where l is a number of transitions between the specific state and its corresponding fixed point. Breadth-first search technique will add all the lower level states to the transition system prior to the higher level states. Such derivation will result in constructing only the shortest paths from a certain state to its corresponding fixed point.

3.2 Equivalent States

Our bottom-up derivation reduces the number of states in the transition system by excluding the states that do not have a corresponding path to the fixed points. The number of states can be further reduced with combining the equivalent states into the single aggregate state. We follow the idea of Bouajjani et al. In [1] they proposed an algorithm which, instead
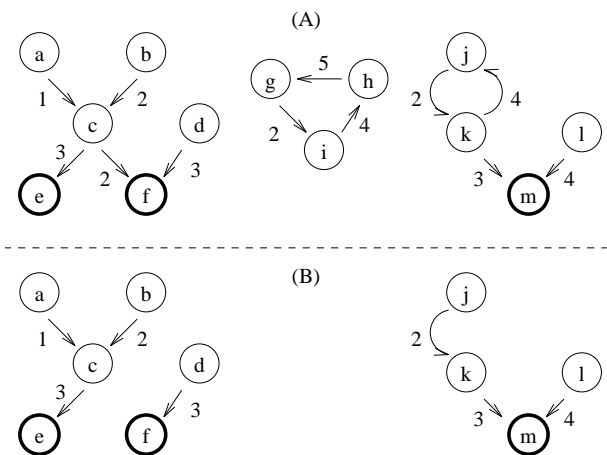
Figure 6: An example of original (expanded) transition system (A) and corresponding bottom-up constructed transition system (B).

Figure 7: Transition system with grouped equivalent states constructed from the system in Fig.6.A.

of reducing a transition system from its non-reduced version, performs a reduction during its generation.

We use the algorithm proposed by Bouajjani et al. and combine it with breadth-first search derivation. The initial transition system consists of a single aggregate state which includes all the fixed points. Again, the transition system is gradually expanded, this time grouping all the states that lead to the same aggregate state using a specific transition. I.e., having a state $I \in \mathcal{T}$, where $\mathcal{T}$ is a current transition system under generation, we can expand $\mathcal{T}$ with aggregate state J and transition r, so that $I \xrightarrow{r} J$. All the aggregate states in the new transition system have to be mutually exclusive, i.e. $\nexists s . s \in I \wedge s \in J . I \neq J$, where s is some primitive state and I and J are aggregate states in the transition system.

As before, we use the breadth-first search technique. We also use a heuristic technique that adds to the transition system first those aggregate states that include the highest number of primitive states. Ties are broken arbitrarily.

Fig.7 shows a transition system with equivalent states grouped into aggregate states that corresponds to the transition system from Fig.6. The number of states in the transition system is reduced from 10 to 5, while the original non- reduced transition system included 13 states.
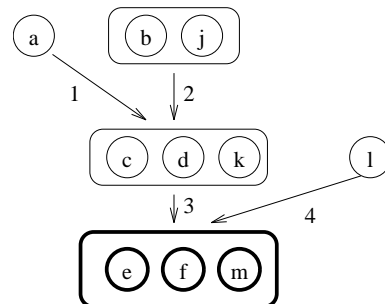
3.3 Reduction of Transition System Due to Rule Execution Concurrency

The transition systems introduced above have *single-rule transitions*. We here present a method that can generate a transition system with transitions labeled with one or more rules (*multiple- rule transitions*).

Suppose we have two aggregate states I and J, and a set of rules $R = \{r_1, ..., r_k\}$ that are all enabled at I, i.e. $\forall r_i \in R . \exists s \in I$ so that r_i is enabled at s. We can write $I \xrightarrow{R} J$) if $\forall s \in I . \exists s' \in J$ so that $s \xrightarrow{r} s'$. For rules in R we impose the following constraints:

C1. rules in R can be fired in parallel, and

C2. firing the rules in R should not result in a cycle that would include only the primitive states in I.

For two rules r_a and r_b both belonging to R, the formalization of the above constraints is:

D1. rule r_a and r_b do not potentially disable each other,

D2. $L_a \cap L_b = \emptyset$, or $L_a \cap L_b \neq \emptyset$ and the same expression is assigned to the variables of the subset, and

D3. $L_a \cap R_b = \emptyset$ and $L_b \cap R_a = \emptyset$.

D1 follows the idea of intra-ruleset parallelism introduced by Cheng [4] (**C1**), and **D2** and **D3** guarantee the cycle- freedom (**C2**).

The transition system generation algorithm is now more complex so that not only all the states of the certain level would be considered for the expansion, but also all the possible combinations of rules satisfying **D1** to **D3** would have to be derived and used. Having an aggregate state $S \in \mathcal{T}$ that is considered

for expansion, and a group of rules R, so that each of its firing can result in S, we construct *concurrent execution graph* $\mathbf{CON}(V, E)$. Each vertex in $\mathbf{CON}$ represents a single rule from R and two vertices are connected if $\mathbf{D1}$ to $\mathbf{D3}$ hold for corresponding two rules.

From the graph $\mathbf{CON}$ we construct all possible combinations of rules that can be fired in parallel. The algorithm uses the matrix M representation of $\mathbf{CON}$, where the size of M is $|R|^2$ and M is defined as:

$$M_{i,j} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are neighbors in } \mathbf{CON} \\ 0 & \text{otherwise} \end{cases}$$

The algorithm removes all equal rows and rows that include all corresponding 1's of some other row. Each row of the resulting matrix represents a group of rules to be fired in parallel. Thus, for row i, rules r_j can be fired in parallel iff $M_{i,j} = 1$. In the transition system construction algorithm (Fig.10) we will denote this procedure with $G(\mathbf{CON})$. $R_x \in G(\mathbf{CON})$ would be one of possible rule-sets to be used in a transition.
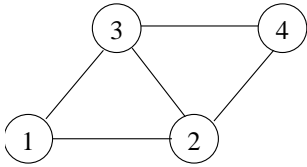

Figure 8: An example of $\mathbf{CON}$ graph.

We here show an example of the derivation of $G(\mathbf{CON})$. Suppose we have a $\mathbf{CON}$ graph for four rules shown in Fig.8. The initial M matrix is:

$$M = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{cccc} 1 & 2 & 3 & 4 \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

and the resulting matrix is:

$$M = \begin{array}{c} \\ 3 \\ 4 \end{array} \begin{array}{cccc} 1 & 2 & 3 & 4 \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

Therefore, $G(\mathbf{CON}) = \{\{1, 2, 3\}, \{2, 3, 4\}\}$ and the transition may be labeled with rules 1, 2 and 3, and 2, 3 and 4.

Considering our example transition system from Fig.7 and knowing that rules 3 and 4 satisfy the constraints $\mathbf{D1}$ to $\mathbf{D3}$, whiles rules 1 and 2 do not, we can construct the reduced transition system as shown in Fig.9.
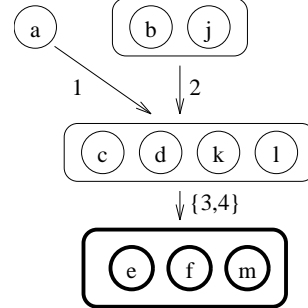

Figure 9: A transition system with multiple-rule transitions and aggregate states.

3.4 Transition System Construction: The Algorithm

The algorithm that combines the approaches introduced above and builds the reduced transition system directly from the original rule- based EQL program is depicted in Fig.10. Variables *States* and *Edges* store the current state of the transition system. *ToExpand* stores the states that are considered to be expanded. Initially, *ToExpand* consists of a single aggregate state that includes all the fixed points. *NextToExpand* stores the states of the next level. In each iteration an aggregate state that stores the biggest number of primitive states is chosen for the expansion. If there is no such state (state includes 0 primitive states), *ToExpand* becomes *NextToExpand* and the next level of states is investigated. The derivation terminates when there are no more states available to expand.

Cover includes all the primitive states currently in the transition system and is used to obtain aggregate states that are mutually exclusive (line 12 in the algorithm).

Despite the fact that the use of multiple-rule transitions reduces the number of states, there is a certain advantage of single-rule transitions systems. Namely, the latter are always deterministic, meaning that at each primitive state of the system at most one rule will be enabled. It has been shown [11] that it is easy to convert multiple-rule transitions system to the one

procedure $TransitionSystem()$
1: $States :=$ FixedPoints$()$; $Cover := States$;
    $Edges := \emptyset$
2: $ToExpand := State$; $NextToExpand := \emptyset$
3: **repeat**
4:   **repeat**
5:     **forall** $S \in ToExpand$ **do**
6:       $Applicable := \{r_i : \exists S' . S \xrightarrow{r} S')\}$
7:       for rules in $Applicable$ construct **CON** graph
8:       **forall** $R \in G(\mathbf{CON})$ **do**
9:         $S'_{S,R} := \cup\, S' - Cover$ where $S \xrightarrow{r} S'$, $r \in R$
10:       **end forall**
11:     **end forall**
12:     chose $S'_{S,R}$ so that $|S'_{S,R}| = $ Max
13:     **if** $|S'_{S,R}| \neq 0$ **then**
14:       $Edges := Edges \cup S'_{S,R} \xrightarrow{R} S$
15:       $States := States \cup \{S'_{S,R}\}$
16:       $Cover := Cover \vee S'_{S,R}$
17:       $NextToExpand := NextToExpand \cup S'_{S,R}$
18:     **end if**
19:   **until** $|S'_{S,R}| = 0$
20:   $ToExpand := NextToExpand$
21: **until** $ToExpand = \emptyset$

Figure 10: Bottom-up breadth-first search construction of reduced transition system.

with single-rule transitions, and we may use two different approaches in combination.

3.5 Synthesis of an Optimized Rule-Based System

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

$$EC_i^{New} = (\bigvee_{S_o \xrightarrow{i} S_d} S_o) \wedge EC_i$$

New rules are then formed with the same assignment parts as in the original rules and new enabling conditions.

4 Discussion

It was shown [12] that the proposed algorithm can be applied to minimize the number of rule firings. The method was used to optimize a set of randomly generated EQL programs of up to 15 rules and up to 10 variables. The Estella - General Analysis Tool proposed by Cheng et al. [5] was used to estimate the response time in regard to the number of rule firings for the original programs. The same metrics were derived directly from the transition systems using the proposed optimization algorithm for the optimized programs. While the optimized programs required on the average 25% to 50% fewer rule firings to reach the fixed point, the enabling condition complexity increased up to 2000%.

The algorithm was also used to optimize the Integrated Status Assessment Expert System (ISA) [9]. We used the two-valued EQL version with 35 rules and 58 variables. The Estella - General Analysis Tool [5] was used to determine the stability of the system and discovered several cycles which led to a termination of the analysis. Thus, the time response of the original program could not be determined and the authors are unaware of any other method that could do so. Still, we have used the ISA expert system to compare the complexity of the unoptimized and optimized programs, and discovered that the later has approximately 10 times more complex enabling conditions.

All this led us to believe that optimization should also address enabling condition complexity and that the response time should not be viewed solely from number of rule firings point of view.

5 Proposal for Additional Iterative Improvement Method

In the previous sections we have shown the optimization technique, which improves the input rule-based program in different ways, but addresses the response time only from the viewpoint of the upper bound of number of rules to fire to reach the fixed point. The optimization process makes the rules' enabling conditions more specific than the original ones and thus increases their complexity. The proposed optimization does not address this issue. Here we introduce the method that attempts to minimize both the number of rules to fire and the complexity of enabling conditions.

The iterative improvement method introduced here starts from the transition system derived by our optimization algorithm and attempts to iteratively improve it. The improvement is defined by means of minimizing the cost function. In each iteration, the method generates a new transition system from the old one and synthesizes the corresponding EQL program. It then compares the cost of both solutions, and if the acceptance criteria is satisfied accepts a new solution.

Owing to its simplicity and potentially good performance [8] we use a simulated annealing optimization method scheme.
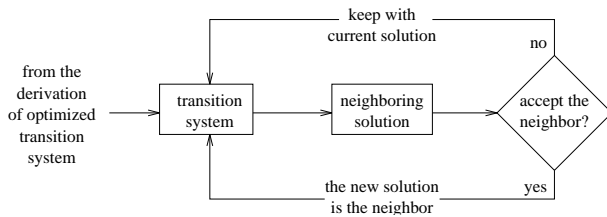

Figure 11: Iterative improvement method.

Fig.11 describes the iterative improvement method. The cost function to be minimized by the simulated annealing program is

$$C = c_1 NRF + c_2 ECC + c_3(NS - NS_0)$$

where NRF is the maximum number of rules to fire to reach a fixed point, ECC represents the complexity of the enabling conditions, NS is the number of states of the newly derived transition diagram, and NS_0 is the number of states in the original transition diagram (output of the greedy or breadth-first algorithm from the previous section). ECC is expressed as the average number of two-valued operations (AND, OR, NOT) to evaluate the enabling condition. Constants c are weights and depending on the type of the optimization preferred can be set accordingly.

The critical phase of the algorithm is derivation of the neighboring solution. The main idea is to *steal* some primitive states from certain aggregate state of the system and either add them to some other aggregate state or create a new aggregate state. The main constraints for the method are that: (1) the resulting system has to be correct, (2) it is cycle-free, and (3) all the primitive states present in the original system have to be present in the neighboring one.

The algorithm for derivation of the neighboring solution is rather complex and lengthy to describe and is presented in detail in [11]. We are currently working on the implementation and evaluation of the impact of such an algorithm.

6 Conclusion

We have presented a new two-phase algorithm for the response time optimization of rule-based expert system. The two-phase algorithm first constructs the reduced and optimized transition system, which encodes all the necessary information about the behavior of the real-time system. Next, the corresponding optimized rule-based system is constructed from the derived transition system. The original and optimized rule-based systems are equivalent: all the launch states of the optimization system reach exactly one corresponding fixed point from the original system. Furthermore, the new system is stable and does not contain cycles, and has optimal performance in respect to the number of rules to fire to reach a fixed point.

The proposed optimization technique would usually increase the complexity of enabling conditions and this might influence the response time of the rule-based program. We propose an additional optimization phase based on the known optimization method of simulated annealing. This uses the derived reduced transition system and attempts to change it so as to minimize both the number of rules to be fired and the complexity of the enabling conditions.

References

[1] A. Bouajjani, J.-C. Fernandez, and N. Halbwachs. Minimal model generation. In E. M. Clarke and R. P. Kurshan, editors, *Proc. of 2nd International Conference, CAV '90*, pages 197–203, New Brunswick, NJ, June 1991. Springer-Verlag.

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

[3] A. M. K. Cheng. *Analysis and synthesis of real-time rule-based decision systems*. PhD dissertation, Dept. of Computer Science, University of Texas at Austin, 1990.

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

[5] 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 Trans. on Software Eng.*, 19(19):863–885, September 1993.

[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] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent system using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, April 1986.

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

[9] C. A. Marsh. The ISA expert system: A prototype system for failure diagnosis on the space station. MITRE report, The MITRE Corporation, Houston, Texas, 1988.

[10] A. Valmari. A stubborn attack on state explosion. In E. M. Clarke and R. P. Kurshan, editors, *Proc. of 2nd International Conference, CAV '90*, pages 156–165, New Brunswick, NJ, June 1991. Springer-Verlag.

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

[12] B. Zupan and A. M. K. Cheng. "Optimization of rule-based systems via state transition system construction". *to be presented at IEEE Conf. on Artificial Intelligence for Applications*, March 1993.