Response-Time Bounds of Rule-Based Programs under Rule Priority Structure *

Rwo-Hsi Wang Aloysius K. Mok
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1188

Abstract

A key index of the performance of a rule-based program used in real-time monitoring and control is its response time, defined by the maximum number of rule firings before a fixed point of the program is reached from a start state. Previous work in computing the response-time bounds for rule-based programs assumes that if two rules are enabled, then either one of them may be scheduled for firing. This assumption may be too conservative in the case programmers choose to impose a priority structure on the set of rules. In this paper, we discuss how to get tighter bounds by taking rule-priority information into account. We show that the rule-suppression relation we previously introduced can be extended to incorporate rule-priority information. A bound-derivation algorithm for programs whose potential-trigger relations satisfy an acyclicity condition is presented, followed by its correctness proof and an analysis example.

1 Introduction

In recent years, there has been increasing interest in the use of rule-based programs in time-critical applications, e.g., monitoring the engine of the Space Shuttle. In these applications, the rule-based program is often used to monitor the environment and must compute an output in response to significant changes that are observed in the environment. Such computation must be completed sufficiently fast so that the system can react to environmental changes in a predictable fashion. The verification/validation of the behavior of rule-based programs is of both practical and theoretical interest. Related work in real-time AI systems can be found in [1] [7] [10] [9] [6].

*The research reported herein is supported by the Office of Naval Research under grant number N00014-94-1-0582 and by a grant from the Texas Advanced Research Program.

In [8], the concept of a response time is defined for the equational rule-based program (EQL) by the (maximum) number of rule-firings before the program reaches a fixed point (a program state where all the program variables attain stable values). It was shown in that paper that the determination of the response time for arbitrary programs is in general undecidable, and is PSPACE-hard for finite-state programs. Recently there have been several papers presenting efficient methods of computing response-time bounds for rule-based programs [5] [4] [11] [3]. However, none of those methods makes use of the rule-priority information in the program. Rule priority is a common feature in rule-based languages. It is used to constrain rule-firing order. With the presence of rule priorities, a program has fewer possible firing sequences, therefore possibly shorter response time, compared with the same program without rule priorities.

In this paper, we address the rule-priority issue in computing response-time bounds. We show that the rule-suppression relation we previously introduced in [11] can be extended to incorporate rule-priority information. Since the rule-suppression relation is useful in deriving tighter bounds and rule priority can now be viewed as a form of rule suppression, making use of such information in bound derivation can result in tighter bounds. A bound-derivation algorithm is presented, followed by its correctness proof. A response-time analyzer is implemented to analyze programs whose potential-trigger relations satisfy an acyclicity relation. An analysis example is given to demonstrate the analysis process.

The remainder of this paper is organized as follows. The EQL language is described in Section 2, together with program state space and program execution. In Section 3 we introduce the basis of our response-time bound analysis theory and extend it to incorporate rule priority. In Section 4 we present an algorithm to compute response-time bounds for programs whose potential-trigger graphs are acyclic. We have implemented an analyzer based on the algorithm. An analysis example is presented in Section 5, followed by concluding remarks and future research topics in Section 6.

2 The EQL language

EQL is a rule-based programming language designed for developing real-time decision systems where the set of decision variables is fixed *a priori* [2]. This seems to be the class of rule-based programs that are most commonly found in real-time expert system applications. An EQL program consists of a finite set of rules each of which is of the form:

$$\textit{priority} \quad v_1 := e_1 \ ! \ v_2 := e_2 \ ! \ \ldots \ ! \ v_m := e_m \ \text{IF} \ \textit{test}$$

where *priority* is an optional specification for the rule's priority (default 0), e_i is an integer expression whose value is to be assigned to variable v_i when the rule fires, $m \geq 1$, and *test* is a predicate on the variables in the program. A rule has three parts:

1. VAR: the left-hand-side variables of the multiple assignment statement,
2. VAL: the right-hand-side expressions of the multiple assignment statement, and
3. EC: the enabling condition.

A rule is *enabled* at some instant if its test is evaluated to true according to the values of its variables at that moment, otherwise it is *disabled.* A rule is *conflictive* at some instant if any of its VAR variable is not equal to the corresponding VAL expression. A *rule firing* is the execution of the multiple assignment statement, which consists of the evaluation of all the VAL expressions, followed by updating the VAR variables with the values of the corresponding expressions. A rule is *firable* iff it is enabled and conflictive.

An *execution* of an EQL program is a sequence of rule firings. When two or more rules are firable, the selection of which rule to fire first is up to the runtime scheduler, which can use a nondeterministic scheduling policy, provided that firable rules with the highest priority are selected first. An EQL program is said to have reached a *fixed point* when none of its rules are firable.

The following example declares three rules in the rule section. The first two rules have the default priority 0; the third rule has a (higher) priority 1.

```
   within_range := false ! type1 := true
         IF x + y > limit OR x + y < -limit
[] within_range := false ! type2 := true
         IF x * x + y * y > limit * limit
[] {P:1} faulty_sensor := true
         IF x > limit OR x < -limit OR
            y > limit OR y < -limit
```

2.1 State-space representation

The *state space* S of an EQL program is the cartesian product of the domains of each of its variables. At each moment during an execution, the state of an EQL program is determined by the values of its variables at that instant. As the result of a rule firing the values of some variables will change, leading the program to a new state. The *state space graph* (or *state transition graph*) of an EQL program is an irreflexive digraph $G = (S, E)$, where $E \subseteq S \times S$ and denotes state transitions. Let $F(s)$ be the set of firable rules of a program in state s. And let $A_r(s)$ be the new state reached by firing rule r in state s. There is an edge (s_i, s_j) in G iff there exists a rule r in the program such that $r \in F(s_i)$ and $A_r(s_i) = s_j$.

From the viewpoint of state space representation, a program execution can be mapped to a (possibly infinite) path in G. The starting vertex of the execution path is the initial state; while the ending vertex, if any, is a *fixed point*, which is a vertex in G whose out-degree is 0. The length of an execution path equals the number of rule firings of the execution. In the following we shall use ϕ to denote a firing, which has two components, $\phi.s$, the state in which the firing takes place, and $\phi.f$, the fired rule. We are now ready to formally define a program execution.

Given an EQL program $\mathcal{P}$ (and thus its state space S) and an initial state $s_0 \in S$, we say a firing sequence $\Phi = \langle \phi_1, \phi_2, \ldots, \phi_n \rangle$, $n \geq 0$, is an *execution* of $\mathcal{P}$ iff

- $n = 0$ and $F(s_0) = \emptyset$, or
- $n > 0$ and
  1. $\phi_1.s = s_0$, and
  2. $\phi_i.f \in F(\phi_i.s)$, $1 \leq i \leq n$, and
  3. $\phi_{i+1}.s = A_{\phi_i.f}(\phi_i.s)$, $1 \leq i < n$, and
  4. if n is finite then $A_{\phi_n.f}(\phi_n.s)$ is a fixed point.

We call n the *length* of the execution.

3 Response-time analysis using rule priority

We define the *response time* of an EQL program to be the length of the longest execution of the program. The goal of our response-time analysis is to determine whether the response time of an EQL program is bounded, and if it is, to find a reasonably tight bound. In [11], we introduced two rule relations, potential-trigger and suppression, which are useful in response-time bound derivation. We also presented an algorithm to compute a bound on program's response-time based on the two rule relations. The analysis in [11] does not make any assumption about how non-determinism is resolved in an execution, i.e., priority information is ignored. In the following we extend our previous results to incorporate rule-priority information. Proofs of some of the lemmas have been omitted for brevity.

Given a program execution $\langle \phi_1, \phi_2, ..., \phi_n \rangle$, we say ϕ_k is *triggered* by ϕ_i, (or, equivalently, ϕ_i is a *trigger* of ϕ_k,) $i < k$, denoted by $\phi_i \rightsquigarrow \phi_k$, iff

1. $\phi_k.f \notin F(\phi_i.s) \vee \phi_k.f = \phi_i.f$, and
2. $\phi_k.f \in F(\phi_j.s) \wedge \phi_k.f \neq \phi_j.f, i < j < k$.

The trigger relation captures the causal relation among rule firings and is closely related to the response time of a program execution. This is manifested in the following theorems.

Lemma 1 *For each firing of rule a at most one firing of rule b can be triggered by the firing of a.*

Lemma 2 *The trigger relation on the firings of an execution is a forest.*

Lemma 3 *If a rule is initially firable, it may be fired for the first time without a trigger. Other than that, every rule firing must have a trigger.*

Motivated by Lemma 2, we have the following definitions.

The *weight* of a firing ϕ_i in an execution Φ, denoted by $\omega_\Phi(\phi_i)$, is defined as

$$\omega_\Phi(\phi_i) \triangleq |\{\phi_j \ : \ \phi_i \rightsquigarrow^* \phi_j\}| \tag{1}$$

where $\rightsquigarrow^*$ is the reflexive and transitive closure of $\rightsquigarrow$. The reader can verify that

$$\omega_\Phi(\phi_i) = 1 + \sum_j \omega_\Phi(\phi_j), \ \phi_i \rightsquigarrow \phi_j \tag{2}$$

The *weight* of a rule r in an execution Φ, denoted by $\omega_\Phi(r)$, is defined as

$$\omega_\Phi(r) \triangleq \begin{cases} 0 & \text{if } r \text{ is never fired in } \Phi \\ \omega_\Phi(\phi) & \phi \text{ is } r\text{'s first firing in } \Phi \end{cases} \tag{3}$$

Theorem 1 *The length of an EQL program execution equals the sum of the weights of the rules whose first firings are not triggered firings.*

Proof. By Lemma 2 we know the length of an execution equals the sum of the sizes of the trees in the trigger relation on the firings. The size of each tree, by definition, is the weight of its root, which is a non-triggered firing. By Lemma 3 we know that each non-triggered firing must be the first firing of some rule. Therefore the weight of a non-triggered firing of some rule equals, by definition, the weight of the rule. □

Let $\omega(r)$ be the largest weight of rule r over all possible executions of the EQL program containing r. A *weight bound* of r, denoted by $\Omega(r)$ is an integer satisfying $\Omega(r) \geq \omega(r)$.

Strictly speaking, rules which may indirectly trigger infinite firings do not have a weight bound by this definition. However, once we determine that the response time of a program is unbounded, it is no longer important to compute the value of weight bounds. To simplify the reasoning, we shall treat ∞ as an "integer" so that every rule has a weight bound.

Lemma 4 *The weight of any firing of a rule r in any execution is less than or equal to $\Omega(r)$.*

Given an EQL program, we say rule a is a *potential trigger* of rule b, denoted by $a \rightsquigarrow b$, iff there exists an execution of the program in which some firing of b is triggered by a firing of a. The *potential-trigger graph* (PT graph) of an EQL program is the digraph representation of the potential-trigger relation on the program rules.

Theorem 2 (spawn principle) *If rules b_1, b_2, ..., b_k are the only rules that can be potentially triggered by rule a, then*

$$\omega(a) \leq 1 + \sum_{i=1}^{k} \Omega(b_i) \tag{4}$$

Proof. By Lemma 1, for any firing ϕ of a at most one firing from each of $b_1, b_2, ..., b_k$ can be triggered. It follows from Lemma 4 that the weights of these triggered firings, if any, are bounded by $\Omega(b_1), \Omega(b_2), ..., \Omega(b_k)$ respectively. Since $\omega(\phi) = 1 + \sum_j \omega(\phi_j), \phi_i \rightsquigarrow \phi_j$, it follows that $1 + \sum_{i=1}^{k} \Omega(b_i)$ is a weight bound of a. □

The spawn principle derives a weight bound for a rule r by assuming that for each firing of r all the children of r in the PT graph will be triggered and then fired before the next firing of r. However, this is not always true. In some cases the children have mutually exclusive enabling conditions, thus at most one of them can be enabled (and triggered). In some other cases firing one rule among the children disables the other rules, thus at most one among the triggered children will be fired. Based on these observations we usher in another important rule relation.

Rule a *suppresses* rule b in state s if

1. $a \notin F(s)$, or
2. $b \notin F(s)$, or
3. $b \notin F(A_a(s))$, or
4. b has a higher priority than a.

Rule a *suppresses* rule b iff Rule a *suppresses* rule b in every state. The *suppression graph* of an EQL program is the digraph representation of the suppression relation on the program rules. We call a set of rules a *suppression group* if every rule in the set suppresses every other rule in the set.

The statement that a suppresses b if b has a higher priority than a may first seem odd. However, it will become clear later that the purpose of suppression is for us to make the assertion, once we know a suppresses b, that "if a and b are both firable in some state *and a is fired first*, then b cannot be firable immediately after a's firing." This assertion holds when b has a higher priority than a, because a can never be fired first if both rules are firable.

Lemma 5 *In an EQL program execution, if rules $b_1, b_2, ..., b_k$ form a suppression group, and they are firable in some state, then at most one of them can stay firable in every following state before it eventually is fired.*

Proof. Let $\langle \phi_1, \phi_2, \ ... \ \rangle$ be a program execution. Suppose b_i and b_j, $i \neq j$, $1 \leq i \leq k$, $1 \leq j \leq k$, are both firable in some state $\phi_l.s$, and stay firable in every following state before they are fired in $\phi_m.s$ and $\phi_n.s$, respectively. Without losing generality we assume $m < n$. This implies $b_j \in F(\phi_m.s)$ and $b_j \in F(\phi_{m+1}.s)$. Furthermore, since both b_i and b_j are firable in $\phi_m.s$ and b_i is fired in $\phi_m.s$, b_j cannot have a higher priority than b_i. However, since $b_i \oslash b_j$, it follows that either $b_i \notin F(\phi_m.s)$, or $b_j \notin F(\phi_m.s)$, or $b_j \notin F(\phi_{m+1}.s)$, or b_j has a higher priority than b_i. Any of these contradicts to the above reasoning. □

Lemma 6 *If rule a is a potential trigger of every rule in a suppression group, then for each firing ϕ of a at most one firing in the group can be triggered by ϕ.*

Proof. Let $\langle \phi_1, \phi_2, ... \rangle$ be a program execution. For a firing ϕ_k to be triggered by a firing of a, say, ϕ_i, $\phi_k.f$ must stay firable in every $\phi_j.s$, $i+1 \leq j \leq k$. By Lemma 5 we know at most one rule in a suppression group can satisfy this condition. □

Theorem 3 (select principle) *If rules b_1, b_2, ..., b_k form a suppression group, and are the only rules that can be potentially triggered by rule a, then*

$$\omega(a) \leq 1 + \max_{1 \leq i \leq k}(\Omega(b_i)) \tag{5}$$

Proof. By definition and Lemma 6, for any firing ϕ of a at most one firing can be triggered. The triggered firing, if any, is a firing among $b_1, b_2, ..., b_k$. By Lemma 4 the weight of the triggered firing is bounded by $\max_{1 \leq i \leq k}(\Omega(b_i))$. It follows that the weight of ϕ is bounded by $1 + \max_{1 \leq i \leq k}(\Omega(b_i))$. □

The spawn principle and the select principle can be combined to derive tighter bounds, as depicted below. To get a succinct representation, for a rule set $\pi = \{r_1, ..., r_k\}$ we shall use $\Omega(\pi)$ to denote $\max_{1 \leq i \leq k}(\Omega(r_i))$.

Theorem 4 (hybrid principle) *If b_1, b_2, ..., b_k are the only rules that can potentially be triggered by rule a, and they can be partitioned into $\{\pi_1, \pi_2, ..., \pi_l\}, l \leq k$, such that every partition element is a suppression group, then*

$$\omega(a) \leq 1 + \sum_{i=1}^{l} \Omega(\pi_i) \tag{6}$$

Proof. By definition and Lemma 6, for any firing ϕ of a at most one firing from each of $\pi_1, \pi_2, ..., \pi_l$ can be triggered. It follows from Lemma 4 that the weights of these triggered firings, if any, are bounded by $\Omega(\pi_1), \Omega(\pi_2), ..., \Omega(\pi_k)$ respectively. Since $\omega(\phi) = 1 + \sum_j \omega(\phi_j), \phi_i \rightsquigarrow \phi_j$, it follows that $1 + \sum_{i=1}^{l} \Omega(\pi_i)$ is a weight bound of a. □

4 Further improvement

Although identifying suppression relation from rule priorities helps in deriving tighter response-time bounds, the bound-derivation principles above do not fully exploit the information carried by rule priorities. For example, consider a two-rule program where rule a is a potential trigger of rule b and a has a higher priority. By the spawn principle we know that the weight bounds of a and b respectively 2 and 1. Therefore we know 3 is a response-time bound of the program. However, for the program to have an execution of length 3, both rules must be firable initially, and then be fired in the sequence $\langle b, a, b \rangle$. Such sequence is impossible, given that a has a higher priority than b. It is easy to see that the longest possible firing sequence is $\langle a, b \rangle$. Therefore the program's response-time bound can be reduced to 2. Although b suppresses a by the result of the previous section, we cannot apply the select principle (to hopefully derive this tighter bound), since a does not suppress b. To fully make use of priority information, an expensive search process is in general necessary. However, substantial improvement at little cost can result by exploiting the following lemma.

In the following, we use $Pr(r)$ to denote the priority of rule r.

Lemma 7 *Let a and b be distinct rules of an EQL program P. For each firing ϕ_{i_0} of a in an execution Φ of P, there exists at most one list of firings ϕ_{i_1}, ϕ_{i_2}, ..., ϕ_{i_m}, $m > 0$, satisfying*

1. $\forall k : 0 \leq k < m : \phi_{i_k} \rightsquigarrow \phi_{i_{k+1}}$, *and*
2. $\forall k : 0 < k < m : Pr(\phi_{i_k}.f) > Pr(b)$, *and*
3. $\phi_{i_m}.f = b$.

Proof. Let Φ be $\langle \phi_1, \phi_2, \ldots \rangle$. Suppose two lists of firings in Φ, ϕ_{i_1}, ϕ_{i_2}, ..., ϕ_{i_m}, $m > 0$, and ϕ_{j_1}, ϕ_{j_2}, ..., ϕ_{j_n}, $n > 0$, both satisfy the above conditions. (Let $\phi_{j_0} = \phi_{i_0}$.) If $i_m = j_n$, by Lemma 2 we get $i_{m-1} = j_{n-1}$. Repeat this argument and we can show that the two firing lists must be identical. If $i_m \neq j_n$, without loss of generality, let $i_m < j_n$. By condition 1 and definition we have $i_k < i_{k+1}$ for $0 \leq k < m$, and $j_k < j_{k+1}$ for $0 \leq k < n$. By condition 2 we have $\phi_{i_k}.f \neq b$ for $0 < k < m$, and $\phi_{j_k}.f \neq b$ for $0 < k < n$. Now consider $\phi_{j_{n-1}}$. We have the following possibilities:

1. $j_{n-1} > i_m$. By condition 1 and the definition of trigger, for every state s from $\phi_{j_0+1}.s$ to $\Phi_{j_{n-1}}.s$, there exists a k, $0 < k < n$, such that $\phi_{j_k}.f \in F(s)$. From this and by condition 2, b cannot be fired in any state from $\phi_{j_0+1}.s$ to $\Phi_{j_{n-1}}.s$. So the firing ϕ_{i_m} of b must occur after $\phi_{j_{n-1}}$. In other words, $j_{n-1} < i_m$. This is contradictory.
2. $j_{n-1} = i_m$. This means $\phi_{j_{n-1}}.f = b$. However, from earlier reasoning we know it is impossible.
3. $i_{m-1} < j_{n-1} < i_m$. Since $\phi_{i_{m-1}} \rightsquigarrow \phi_{i_m}$, by definition $b \in F(\phi_{i_k}.s)$ for $i_{m-1} < k < i_m$. However, since $\phi_{j_{n-1}} \rightsquigarrow \phi_{j_n}$ and $\phi_{j_{n-1}}.f \neq b$, by definition $b \notin F(\phi_{j_{n-1}}.s)$. This is contradictory.
4. $j_{n-1} = i_{m-1}$. This means $\phi_{j_{n-1}}$ triggers both ϕ_{i_m} and ϕ_{j_n}, firings of b. This contradicts with Lemma 1.
5. $j_{n-1} < i_{m-1} (< j_n)$. With analogous argument as used in case 3, we can get a contradiction. □

4.1 A bound-derivation algorithm

Inspired by the previous lemma we present a bound-derivation algorithm, which enhances the one in [WM92].

Algorithm 1 *Compute a response-time bound for an EQL program P whose PT graph is acyclic. Each rule in P has a static priority.*

Input. PT graph $G = (V, E)$ and rule priorities of an EQL program P.

Output. A response-time bound of P if G is acyclic.

Method.

1. Let v_0 be a new vertex. Create $G' = (V', E')$, where $V' = V \cup \{v_0\}$, and $E' = E \cup \{(v_0, v) : v \in V\}$.
2. Create an integer array wb of size $|V'|$, one entry for each vertex in V'.
3. Topologically sort the vertices in G'. If G' is cyclic, (then so is G,) stop.
4. Pick one vertex a at a time from V', in reverse topological order.
5. $wb[a] :=$ GetWeightBound(a, wb, G')
6. If $a \neq v_0$, go to step 4.
7. Output $wb[v_0] - 1$. Stop.

integer function GetWeightBound(a, wb, G')

51. Initialize a local variable wba to 1.
52. Let $T = T_a = \{r : (a, r) \in E'\}$.
53. If $T = \emptyset$ then return wba.
54. Pick a rule p with the highest priority from T.
55. $wba := wba + wb[p]$.
56. Let $T_p = \{q : q \in T \wedge p \succ_q^* q\}$, where $q_1 \succ_q q_2$ iff $Pr(q_1) > Pr(q) \wedge (q_1, q_2) \in E'$. Remove T_p from T.
57. Go to step 53.

Intuitively, T_p contains those rules q such that there exists a path from p to q in the PT graph and all the rules preceding q along the path have a higher priority than q. The process of finding T_p in step 56 can be implemented by performing a depth-first search from p and keeping the minimum priority $minPr$ along the current path. If a node q with a priority less than $minPr$ is encountered, then add q to T_p and set $minPr$ to $Pr(q)$. Initially $minPr$ can be set to $Pr(p)+1$ so that p will be added to T_p.

The correctness proof of the above algorithm is rather lengthy. To facilitate the proof, we build a bridge between program response time and the algorithm's return value. We first define the concept of *relaxed execution* of EQL programs. Then we show that 1. every program execution has a corresponding relaxed execution with the same execution length, and 2. the "relaxed response time" of a program is bounded by the return value of the algorithm.

Let P be an EQL program with rules $R = \{r_1, r_2, ..., r_m\}$. Let Pr be the priority function of the rules in P. A *relaxed execution* of P is a sequence $\langle \varphi_1, \varphi_2, ..., \varphi_n \rangle$, such that each φ_i, $1 \le i \le n$, has two components, $\varphi_i.F$ and $\varphi_i.f$, satisfying the following conditions.

1. $\varphi_i.F \subseteq R$
2. $\varphi_i.f \in \varphi_i.F$
3. $\forall r : r \in \varphi_i.F : Pr(\varphi_i.f) \ge Pr(r)$

In the relaxed execution, we say $\varphi_i.f$ is the i^{th} fired rule, $\varphi_i.F$ the set of firable rules before the i^{th} firing, and n the *length* of the relaxed execution. The concept of trigger in a relaxed execution can then be defined similarly.

Given a relaxed execution $\Phi = \langle \varphi_1, \varphi_2, ..., \varphi_n \rangle$ of an EQL program P, φ_j is *triggered* by φ_i, (equivalently, φ_i is a *trigger* of φ_j,) $i < j$, denoted by $\varphi_i \rightsquigarrow \varphi_j$, iff

1. $\forall k : i < k < j : \varphi_j.f \in \varphi_k.F \wedge \varphi_j.f \neq \varphi_k.f$, and
2. $\varphi_j.f \notin \varphi_i.F \vee \varphi_j.f = \varphi_i.f$

We say φ_j is a *triggered firing* in Φ.

Lemma 8 *Any two triggered firings of the same rule in a relaxed execution of an EQL program must be triggered by different firings.*

Proof. Let $\Phi = \langle \varphi_1, \varphi_2, \ldots, \varphi_n \rangle$ be a relaxed execution. Suppose that two firings of rule b, φ_k, φ_l, are both triggered by firing φ_i in Φ. Without losing generality, we assume $k < l$. Since $\varphi_i \rightsquigarrow \varphi_l$, by definition, $b \neq \varphi_j.f$, for $i < j < l$. And since $\varphi_i \rightsquigarrow \varphi_k$, we must have $b = \varphi_k.f$ and $i < k < l$. However, this is a contradiction. □

Corollary 1 *For each firing φ in a relaxed execution of an EQL program, for each rule at most one firing of the rule can be triggered by φ.*

Lemma 9 *A triggered firing in a relaxed execution has exactly one trigger.*

Proof. By definition we know that a triggered firing has at least one trigger. To show a triggered firing has at most one trigger, let $\langle \varphi_1, \varphi_2, ..., \varphi_n \rangle$ be a relaxed execution. Suppose two firings $\varphi_i, \varphi_j, i \neq j$, are both triggers of a firing φ_k. Without losing generality we assume $i < j$. Thus $i < j < k$. And since $\varphi_i \rightsquigarrow \varphi_k$, we must have $\varphi_k.f \in \varphi_j.F \wedge \varphi_k.f \neq \varphi_j.f$. However, since $\varphi_j \rightsquigarrow \varphi_k$, we must have $\varphi_k.f \notin \varphi_j.F \vee \varphi_k.f = \varphi_j.f$. This is a contradiction. □

Lemma 10 *The trigger relation on the firings of a relaxed execution of an EQL program is a forest.*

Proof. By definition and Lemma 9 we know that every firing in a relaxed execution has at most one trigger. And since a firing can only be triggered by an earlier firing, there cannot be a cycle in the trigger relation. □

Lemma 11 *In a relaxed execution of an EQL program, for each rule, only its first firing may have no trigger.*

Proof. For each φ_k of r in a relaxed execution $\langle \varphi_1, \varphi_2, ..., \varphi_n \rangle$. Let i be the smallest number such that $r \in \varphi_j.F \wedge r \neq \varphi_j.f$, $i < j < k$. If $i \ge 1$ then φ_i is a trigger of φ_k. If $i = 0$ then φ_k is the first firing of r. □

The *weight* of a firing φ_i in a relaxed execution Φ, denoted by $\omega_\Phi(\varphi_i)$, is the size of the subtree whose root is φ_i in the trigger relation on the firings. Formally,

$$\omega_\Phi(\varphi_i) \triangleq |\{\varphi_j \ : \ \varphi_i \rightsquigarrow^* \varphi_j\}| \tag{7}$$

where $\rightsquigarrow^*$ is the reflexive and transitive closure of $\rightsquigarrow$. The reader can verify that

$$\omega_\Phi(\varphi_i) = 1 + \sum_j \omega_\Phi(\varphi_j), \ \varphi_i \rightsquigarrow \varphi_j \tag{8}$$

The *weight* of a rule r in a relaxed execution Φ, denoted by $\omega_\Phi(r)$, is defined as

$$\omega_\Phi(r) \triangleq \begin{cases} 0 & \text{if } r \text{ is never fired in } \Phi \\ \omega_\Phi(\varphi) & \varphi \text{ is } r\text{'s first firing in } \Phi \end{cases} \tag{9}$$

Lemma 12 *The length of a relaxed execution of an EQL program equals the sum of the weights of the rules whose first firings are not triggered firings.*

Proof. By Lemma 10 we know the length of an execution equals the sum of the sizes of the trees in the trigger relation on the firings. The size of each tree, by definition, is the weight of its root, which is a non-triggered firing. By Lemma 11 we know that each non-triggered firing must be the first firing of some rule. Therefore the weight of a non-triggered firing of some rule equals, by definition, the weight of the rule. □

Let P be an EQL program and $G = (V, E)$ be a digraph. A relaxed execution $\Phi = \langle \varphi_1, \varphi_2, ..., \varphi_n \rangle$ of P *PT-conforms to* G iff

$$\forall i :: \forall j :: \varphi_i \rightsquigarrow \varphi_j \rightarrow (\varphi_i.f, \varphi_j.f) \in E.$$

Let $G = (V, E)$ be a digraph. A *relaxed weight bound* of a rule in P PT-conforming to G is an upper bound on the weight of the rule in every relaxed execution of P PT-conforming to G. A *relaxed response-time bound* of P PT-conforming to G is an upper bound on the length of every relaxed execution of P PT-conforming to G.

Lemma 13 *For every execution Φ of an EQL program P, there exists a relaxed execution Φ' of P with the same rule firing sequence such that Φ' PT-conforms to the PT graph of P.*

Proof. For any execution $\Phi = \langle \phi_1,\ \phi_2,\ ...,\ \phi_n \rangle$ of P, the sequence $\Phi' = \langle \varphi_1,\ \varphi_2,\ ...,\ \varphi_n \rangle$, where $\varphi_i.f = \phi_i.f$ and $\varphi_i.F = F(\phi_i.s)$, $1 \leq i \leq n$, is a relaxed execution of P. Both Φ and Φ' have the same firing sequence.

To show that Φ' PT-conforms to PT, notice that for every i and j,

$$\begin{array}{ll} & \varphi_i \rightsquigarrow \varphi_j \\ \Rightarrow & (\varphi_j.f \notin \varphi_i.F \vee \varphi_j.f = \varphi_i.f) \wedge \\ & (\forall k : i < k < j : \varphi_j.f \in \varphi_k.F \wedge \varphi_j.f \neq \varphi_k.f) \\ \Rightarrow & (\phi_j.f \notin F(\phi_i.s) \vee \phi_j.f = \phi_i.f) \wedge \\ & (\forall k : i < k < j : \phi_j.f \in F(\phi_k.s) \wedge \phi_j.f \neq \phi_k.f) \\ \Rightarrow & \phi_i \rightsquigarrow \phi_j \\ \Rightarrow & (\phi_i.f, \phi_j.f) \in PT \end{array}$$

□

Corollary 2 *Let G be the PT graph of an EQL program P. A relaxed response-time bound of P PT-conforming to G is a response-time bound of P.*

We have shown that program execution and relaxed execution have very similar properties as far as the trigger relation is concerned. Yet unlike a program execution, which is defined in terms of program states, a relaxed execution is defined in terms of rules and their priorities only. This property allows us to easily manipulate a relaxed execution in the following proofs, without much concern about the underlying state-transition graph.

Lemma 14 *Given a relaxed execution Φ of an EQL program P PT-conforming to a graph G, for any firing φ_i in Φ, there exists a relaxed execution $\Phi'(= \langle \varphi'_1, \varphi'_2, ... \rangle)$ of P PT-conforming to G such that $\varphi_i.f = \varphi'_1.f$ and $\omega_{\Phi'}(\varphi'_1) = |\Phi'| = \omega_\Phi(\varphi_i)$.*

Proof. We prove the lemma by actually constructing such Φ'. We shall call this procedure Δ *transformation on Φ with respect to φ_i*. Let $\Phi = \langle \varphi_1,\ \varphi_2,\ ... \rangle$. Let $\Phi^*_i = \{\varphi : \varphi_i \rightsquigarrow^* \varphi\}$. We construct Φ' by first copying the firings in Φ^*_i from Φ while retaining their relative order in Φ. Note this defines a bijective mapping from Φ^*_i to Φ' and thus ensures $|\Phi'| = |\Phi^*_i| = \omega_\Phi(\varphi_i)$. For each $\varphi_k \in \Phi^*_i$, let $\varphi'_{\bar{k}}$ be its corresponding firing in Φ'. We then modify the set of firable rules of each firing in Φ' as follows.

1. Initially set all sets empty.

2. Set $\varphi'_1.F$ to $\{\varphi_i.f\}$.

3. For each $\varphi_k \in \Phi^*_i, k \neq i$, let $\varphi_j (\in \Phi^*_i) \rightsquigarrow \varphi_k$. Add $\varphi'_{\bar{k}}.f$ to $\varphi'_l.F, \bar{j} < l \leq \bar{k}$.

(end of transformation)

Note that with this transformation, $\varphi'_1.f = \varphi_i.f$, and $\forall k : \varphi_k \in \Phi^*_i : \varphi'_{\bar{k}}.f = \varphi_k.f \wedge \varphi'_{\bar{k}}.F \subseteq \varphi_k.F$. In the following, for φ'_k in Φ', we use $\varphi_{\hat{k}}$ to denote its corresponding firing in Φ^*_i. We then show that

1. Φ' is a relaxed execution of P. By step 2 we have $\varphi'_1.f \in \varphi'_1.F$ and, since $\varphi'_1.F$ contains only $\varphi'_1.f$, $\varphi'_1.f$ has the highest priority in $\varphi'_1.F$. For each φ'_k in $\Phi', k \neq 1$, by step 3 we have $\varphi'_k.f \in \varphi'_k.F$. Since $\varphi'_k.f = \varphi_{\hat{k}}.f$, which has the highest priority in $\varphi_{\hat{k}}.F$ and $\varphi'_k.F \subseteq \varphi_{\hat{k}}.F$, $\varphi'_k.f$ also has the highest priority in $\varphi'_k.F$.

2. For every φ_j and φ_k in Φ^*_i, $\varphi_j \rightsquigarrow \varphi_k \Rightarrow \varphi'_{\bar{j}} \rightsquigarrow \varphi'_{\bar{k}}$. Since $\varphi_j \rightsquigarrow \varphi_k$, by step 3 we have $\varphi'_{\bar{k}}.f \in \varphi'_l.F, \bar{j} < l < \bar{k}$. And since $\varphi_k.f \neq \varphi_l.f, j < l < k$, we have $\varphi'_{\bar{k}}.f \neq \varphi'_l.f, \bar{j} < l < \bar{k}$. Finally, if $\varphi_k.f = \varphi_j.f$ then $\varphi'_{\bar{k}}.f = \varphi_k.f = \varphi_j.f = \varphi'_{\bar{j}}.f$. Otherwise $\varphi_k.f \notin \varphi_j.F$, which implies $\varphi'_{\bar{k}}.f \notin \varphi'_{\bar{j}}.F$ (because $\varphi'_{\bar{k}}.f = \varphi_k.f$ and $\varphi'_{\bar{j}}.F \subseteq \varphi_j.F$).

3. For every φ'_j and φ'_k in Φ', $\varphi'_j \rightsquigarrow \varphi'_k \Rightarrow \varphi_{\hat{j}} \rightsquigarrow \varphi_{\hat{k}}$. For every $\varphi'_k \in \Phi', k > 1$, it has at most one trigger, and its corresponding firing in Φ^*_i, $\varphi_{\hat{k}}$, has exactly one trigger, which is also in Φ^*_i. Thus there exists a φ'_j in Φ', such that $\varphi_{\hat{j}} \rightsquigarrow \varphi_{\hat{k}}$. From result 2 above we know $\varphi'_j \rightsquigarrow \varphi'_k$.

4. Φ' PT-conforms to G. This follows from result 3 and the fact that Φ PT-conforms to G.

5. $\forall \varphi'_k : \varphi'_k \in \Phi' : \varphi'_1 \leadsto^* \varphi'_k$. Note that $\varphi_{\hat{k}} \in \Phi^*_i$, $\forall \varphi : \varphi \in \Phi^*_i : \varphi_i \leadsto^* \varphi$, and $\varphi'_{\hat{i}} = \varphi'_1$. The rest follows from result 2.

6. $\omega_{\Phi'}(\varphi'_1) = |\Phi'|$. This follows from result 5. □

Lemma 15 *Given an EQL program P whose PT graph G is acyclic, let G′ be the augmented graph of G constructed in step 1 of Algorithm 1. The function GetWeightBound(a, wb, G′) returns a relaxed weight bound of rule a PT-conforming to G′.*

Proof. Let $ord(a)$ be the ordinal number of rule a picked in step 4. From steps 4 to 6 we know that by the time GetWeightBound(a, wb, G') is invoked all the $wb[r]$, $(a, r) \in E'$, have been assigned the returned value of GetWeightBound(r, wb, G'). This ensures that the value calculated in step 53 is always defined.

To show that the returned value is a relaxed weight bound of a PT-conforming to G', we proceed by induction on $ord(a)$. When $ord(a) = 1$, there are no out-going edges from a in G'. Thus any firing of a in a relaxed execution PT-conforming to G' cannot trigger any other firing, meaning that the weight of the firing of a is 1. Further, since there are no out-going edges from a in G', the set T in step 52 is empty, causing 1, the initial value of wba, to be returned. Thus the returned value is a relaxed weight bound of a PT-conforming to G',

Assume the assertion holds for $ord(a) \le k$. Next we need to consider the case $ord(a) = k + 1$. Given a relaxed execution $\Phi = \langle \varphi_1, \varphi_2, ... \rangle$ of P PT-conforming to G', the assertion obviously holds if Φ does not have any firing of a. Otherwise, we need to show that for any firing φ_l of a in Φ, GetWeightBound(a, wb, G') $\ge \omega_\Phi(\varphi_l)$. We shall define a transformation Γ, which transforms such Φ into another relaxed execution of P PT-conforming to G'. The proof is complete if we can show that for any φ_l of a in Φ

1. $|\Gamma(\Phi)| \ge \omega_\Phi(\varphi_l)$, and

2. GetWeightBound(a, wb, G') $\ge |\Gamma(\Phi)|$.

We now describe Γ. Let B be the set of rules picked in step 54. Let C be $T_a - B$. Let φ_l be the "heaviest" firing among all the firings of a in Φ. Δ-transform Φ with respect to φ_l into say, Φ'. Let φ'_k denote a firing in Φ'. By Lemma 14, Φ' PT-conforms to G', $\varphi'_1.f = a$, and $\omega_{\Phi'}(\varphi'_1) = |\Phi'| = \omega_\Phi(\varphi_l)$. The purpose of Γ transformation is to derive a relaxed execution PT-conforms to G' which does not have any firings of rules in C being triggered by the firing of a (so that we can hopefully apply the induction hypothesis.)

Find in Φ' a φ'_n satisfying $\varphi'_n.f \in C \wedge \varphi'_1 \leadsto \varphi'_n$. If such φ'_n does not exist then Φ' is the desired $\Gamma(\Phi)$. Otherwise let $D = \{r : r \succ^+_{\varphi'_n.f} \varphi'_n.f \wedge r \neq a\}$. Let φ'_m be the last firing before φ'_n satisfying $\varphi'_m.f \in D$. (Recall that $q_1 \succ_q q_2$ iff $Pr(q_1) > Pr(q) \wedge (q_1, q_2) \in E'$.) Note $\varphi'_1 \leadsto^+ \varphi'_m$. Now we have the following possibilities.

1. φ'_m exists and $(\varphi'_m.f, \varphi'_n.f) \in E'$. Remove $\varphi'_n.f$ from F'_i for $1 < i \le m$. As the result, φ'_n is now triggered by φ'_m, instead of φ'_1. The new Φ' is still a relaxed execution of P PT-conforming to G'.

2. φ'_m exists and $(\varphi'_m.f, \varphi'_n.f) \notin E'$. Since $\varphi'_m.f \succ^+_{\varphi'_n.f} \varphi'_n.f$, there exist $q_1, ..., q_k$, $k \ge 1$, satisfying $\varphi'_m.f \succ_{\varphi'_n.f} q_1 \succ_{\varphi'_n.f} ... \succ_{\varphi'_n.f} q_k = \varphi'_n.f$. Let $m1$ be the smallest $i, m < i \le n$ such that $Pr(q_1) \ge Pr(\varphi'_{m1}.f)$. (Since $Pr(q_1) > Pr(\varphi'_n.f)$, such $m1$ always exists.) Add q_1 to $\varphi'_i.F$ for $m < i < m1$, and insert a new firing φ'_{m0} of q_1 between φ'_{m1-1} and φ'_{m1}, where $\varphi'_{m0}.F = \varphi'_{m1}.F \cup \{q_1\}$. With this modification the original triggering relationships remain unchanged. In addition, $\varphi'_m \leadsto \varphi'_{m0}$. And the new Φ' is still a relaxed execution of P PT-conforming to G'. Now let φ'_{m0} be the new φ'_m. Keep applying this modification for $q_2, ..., q_{k-1}$ and we'll have case 1.

3. φ'_m does not exist. Then there exist $q_1, ..., q_k$, $k \ge 1$, satisfying $q_1 \in B \wedge q_1 \succ_{\varphi'_n.f} ... \succ_{\varphi'_n.f} q_k = \varphi'_n.f$. Let $m1$ be the smallest $i, 1 < i \le n$ such that $Pr(q_1) \ge Pr(\varphi'_{m1}.f)$. Add q_1 to $\varphi'_i.F$ for $1 < i < m1$, and insert a new firing φ'_{m0} of q_1 between φ'_{m1-1} and φ'_{m1}, where $\varphi'_{m0}.F = \varphi'_{m1}.F \cup \{q_1\}$. With this modification the original triggering relationships remain unchanged. In addition, $\varphi'_1 \leadsto \varphi'_{m0}$. And the new Φ' is still a relaxed execution of P PT-conforming to G'. Now let φ'_{m0} be the φ'_m and we have case 2.

By repeating the above modification we'll eventually apply the modification for case 1 and get say Φ'', such that

1. Φ'' is a relaxed execution of P,

2. Φ'' PT-conforms to G',

3. $|\Phi'| \le |\Phi''| = \omega_{\Phi''}(\varphi''_1)$, and

4. Compared with Φ', Φ'' has *one fewer* firing (φ'_n) of rules in C triggered by the first firing, which is a firing of a.

Rename Φ'' to Φ' and repeat this "Φ'-to-Φ''" procedure until there are no firings of rules in C triggered by φ'_1 of a. Then we have $\Gamma(\Phi)$.

It is obvious that $|\Gamma(\Phi)| \geq \omega_\Phi(\varphi_l)$. To show that GetWeightBound(a, wb, G') $\geq |\Gamma(\Phi)|$, note that $|\Gamma(\Phi)| = \omega_{\Gamma(\Phi)}(\varphi'_1)$, and in $\Gamma(\Phi)$ all the firings triggered by φ'_1 are of the rules in B. The rest follows from Corollary 1, Equation 8, and the induction hypothesis. □

Lemma 16 *Given an EQL program P whose PT graph G is acyclic, let G' be the augmented graph of G constructed in step 1 of Algorithm 1. The return value, if any, of Algorithm 1 for an EQL program P is a relaxed response-time bound of P PT-conforming to G'.*

Proof. We can think of the v_0 in G' as an additional rule which is a potential trigger of every rule in P. For any relaxed execution Φ of P, we can construct a relaxed execution Φ' by prefixing Φ with $\varphi_1(\{v_0\}, v_0)$. Thus every non-triggered firing in Φ is triggered by φ_1 of v_0 in Φ'. By Lemma 12 and Equation 8 the length of Φ would be $\omega_{\Phi'}(\varphi_1) - 1$, which equals $\omega_{\Phi'}(v_0) - 1$. By Lemma 15 we know $wb[v_0] \geq \omega_{\Phi'}(v_0)$. □

Theorem 5 *The return value, if any, of Algorithm 1 for an EQL program is a bound on the response time of the program.*

Proof. The theorem follows from Corollary 2 and Lemma 16. □

5 An analysis example

Based on the bound-derivation algorithm in the previous section and the approximation conditions presented in [11], we implemented a response-time analyzer for EQL programs with acyclic PT graphs. To demonstrate the analysis process we present the analysis example of the following contrived EQL program.

```
PROGRAM ex1;
VAR
        v, w, x, y, z   : INTEGER;
INPUTVAR
        p, q    : INTEGER;
RULES
   {P:2} z := w - 3 ! w := 0
         IF  w > 0
[] {P:3} v := p ! x := q ! y := 1
         IF  y = 0 AND p > 0
[] {P:1} w := x + y ! v := q
         IF  v <> q
[] {P:0} z := 0
         IF  z <> 0 OR q < 5
[] {P:2} v := x / 2 ! x := 0
         IF  x > 0
END.
```

When the response-time analyzer is invoked to analyze the program it first determines various (approximated) rule relations:

```
Potential enableness relation:
   1 -> 4
   2 -> 3 5
   3 -> 1
   5 -> 3
Disableness relation:
   1 -> 1
   2 -> 2
   5 -> 5
Mutual exclusion relation:
Conflict relation:
   1 -> 1 3 4
   2 -> 3 5
   3 -> 1 2 5
   4 -> 1
   5 -> 2 3 5
Suppression relation:
   1 -> 1 2
   2 -> 2
   3 -> 1 2 3 5
   4 -> 1 2 3 4 5
   5 -> 2 5
Potential trigger relation:
   1 -> 4
   2 -> 3 5
   3 -> 1
   5 -> 3
```

With the potential triggerand the rule-priority information, the weight bounds for the rules are then computed.

```
w.b. of R4 is 1
w.b. of R1 is 2
w.b. of R3 is 3
w.b. of R5 is 4
 R5 -> 4; Tp=( R5 R3 )
w.b. of R2 is 5

 R2 -> 5; Tp=( R2 R3 R4 R5 )  R1 -> 2;

The program takes at most 7 firings to reach a
 fixed point.
```

The above output shows that, for example, to compute a weight bound for rule 2, in step 54 the analyzer picks rule 5 first between the two rules (rules 3 and 5) potentially triggered by rule 2. (Note that rule 5 has a higher priority than rule 3.) The analyzer then finds that rule 3 can also be included in the set T_p in step 56, and thus returns 5 as a weight bound of rule 2. The above analysis takes only a fraction of a second on a Sparc 10 workstation. For comparison, without using the rule-priority information, the analyzer would report a bound of 18 firings for the program.

6 Conclusion

Previous work in computing the response-time bounds for rule-based programs usually assumes non-deterministic choice when two or more rules are enabled. This assumption may result in overly pessimistic estimation of the response time when programmers deliberately exploit an *a priori* priority structure over the rules in conflict resolution. In this paper, we discuss how to get tighter bounds by taking rule-priority information into account. We have presented a bound-derivation algorithm, followed by its correctness proof. A response-time analyzer has been implemented to analyze programs whose potential-trigger relations satisfy an acyclicity condition which seems to hold for most of the time-critical application programs that we have encountered. An analysis example is given to demonstrate the analysis process. Although we did not prove it in this paper, the input to the algorithm does not have to be the exact PT graph. It can be any supergraph of the PT graph, provided that no cycles are in the graph. This allows us to use approximation conditions in computing the potential-trigger relation, under the chance of getting looser bounds. The presented algorithm, which is essentially based on the spawn principle, does not use the detected suppression relation though. In a future paper we shall present an improved version based on the hybrid principle.

The target rule-based language in this paper, EQL, is designed to capture the subset of expert systems where the set of decision variables is fixed *a priori*. This seems to be the class of rule-based programs that are most commonly found in expert system applications that are truly time-critical, i.e., no more than a few milliseconds of response time is tolerable. The use of more general expert system programs with dynamically varying working memories for such time-critical applications is beyond the state-of-the-art, partially because currently, the application engineer has no tool to determine with any certainty whether "deliberation" is worth the time lost to taking immediate action. In future work, we shall extend the techniques in this paper to rule-based languages with dynamic working memories for the decision variables.

References

[1] Benda, M., "Real-Time Applications of AI in the Aerospace Industry," Presentation at the Fall School on Artificial Intelligence. The Research Institute of Ecole Normal Superieure, France, Sep. 1987.

[2] Cheng, A. M. K., "**EQL**: A Rule-Based Language for Programming Real-Time Decision Systems," Technical Report, Dept. of Computer Sciences, Univ. of Texas at Austin, Mar. 1989.

[3] Cheng, A. M. K., Browne, J. C., Mok, A. K. and Wang, R.-H., "Analysis of Real-Time Rule-Based Systems with Behavioral Constraint Assertions Specified in Estella," *IEEE Trans. Software Engineering*, vol. 19, no. 9, Sep. 1993.

[4] Cheng, A. M. K. and Chen, C.-H., "Efficient Response Time Bound Analysis of Real-Time Rule-Based Systems," in *Proc. IEEE COMPASS '92*, June 1992.

[5] Cheng, A. M. K. and Wang, C.-K., "Fast Static Analysis of Real-Time Rule-Based Systems to Verify Their Fixed Point Convergence," in *Proc. IEEE COMPASS '90*, June 1990.

[6] Garvey, A. and Lesser, V., "A survey of Research in Deliberative Real-Time Artificial Intelligence," Real-Time Systems J., vol. 6, no. 3, May 1994.

[7] Hendler, J. and Agrawala, A., "Mission Critical Planning: AI on the MARUTI Real-Time Operating System," in *Proc. Workshop on Innovative Approaches to Planning, Scheduling and Control*, pp. 77-84, Nov. 1990.

[8] Mok, A. K., "Formal Analysis of Real-Time Equational Rule-Based Systems," in *Proc. the 10th Real-Time Systems Symp.*, Dec. 1989.

[9] Musliner, D. J., Durfee, E. H., and Shin, K. G., "CIRCA: A Cooperative Intelligent Real-Time Control Architecture," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 23, no. 6, pp. 1561-1574, 1993.

[10] Paul, C. J., Acharya, A., Black, B. and Strosnider, J. K., "Reducing Problem-Solving Variance to Improve Predictability," *CACM*, vol. 34, no.8, pp. 81-93, Aug. 1991.

[11] Wang, R.-H., Mok, A. K., "Deriving Response-Time Bounds for Equational Rule-Based Programs," in *Proc. International Computer Symp.*, Taiwan, R.O.C., Dec. 1992.