Managing Conflicts between Rules

H. V. Jagadish
AT&T Labs
jag@research.att.com

Alberto O. Mendelzon
University of Toronto
mendel@db.toronto.edu

Inderpal Singh Mumick
Savera Systems
mumick@savera.com

September 19, 1998

Abstract

Rules are used as a programming paradigm in several application domains, including active databases, planning, expert systems, and billing. For example, active databases have rules that execute upon the occurrence of particular events if specified condition predicates are satisfied. It is often the case that multiple rules are fireable when a particular event occurs. We propose a declarative mechanism to control the interaction and execution of multiple rules. The mechanism is based upon logical meta-rules that can express various types of relationships between rules. The meta-rules allow us to reason statically about the rule behavior. We can determine, in polynomial time, if a rule will never execute, whether two rules can ever be executed together, and whether a rule system is guaranteed to have a unique execution set for all possible rules that become fireable.

In this paper, we illustrate our techniques using rules in an active database. A system based upon the meta-rules and the static analysis presented here has been found to be of value in a billing application at AT&T to control interactions between discount plans.

1 Introduction

Motivation: Rules are commonly used as a programming paradigm in applications where there exists a cause and effect relationship between events and actions taken in response to events. These applications include query optimization, expert systems, active databases, and billing. For example, in a query optimizer, rules are used to specify query transformations whenever the query matches a given pattern. Similarly, in a billing system, rules can be used to select the discount plans to be applied when a usage event occurs. Active databases have rules that execute upon the occurrence of particular events if specified conditions are satisfied. In all these applications, it is often the case that multiple rules are fireable when a particular event occurs. However, on a sequential computer, the action part of only one rule can be executed at a time. Designers of active databases and rule-based systems have had to struggle with this issue, and a wide variety of techniques have been proposed and implemented to control rule execution (the execution of the action part of a rule) in such cases. Several of these techniques are discussed in the related work Section 8. We desire a **declarative** mechanism to control **completely** which of the multiple fireable rules are actually executed, and in what order. By declarative, we mean that the programmer should use a high level language, such as a logical specification to control the interactions. By complete, we mean that the system should be able to reason statically with the specification and determine if *all* interactions between the rules have been resolved, and whether a unique execution set and order can be guaranteed. We present our techniques within the active database framework.

The motivation for a **declarative** approach is as follows. In a typical active database, with sequential execution of fired triggers, once the first trigger to execute out of the fireable set has been selected, its execution can be used to modify the truth of the conditionals in the other (fireable) rules. These conditionals may be placed in the action parts of rules, if a fireable rule is simply executed when its turn comes. Alternatively the conditionals can be made part of the condition predicate, if this predicate is re-evaluated immediately prior to execution. There are active databases that follow both policies. For our purposes in this paper, this choice is immaterial. In any case, as a consequence of changing the truth of a conditional, a lower priority fireable rule may be inhibited by a higher priority rule that became fireable at the same time. Our experience in writing active rules in database programs indicated that this was an error-prone programming artifice. Additional "flag" variables (boolean variables) were introduced, the action parts (and/or condition predicates) for rules became more complex on account of these flag variables, and it was not always easy to determine exactly under what conditions to set or reset the flag during execution of a rule. Users would much prefer a mechanism where one could explicitly state, for instance, that if rules A and B together become fireable, then rule A inhibits rule B. Note that this notion is not captured by rule priorities: that would be a statement to the effect that if rules A and B together become fireable, then rule A executes *before* rule B.

The motivation for wanting to specify behavior **completely** is that most active databases and rule systems, even when they do permit the explicit specification of priorities, allow an "escape hatch" and use system-derived priorities, based upon selected ad hoc heuristics, where the user specification does not suffice. While allowing such an escape hatch is probably good engineering design, we believe that it is important to be able to tell the rule specifier whether the escape hatch is ever going to be used. In many circumstances, programmers may wish to exercise complete control over what happens under any given set of circumstances, and may prefer to obtain a guarantee that a heuristic ordering will never be applied. In other cases programmers may wish to identify the cases where a heuristic ordering is necessary.

Results: The rule specifier is provided a *meta-rule* language to manage rule interactions. We allow four types of meta-rules. *Positive requirement* meta-rules state that if a certain rule is selected for execution, then some other rule must also be selected. *Disabling* meta-rules allow one rule to prevent another from executing. *Preference* meta-rules give a rule priority over another one if only one of a pair can be selected for execution (e.g. due to some disabling meta-rules). Finally, *Scheduling* meta-rules give control over the order in which rules are executed, and are similar to relative priority rules.

The advantage of our meta-rules approach over conventional solutions is that (1) It provides a powerful and expressive mechanism for controlling rule interactions, and (2) It makes this control explicit and formal, in a language that is sufficiently tractable that automated analysis of the meta-rules is possible. We show how to infer all additional meta-rules from a given set and how to perform, at specification time, analyses such as finding "dead" rules that can never fire, or determining whether there will always be a unique set of rules selected for execution from any possible fireable set.

Often, a semantic analysis of a rule system can lead to such insights. In fact there is a large body of literature devoted to such semantic analysis. In this paper, we do not look at the semantics of the rules at all. Instead, we assume that universally applicable results of any semantic analysis are explicitly encoded in the meta-rules. Of course, results specific to particular choices of values for some variables or rule firings cannot be so encoded, and must be taken into account separately after the global analysis.

We begin with a motivating example in Section 2 from a frequent flier application that shares several common features with billing applications we are familiar with. Our broad framework of rules and meta-rules is outlined in Section 3. In Section 4 we develop a sound and complete axiomatization for meta-rules. The axiomatization and the existence of a polynomial-time algorithm to test implication of meta-rules are exploited in Section 6 to determine properties of a given rule system, such as whether it is guaranteed to result in a unique execution of rules no matter which set of rules become fireable. For this deterministic case, Section 7 has an algorithm to determine, given a set of fireable rules, which rules are executed and in what order. Related work is discussed in Section 8, and concluding remarks are presented in Section 9.

2 Motivating Example

The work presented in this paper is motivated by real problems encountered in billing applications at AT&T. To decrease software development (and maintenance) cost, and to enhance flexibility, there is a move to make data management and processing "rule-based" wherever possible. As an attempt is made to capture in rules the involved semantics of discounting plans, rule conflicts and ambiguities are inadvertently created. The work presented here is an attempt to address this real problem. Since we cannot describe here the exact demands of our specific billing application, we have created an example that captures much of the spirit of the problem (think of `FlightDetail` as the analog of a telephone call table, and mileage computation as the analog of charge computation). We use this example throughout the paper.

The rule interaction problem is present in relational as well as in object-oriented databases. The techniques presented in this paper are applicable to both models. However, due to our familiarity with the Ode [AG89] database and its support for active data [GJ91], we present the example below in O++, the programming language for Ode.

EXAMPLE 2.1 Consider an airline with a database recording frequent flier miles for each customer. The database has in it the following classes:

1. A `Customer` class with several attributes including `total_miles`, representing the total frequent flier mileage accumulated by the customer, and `status`, representing the current membership grade for the customer, such as Basic, Silver, Gold, or Platinum.
2. A `FlightDetail` class with one instance per flight per customer, with attributes `cust_id`, `origin`, `destination`, `date`, `class_of_service`, `miles`, among others. The `miles` attribute records the number of frequent-flier miles earned by the customer on account of this flight. The `cust_id` attribute is a pointer to an instance of the `Customer` class.

One frequently applied update to this database is the creation of a new instance of the `FlightDetail` class, recording the fact that a customer took a flight. Associated with each `FlightDetail` object, and fired upon creation of the `FlightDetail` object, are triggers that cause an appropriate value of the `miles` attribute to be computed, and cause the `total_miles` attribute of the associated `Customer` object to be updated. Some of the triggers written for this purpose are described below. The function distance(`origin`, `destination`) is assumed to compute the flight distance between two given points.

[*a*]: `UPON FlightDetail`
        if (`TRUE`)
        `miles` = distance(`origin`, `destination`) ;

Rule [*a*] does the basic miles computation based upon the distance between the pair of cities flown.

[*b*]: `UPON FlightDetail`
        if (`miles` < 500)
        `miles` = 500 ;

At least 500 are awarded for each flight taken.

[*c*]: `UPON FlightDetail`
        if (`class_of_service` == "first")
        `miles` = 2 * `miles` ;

Rule [*c*] awards double miles to first class customers.

[*d*]: `UPON FlightDetail`
        if (`cust_id`→`status` == "premium")
        `miles` = 2 * `miles` ;

Rule [*d*] awards double miles to "premium" customers.

[*e*]: `UPON FlightDetail`
        if (`cust_id`→`total_miles` >= 100,000)
        `miles` = 500 + `miles` ;

Rule [*e*] awards an extra 500 miles to any customers with more than 100,000 miles in their account already.

[*f*]: `UPON FlightDetail`
        if (`destination` = "SFO")
        `miles` = 500 + `miles` ;

Rule $[f]$ awards an extra 500 miles for flights into San Francisco airport as a special promotion.

```
[g]:   UPON FlightDetail
           if (TRUE)
           cust_id→total_miles =
                   cust_id→total_miles + miles ;
```

Rule $[g]$ accumulates into the customer account the total mileage awarded.
□

It is easy to see that these triggers have interactions. For instance, if the trigger [g] were executed before all the other applicable ones, then the result stored for total miles in the `CUSTOMER` class would be wrong. Similarly, [a] must have completed execution before any of the other rules can execute properly.

Some controls on trigger execution, such as those discussed in the preceding paragraph, are required simply to produce a meaningful result. Other controls may be applied to select between different possibilities. For instance, applying [f] before [d], will give premium customers 1000 miles extra, rather than 500 miles extra, if SFO is their destination. Applying [d] before [f] will give even premium customers only 500 extra miles for SFO. Which order is chosen affects the final value for miles earned.

Often, premium customers may have more than 100,000 miles in their account. We may desire that these customers get only the double miles and not the 500 mile bonus in addition. In other words, rule [d] and rule [e] disable each other, with respect to a specific customer. Further, for a premium member with more than 100,000 miles we want to prefer the premium bonus to the 100,000 mile bonus.

Simple solutions, such as firing all rules, and letting transaction semantics serialize execution, will not produce desired results. Even explicit ordering of rule execution is not enough unless de-activation of later rules is allowed in the action part of earlier rules. Expressing the correct checks for such de-activation is non-trivial.

A workable alternative is to merge all these rules into a single large trigger with a complex action part. Such an approach suffers from several drawbacks. First, the action part of the resulting mega-trigger would itself involve condition-checking to manage flow of control. A general purpose programming language, rather than just SQL, may be called for, making the application complex, and losing the advantage of a rule-based system. Also, specifying the flow correctly may be difficult. Second, the design is no longer modular, and is not a direct representation of the ideas in a user's mind, unlike the triggers written above. If a new promotion is to be applied to the frequent flier miles computation, this promotion may not be simply a matter of expressing a new rule, but instead a complex re-programming task. Third, to the extent that the resulting complex program is specified procedurally, we may lose opportunities for optimization that are present with a declarative specification.

One could argue that such a single large procedural trigger is simpler to write and to understand, and it certainly is so from the the perspective of a large system with many triggers – merging multiple triggers into a smaller number of larger triggers makes this large system easier to understand and manage. However, we pay the price in terms of the complexity of the trigger itself. While merging a couple of rules into a larger trigger may sound easy, such a merger is not straightforward when the number of rules becomes large, and in fact is a major source of complexity in real systems today that compute frequent flier miles, telephone discounts and such.

Our approach to the management of these interactions between rules is through the declarative specification of meta-rules. For instance, a meta-rule could state, in the example above, that only one of [d] and [e] should execute when both are applicable to a customer. Our problem then is how to specify and implement meta-rules that select a subset of applicable rules for execution in any context, and further determine in which order the rules in the subset should be executed.

3 Rules and Meta-Rules

An active rule system $\mathcal{S} = \langle V, \mathcal{M} \rangle$ consists of a set of rules, V, and a set of meta-rules, $\mathcal{M}$. We first present our model for active rules, and then introduce a language for specifying meta-rules to control interaction between rules.

3.1 Rules

A rule comprises an event, a condition, and an action, and is associated with a particular object in the database. Each rule is *activated* upon an event occurrence, such as creation of the object. The set of rules associated with an object that can be activated upon a common event is denoted as V. The requirement that the set of rules forming V be associated with a single object is due to our consideration of the ODE active database model. In general, V is any set of rules that can be activated together. In a relational system, V could be the set of active rules defined on a relation that can be activated together.

When a rule is activated, its condition is checked. If the condition of a rule evaluates to true, the rule is said to be *fireable*. A fireable rule may or may not be *executed*, depending upon the meta-rules specified to control execution of the rules.

The set of fireable rules associated with a particular object (or tuple) is denoted as I, or the input set (input for the rule selection algorithm). Semantic analysis may show that certain subsets of V cannot simultaneously be firable. However, we do not look at the semantics, and in general assume that I could be any (improper) subset of V. The subset of I that actually executes is denoted as O, or the execution set (output from the rule selection algorithm). Clearly, $O \subseteq I \subseteq V$. An ordering for all the rules in O is determined, and all these rules are executed in the specified sequence. This execution model is similar to that of [DBB+88, GJ91].

Many other rule systems have an execution model in which exactly one of the the fireable rules is selected for execution. After this rule has executed, the fireable set of rules is re-evaluated. In such systems, the execution set comprises only a single rule from I. We do not consider this execution model further in this paper, for two reasons. First, our work is inspired in the context of the Ode system, and so the execution model we consider is that of Ode, even though this differs from that of some other rule systems. Second, and most important, rule systems that reevaluate the fireable set after every rule execution tend to be appropriate for "closed" environments, where the events of interest are in the same domain as the actions taken; for instance, both could be updates to a database. (Sometimes, there is not even a notion of an event: rules are simply fireable as soon as their condition part is satisfied). In our case, we are keen to permit events of interest that are external, even if this then gets reflected as an update to a database. For instance, the act of a passenger boarding a flight is a real-life event, and there are a number of actions that we wish to take in consequence. It would not be appropriate to take only the first action and then wait for the next event.

EXAMPLE 3.1 Consider the active rules $[a] \cdots [g]$ in the frequent flier database of Example 2.1. All these rules are activated upon the creation of a `FlightDetail` object. Thus $V = \{a, b, c, d, e, f, g\}$.

Rules $[a]$ and $[b]$ are always fireable, since `miles` < 500 is true initially. (Rule $[b]$ may not fire after rule [a] executes, but we do not know this in advance.) Rule $[c]$ is fireable if the flight is taken in first class. Rule $[d]$ is fireable if the flight is taken by a "Premium" customer. Rule $[e]$ is fireable if the flight is taken by a customer with more than $100,000$ total miles. Rule $[f]$ is fireable if the flight ends in San Francisco. Rule $[g]$ is always fireable. Thus, the input set I depends upon the newly created flight detail object. □

Because we analyze rules after the fireable set has been determined, from now on we treat rules as atomic objects with no internal structure. We do not try, for example, to detect mutually exclusive conditions or rules that will never become fireable.

3.2 Controlling Information

Our idea is to use statements in a control language to manage the interaction of rules in V. What should this control language be? It appears to be a bad idea to have a very rich control language, with possible data value dependencies. In such a case, the control meta-rules may themselves begin to look like rules, and may themselves require some meta-control!

A reasonable choice appears to be to use some subset of propositional logic, with two propositional symbols for each rule in the underlying system. For rule R_j, proposition I_j can mean "rule R_j is enabled to fire" and proposition O_j can mean "rule R_j actually fires." Control rules would be statements of the form: "if certain rules are/are not enabled, and certain rules fire/do not fire, then certain other rules must/must not fire."

Such a language would certainly be very expressive. The problem is that such expressive power carries with it high computational complexity: we could easily encode an arbitrary conjunctive normal form propositional formula into a set of statements in this language, showing that satisfiability of sets of control statements, and implication of one such statement from a given set, would both be computationally intractable.

In consequence, we propose as the meta-rule language a subset of propositional logic, that still provides the expressive power to be able to enforce most of the types of rule interactions that users and applications may desire. This statement is based on an empirical study of rule interactions in a major rule-based database application in AT&T, in which we satisfied ourselves that, at least so far as this application was concerned, our set of meta-rules is enough to express *all* the interactions of interest.

It is also easy to show that the addition of small additional functionality, such as disjunctions, into the chosen set of meta-rules makes inference over the meta-rule set intractable.

3.3 Meta-Rules

Let $\mathcal{M}$ be a set of *meta-rules* that control the interaction and execution of the rules in V. These meta-rules are used to (1) select the set O from I, and (2) to order the resulting execution set. The rules in the set $I - O$ are not executed, as a result of the restrictions imposed by the meta-rules.
Meta-rules are of four types, as listed below, where, $A, B, C, D, \ldots \in V$:

Positive requirement meta-rules : $A \supset B$; If rules A and B are in the input set and rule A executes, then rule B must execute as well.

In the airline example, rule [c] requires that rule [a] must also execute. To express this constraint, we write a meta-rule $[c] \supset [a]$.

Disabling rules : $\overline{AB}$; Rules A and B disable each other; only one of them may execute, even if both A and B are in the input set. The meta-rule $\overline{A}$ is a special unary case of this rule and means that the rule A can never be executed. It can be viewed as short hand for $\overline{AA}$.

In the airline example, only one of rules [d] and [e] should be allowed to execute.

Preference rules : $B > A$; If rules A and B are in the input set, and are both fireable, then B should not be dropped from the execution set unless A also is. In other words, if A and B are both fireable, and there is some reason why A and B cannot both be executed, then we prefer to execute B rather than A.

In the airline example, when rules [d] and [e] are both fireable, execution of both together is disallowed by the disabling meta-rule $\overline{de}$. At this point, we prefer to execute rule $[d]$ rather than rule $[e]$, that is, $d > e$.

Scheduling rules : $A \prec B$; If rules A and B are in the input set and are both selected for execution, execute A before B.

In the airline example, when rules $[a]$ and $[b]$ are both executable, we can require the rule $[a]$ execute before $[b]$ by stating $a \prec b$.

To understand the difference between positive requirement and preference rules, consider the airline example.

EXAMPLE 3.2 As mentioned earlier, $d > e$ means that we must choose rule $[d]$ when both rules $[d]$ and $[e]$ are fireable but we have to choose only one. If the fireable set contains only $[e]$, then we want to execute $[e]$. If the fireable set contains only $[d]$, then we want to execute $[d]$. On the other hand, $c \supset a$ means that for rule $[c]$ to execute, rule $[a]$ must execute. If only rule $[c]$ is fireable, then $[c]$ cannot execute. However, if only rule $[a]$ is fireable, then $[a]$ can execute, and if both rules $[a]$ and $[c]$ are fireable, but we must choose one of these, then rule $[a]$ can execute. □

Note that the positive requirement meta-rule is asymmetric while the disabling meta-rule is symmetric. The reason why we chose the asymmetric requirement rule is that one can express a symmetric requirement rule (A and B require each other), using two asymmetric requirement rules ($A \supset B$ and $B \supset A$), while we cannot express the asymmetric requirement rule using the symmetric requirement rules and other rules.

For disabling rules, we can express an asymmetric disabling meta-rule (if A is in the input set, then B cannot fire, but if B is in the input set, A can still fire), if desired, through a symmetric disabling meta-rule ($\overline{AB}$) and a preference meta-rule ($A > B$).

EXAMPLE 3.3 For the airline example, the following meta-rules capture the desired policy to resolve the interactions stated in Section 2.
$b \supset a$: Rule [b] positively requires rule [a].
$c \supset b$, $d \supset b$, $e \supset b$, $f \supset b$, and $g \supset b$: Rules [c], [d], [e], [f], and [g] all positively require rule [b].
$\overline{de}$: Rules [d] and [e] cannot execute together.

$d > e$: If both [d] and [e] are fireable, then prefer [d].

$a \prec b$, $b \prec c$, $b \prec d$, $b \prec e$, $b \prec f$, $b \prec g$, $c \prec f$, $c \prec g$, $d \prec f$, $d \prec g$, $e \prec f$, $e \prec g$, and $f \prec g$: Whenever a listed pair is executed, one must be scheduled before the other. $\square$

The formal semantics of a set $\mathcal{M}$ of meta-rules is defined by considering the models for the set $\mathcal{M}$.

Definition 3.1 (Model): A *model* for a set $\mathcal{M}$ of meta-rules is a quadruple $\langle V, I, O, \omega \rangle$ where V is a set of rules (treated as atomic objects for our purposes), $O \subseteq I \subseteq V$, and ω is a total order on the set O.

We say that $\langle$V,I,O,$\omega\rangle$ *satisfies* meta-rule μ if:

1. When μ is of the form $A \supset B$, then if $A \in O$, then $B \in O$.
2. When μ is of the form $\overline{AB}$, then $\neg(A \in O \;\&\; B \in O)$.
3. When μ is of the form $B > A$, then if $AB \subseteq I \;\&\; A \in O$, then $B \in O$.
4. When μ is of the form $A \prec B$, then if A and B both appear in the execution set O, then A appears before B in the total order ω.

The quadruple $\langle V, I, O, \omega \rangle$ is a model for the set $\mathcal{M}$ of meta-rules if $\langle V, I, O, \omega \rangle$ satisfies every meta-rule $\mu \in \mathcal{M}$. $\square$

EXAMPLE 3.4 For the airline example, one may verify that the quadruple

$$\langle \{a, b, c, d, e, f, g\}, \{a, b, c, d, e, g\}, \{a, b, c, d, g\}, (a \to b \to c \to d \to g) \rangle.$$

is a model. The term $(a \to b \to c \to d \to g)$ specifies the total order ω between the rules in the execution set. $\square$

3.4 Analysis Goals

Given an active rule system $\mathcal{S} = \langle V, \mathcal{M} \rangle$, one would like to determine the following:

- Given a rule $A \in V$, is it possible that A does not execute for any input set? A rule that can never execute due to the meta-rule specification will be called a *dead* rule.[1] What are all such *dead* rules?

[1] A rule may never fire because it has a condition predicate that can never be true, or is triggered by an event that can never occur. Such analysis, of the semantics of the rules themselves, is outside the scope of this paper. When we say *all* dead rules, we mean all rules that are dead on account of the specified set of meta-rules.

- Given a pair of rules $A, B \in V$, can A and B execute together for some input? What are all such A, B pairs?[2]
- Is a non-empty solution O possible for at least one valid non-empty input set I?
- Is the set of meta-rules guaranteed to result in a unique total order ω for every execution set O for every fireable set I?
- Are some of the meta-rules in $\mathcal{M}$ redundant?

We will answer these questions in Section 6. But in order to do so, we need to reason with meta-rules, and understand how to infer new meta-rules from a given set. We give the inference rules in Section 4.

4 Inferring Meta-Rules

Given a set $\mathcal{M}$ of meta-rules, other meta-rules will be logically implied by this set. For instance, if rule A positively requires B and B in turn positively requires C, then any model of these two meta-rules must also satisfy that A positively requires C, even though this requirement may never have been explicitly stated in a meta-rule. Similarly, if A positively requires C, but A and C disable each other, then A can never execute (rule A disables itself).

Inference of meta-rules in the above manner can be done independent of the input set I. We assume in this section that we are given a *rule system*, $\mathcal{S}$, comprising the set V and the meta-rules $\mathcal{M}$. The main result in this section is an inference algorithm guaranteed to infer every interaction relationship derivable on the basis of the given set of rules and meta-rules. The inference algorithm can be used to identify "dead" rules that can never execute, and also forms a necessary basis for the other results presented in the next sections. Our first set of inference rules will ignore the scheduling meta-rules. Later we will bring the scheduling meta-rules back into the analysis and show they are related to the positive requirement, disabling, and the preference rules.

Definition 4.1 (Implication): In a rule system $\mathcal{S}$, comprising the set V and the meta-rules $\mathcal{M}$, we say that $\mathcal{M}$ *implies* a meta-rule μ if every model of $\mathcal{M}$ satisfies μ. □

4.1 Complete Inference Axioms

We present below a set of syntactic inference rules A1–A8 and show that they capture meta-rule implication. The inference rules are of the form $\Gamma \vdash \gamma$, where Γ is a set of meta-rules and γ is a meta-rule. We say that meta-rule μ can be derived from the set of meta-rules $\mathcal{M}$, and denote this by $\mathcal{M} \vdash \mu$, when either μ is in $\mathcal{M}$, or it can be derived from $\mathcal{M}$ by repeated number of applications of axioms A1–A8.

A1 $\overline{AB} \vdash \overline{BA}$.
If A cannot execute with B, then B cannot execute with A.

A2 $(A \supset B \wedge \overline{BC}) \vdash \overline{AC}$.
If A positively requires B, and if B and C cannot execute together, then A and C can never execute together.

A3 $\overline{A} \vdash \overline{AB}$.
If A can never execute, then for every rule B, A and B can never execute together.

A4 $\vdash A \supset A$.
Rule A positively requires itself.

A5 $(A \supset B \wedge B \supset C) \vdash A \supset C$.
If A positively requires B, and if B positively requires C, then A positively requires C.

[2]As in the case of dead rules, we do not analyze whether there actually is a database state and event occurrence for which the condition predicates of the two rules can actually be satisfied simultaneously. We merely capture whether such simultaneous satisfaction is possible given the set of meta-rules. In fact, if the application semantics indicate that two rules can never fire together, it is useful to capture this as a meta-rule.

A6 $\overline{A} \vdash A \supset B$.
If A can never execute, then for every rule B, A positively requires B.

A7 $\vdash A > A$.
If we have to choose between A and A, choose A.

A8 $(A \supset B \wedge C > B) \vdash C > A$.
If A positively requires B, and if C is preferred over B, then C is preferred over A.

The following theorem shows that $\mathcal{M}$ implies μ if and only if μ can be derived from $\mathcal{M}$ using these rules.

Theorem 4.1 *The set of inference rules A1–A8 is sound and complete for positive requirement, disabling, and preference rules.* □

Proof: Soundness is straightforward. For example, consider A8. Suppose $\langle V, I, O\rangle$ satisfies $A \supset B$ and $C > B$ and $AC \subseteq I$; then if $A \subseteq O$, then $B \subseteq O$, and hence $B \subseteq I$ so $BC \subseteq I$ and by $C > B$, $C \in O$, showing $C > A$.

To prove completeness, let $\mathcal{M}$ be a set of meta-rules. Suppose first that there is a meta-rule $\mu = A \supset B$ such that $\mathcal{M} \not\vdash \mu$. Construct a model of $\mathcal{M}$ in which μ does not hold as follows. For each rule C mentioned in $\mathcal{M}$, let C^+ be the set of all rules D such that $\mathcal{M} \vdash C \supset D$ using the axioms A1–A8. Consider the structure $\langle V, I, O\rangle$ in which V is the set of all rules mentioned in $\mathcal{M}$ and $I = O = A^+$. We claim this is a model of $\mathcal{M}$. First note that, since $I = O$, any meta-rule of the form $B > Y$ is satisfied. Now consider any meta-rule $C \supset D$ in $\mathcal{M}$. If $C \notin A^+$, then $C \supset D$ is trivially satisfied by O. If $C \in A^+$, then by axiom A5 $D \in A^+$, so $C \supset D$ holds in O. Now consider a meta-rule $\overline{CD}$ in $\mathcal{M}$. If $\{CD\}$ is not a subset of A^+ clearly O satisfies the meta-rule. Suppose $CD \subseteq A^+$. Then $\mathcal{M} \vdash A \supset C$, which with $\overline{CD}$ yields $\overline{AD}$ by axiom A2. Also $\mathcal{M} \vdash A \supset D$ (since we assumed that $CD \subseteq A^+$), which together with $\overline{AD}$ and axioms A1 and A6 yield $A \supset B$, contradicting $\mathcal{M} \not\vdash A \supset B$.

Now suppose there is a meta-rule $\mu = \overline{AB}$ such that $\mathcal{M} \not\vdash \mu$. Construct a model of $\mathcal{M}$ in which μ does not hold as follows. Let V be as before, and $I = O = A^+ \cup B^+$. By axiom A4, A and B are both in $A^+ \cup B^+$, so μ does not hold in O. Now we have to show that O is a model of $\mathcal{M}$. As above, every meta-rule $C > D$ holds. Consider any meta-rule $\overline{CD}$ in $\mathcal{M}$ and suppose this rule does not hold in O. First, suppose both C and D are in A^+ (the case where they are both in B^+ is symmetric.) As above, we can derive $A \supset C$, which with $\overline{CD}$ and A2 yields $\overline{AD}$. We can also derive $A \supset D$, which with $\overline{AD}$ and axiom A3, yields $\overline{AB}$, a contradiction. So it must be the case that, say, $C \in A^+$ and $D \in B^+$. From $B \supset D$ and $\overline{CD}$ we get $\overline{BC}$ by A2. From $A \supset C$ and $\overline{BC}$ we get $\overline{AB}$ by A2, a contradiction since we assumed $\mathcal{M} \not\vdash \overline{AB}$. Hence all exclusion rules of $\mathcal{M}$ hold in O. Now consider any meta-rule $C \supset D$ in $\mathcal{M}$. Either C is not in O, in which case the meta-rule holds trivially, or D is in O by repeated application of axiom A5.

Finally, suppose there is a meta-rule $\mu = A > B$ such that $\mathcal{M} \not\vdash \mu$. Construct a model with V as above, $I = \{A, B^+\}$, and $O = B^+$. Clearly μ does not hold unless $A \in B^+$; but by axioms A7 and A8, this derives $A > B$, a contradiction. To show that this is a model of $\mathcal{M}$, consider first any meta-rule $C \supset D$ in $\mathcal{M}$. If $C \in B^+$, then $D \in B^+$, so the meta-rule holds. Now consider any meta-rule of the form $\overline{CD}$ in $\mathcal{M}$. The only way this can fail is if $CD \subseteq B^+$. From $B \supset C$ and $\overline{CD}$ we derive $\overline{BD}$ by axiom A2. From $\overline{BD}$ and $B \supset D$, we get $B \supset A$ by axioms A1 and A6, and $A > B$ by axioms A7 and A8, a contradiction. To complete this case, consider any meta-rule $C > D$ in $\mathcal{M}$. Suppose $CD \subseteq \{A, B^+\}$ and $D \subseteq B^+$. If $C \in B^+$, then $C > D$ is satisfied and we are done. If not, then $C = A$. From $A > D$ and $B \supset D$, by axiom A8, we get $A > B$, which we assumed was not derivable from $\mathcal{M}$, a contradiction. □

A consequence of Theorem 4.1 is that we can determine what positive requirement and disabling meta-rules are implied by a set without considering the preference rules, as stated by the following Corollary.

Corollary 4.2 *Let $\hat{\mathcal{M}}$ be the set of rules in $\mathcal{M}$ of the form $A \supset B$ or $\overline{AB}$. Let μ be a rule of one of these forms. Then $\mathcal{M} \vdash \mu$ if and only if $\hat{\mathcal{M}} \vdash \mu$.* □

4.2 Axioms involving scheduling rules

Finally, we bring the scheduling meta-rules back into the picture. It might seem that the scheduling rules are independent of the others, because they only constrain the order in which rules fire after they have been selected by applying the other meta-rules. However, this is not the case; there is a subtle interaction between the scheduling, positive requirement, and the disabling meta-rules, captured by the following three inference axioms. We use the notation $A \supset X$, where A is a rule and $X = \{B, B_1, \ldots, B_k\}$ is a set of rules, to abbreviate $A \supset B$ for every $B \in X$.

A9 $(A \prec B_1 \wedge B_1 \prec B_2 \wedge \ldots \wedge B_k \prec B) \wedge A \supset X \wedge B \supset Y \vdash A \prec B$, where $X \cup Y = \{B_1, \ldots B_k\}$.

A10 $(B_1 \prec B_2 \wedge \ldots \wedge B_k \prec B_1) \wedge A \supset X \wedge B \supset Y \vdash \overline{AB}$, where $X \cup Y = \{B_1, \ldots B_k\}$.

A11 $\overline{AB} \vdash A \prec B$.

Axiom A9 says that $\prec$ is transitive as long as the two ends of a path require all the mid-points to be also present. Axiom A10 says that if there are two rules A and B that together require the execution of a set of rules whose precedence relation is cyclic, then A and B cannot be both selected for execution. Axiom A11 may seem counter-intuitive; but note that $A \prec B$ means "if A and B both appear in the execution sequence, then A appears before B", which is trivially true if A and B can never both appear in the execution (and in fact $B \prec A$ can also be derived by axiom A1).

Theorem 4.3 *The set of inference rules A1–A11 is sound and complete for positive requirement, disabling, preference, and scheduling rules.* □

Proof: (Soundness) Consider A9. Suppose $\langle V, I, O, \omega\rangle$ satisfies $A \prec B_1$, $B_1 \prec B_2$, …, $B_k \prec B$, $A \supset X$, $B \supset Y$, and $X \cup Y = \{B_1, \ldots B_k\}$. Let $\{A, B\} \subset I$. Then, due to the positive requirement meta-rules, $\{B_1, \ldots, B_k\} \subseteq O \subseteq I$. Given the precedence meta-rules, the ordering ω on the execution set O ensures that $A \prec B$. For A10, suppose the premises of the axiom hold in $\langle V, I, O, \omega\rangle$, and suppose A and B are both in O. As for A9, this implies $\{B_1, \ldots, B_k\} \subseteq O \subseteq I$; but the scheduling meta-rules for the B_i's are cyclic and hence cannot be satisfied, a contradiction. We conclude that A and B cannot be both in O, that is, $\overline{AB}$ holds in $\langle V, I, O, \omega\rangle$. Finally, soundness of A11 follows immediately from the explanation in the paragraph before the Theorem.

(Completeness) First we need to extend the proof of Theorem 4.1 to this larger class of meta-rules. For each construction of the partial interpretation $\langle V, I, O\rangle$ in this proof, we define a total order on O as follows. Construct a directed graph with a node for each rule $A \in O$ and an edge from A to B if $\mathcal{M} \vdash A \prec B$. Suppose this graph is cyclic, with a cycle $B_1, \ldots, B_k, B_1$. We show that in each construction in the proof of Theorem 4.1, cyclicity of this graph leads to a contradiction. In the first construction, where $O = A^+$ for some rule $A \supset B$ not implied by $\mathcal{M}$, from $A \supset \{B_1 \ldots B_k\}$ we derive $\overline{A}$ by A10, and hence $A \supset B$, a contradiction. In the second construction in the proof, where $O = A^+ \cup B^+$ for some $\overline{AB}$ not derivable from $\mathcal{M}$, we similarly derive $\overline{AB}$, a contradiction. Finally, in the third construction, where $I = \{A, C\}$ and $O = A^+$ for some $C > A$ not implied by $\mathcal{M}$, we similarly derive $A \supset C$ and hence $C > A$, a contradiction. So, in all three constructions the graph constructed above is acyclic. We can therefore impose on O the total order resulting from any topological sort of this graph, which will make every scheduling meta-rule in $\mathcal{M}$ true. This shows the constructions of the proof of Theorem 4.1 can all be extended to the case where $\mathcal{M}$ contains also scheduling meta-rules.

To finish the proof, we need to show that for any scheduling meta-rule $A \prec B$, if $\mathcal{M} \not\vdash A \prec B$ then there is a model of $\mathcal{M}$ in which $A \prec B$ does not hold. Let $I = O = A^+ \cup B^+$. Consider, as above, the graph with a node for each rule in O and an edge from node C to node D if $\mathcal{M} \vdash C \prec D$. First, the graph must be acyclic; if it were not, then $\mathcal{M}$ would derive $\overline{AB}$ by A10, hence $A \prec B$ by axiom A11, a contradiction. Also, the graph cannot contain a path from A to B, or else $\mathcal{M}$ would derive $A \prec B$

by A9. Therefore, there is a total order obtained by a topological sort of the graph in which B comes before A. This order makes every other scheduling meta-rule in $\mathcal{M}$ true, and $A \prec B$ false.

Finally, all the requirement and preference meta-rules in $\mathcal{M}$, are seen to be satisfied exactly as in the proof of Theorem 4.1. $\Box$

Using these rules we obtain an extension of Corollary 4.2:

Corollary 4.4 *Let $\hat{\mathcal{M}}$ be the set of meta-rules in $\mathcal{M}$ of the form $A \supset B$ or $\overline{AB}$ or $A \prec B$. (That is, all except the preference rules). Let μ be a meta- rule of one of these forms. Then $\mathcal{M} \vdash \mu$ if and only if $\hat{\mathcal{M}} \vdash \mu$.* $\Box$

5 Algorithms to infer meta-rules

A polynomial time algorithm to test whether a meta-rule follows from a given set of meta-rules can be developed using axioms A1-A11.

In fact, we now give an algorithm to compute, given a set of meta-rules $\mathcal{M}$, the set of positive requirement, disabling, and scheduling meta-rules implied by $\mathcal{M}$. We do this by computing, for each rule A, the set $A^+ = \{\text{rules } B \mid \mathcal{M} \vdash (A \supset B)\}$ and the set $A^- = \{\text{rules } B \mid \mathcal{M} \vdash \overline{AB}\}$. We also compute a directed graph $G_{\mathcal{M}}$ whose nodes are the rules in V, and there is an edge (A, B) in $G_{\mathcal{M}}$ if and only if the scheduling rule $A \prec B$ is implied by $\mathcal{M}$.

Algorithm 5.1
INPUT: A set $\mathcal{M}$ of meta-rules on a set of rules V.
OUTPUT: (a)For each $A \in V$, the sets $A^+ = \{\text{rules } B \mid \mathcal{M} \vdash (A \supset B)\}$ and
$A^- = \{\text{rules } B \mid \mathcal{M} \vdash \overline{AB}\}$.
(b)A graph $G_{\mathcal{M}} = (V, E)$ such that $(A, B) \in E$ if and only if $\mathcal{M} \vdash A \prec B$.
METHOD:
1. **for each** $A \in V$
2. $A^p := \{A\} \cup \{C \mid A \supset C\}$;
3.$A^m := \{C \mid \overline{AC} \in \mathcal{M}\} \cup \{C \mid \overline{CA} \in \mathcal{M}\}$
4.**end for**
5.$G_{\mathcal{M}} := (V, E)$ where $E = \{(A, B) \mid A \prec B \in \mathcal{M}\}$.
6.**while** changes occur
7. **if** $C \in A^p$ for some $A, C \in V$ **then** $A^p := A^p \cup C^p$ **end if**
8. **if** $A^p \cap B^m \neq \emptyset$ for some $A, B \in V$
then $A^m := A^m \cup \{B\}$; $B^m := B^m \cup \{A\}$; add edges (A, B) and (B, A) to $G_{\mathcal{M}}$ **end if**
9. **for each pair** $(A, B) \in V$
10. $G_{\mathcal{M}}(A, B) :=$ subgraph of $G_{\mathcal{M}}$ induced by $A^p \cup B^p$
11. **if** $(A, B) \in$ transitive_closure$(G_{\mathcal{M}}(A, B))$ **then** add edge (A, B) to $G_{\mathcal{M}}$ **end if**
12. **for each** $C \in V$ such that $(C, C) \in$ transitive_closure$(G_{\mathcal{M}}(A, B))$
13. $A^m := A^m \cup \{B\}$; add edge (A, B) to $G_{\mathcal{M}}$
14. $B^m := B^m \cup \{A\}$; add edge (B, A) to $G_{\mathcal{M}}$
15. **end for**
16. **end for**
17. **for each** A such that $A^p \cap A^m \neq \emptyset$
18. $A^p := A^m := V$
19. **end for**
20. **end while**
$\diamond$

The main loop of the algorithm repeatedly makes inferences of meta-rules, by adding elements to some A^p or some A^m, or by adding edges to the precedence graph $G_{\mathcal{M}}$. In particular, let us explain the first **for each** loop within the main loop, starting at line 9. First, we construct for each pair of rules A and B a graph $G_{\mathcal{M}}(A, B)$, the subgraph of the precedence graph $G_{\mathcal{M}}$ induced by all the nodes that have been shown to be positively required by A or B. We examine the transitive closure of this graph. If there is an edge from A to B in this transitive closure, it means we can apply axiom A9 to infer $A \prec B$, which we represent by inserting the (A, B) edge in $G_{\mathcal{M}}$. Now, in the nested **for each** loop, we look for rules C in the same transitive closure graph that participate in cycles. For each such rule we find, we can apply axiom A10 to conclude $\overline{AB}$ and hence $\overline{BA}$, which we represent by updating A^m and B^m.

Theorem 5.1 *At the end of the execution of Algorithm 5.1, $A^p = A^+$, $A^m = A^-$, and edge (A, B) is in $G_{\mathcal{M}}$ if and only if $\mathcal{M} \vdash A \prec B$.* $\Box$

Proof: Soundness of the algorithm, that is, $A^p \subseteq A^+$, $A^m \subseteq A^-$, and $\mathcal{M} \vdash A \prec B$ for every edge (A, B) in GM, follows easily from soundness of the inference rules above. If $A^p \cap A^m \neq \emptyset$, then $\overline{A}$ is true, A can never execute, and hence A can trivially disable as well as positively require every other rule in V, that is, $A^+ = A^- = V$.

Completeness is proved by induction on the length of the derivation of the meta-rules. There are three types of meta-rules to consider.

1. Positive requirement rules. Suppose $\mathcal{M} \vdash A \supset C$. We want to show that C is placed in A^p by the algorithm. Proceed by induction on the length of a shortest derivation of $A \supset C$ from $\mathcal{M}$ using the axioms. For the basis, the only derivation of length 1 is one where $A \supset C$ is in $\mathcal{M}$, or one where axiom A4 is used, and in both cases AP is appropriately set by the initialization step. For the induction step, there are two cases to consider, depending on the last axiom used to derive $A \supset C$: A5 or A6. If it is A5, then by induction B will be placed in A^p and C in B^p, so line 7 will eventually place C in A^p. If it is A6, by induction A will be placed in A^m , and we know A is also placed in A^p by the initialization, so lines 17 and 18 will place C in A^p.
2. Disabling rules. Suppose $\mathcal{M} \vdash \overline{AC}$ Proceed by induction as before. For the basis, the initialization will place in A^m any C such that $\overline{AC}$ is in $\mathcal{M}$. For the induction step, there are four axioms to consider as the last step in the derivation of $\overline{AC}$: A1, A2, A3, and A10. Since the algorithm always treats A and C symmetrically whenever it infers $\overline{AC}$ or $\overline{CA}$, A1 is taken care of. Inferences ending in A2 will be performed by line 8. A3 is taken care of by lines 17 and 18. For A10, note that, by induction, each B_i will be placed either in A^p or in C^p. Also by induction, there is an edge in $G_{\mathcal{M}}$ from each B_i to B_{i+1}, for $1 \leq i < k$, and from B_k to B_1. These edges are also present in the graph induced by $A^p \cup C^p$; therefore there is a cycle from B_1 to B_1 in this graph, so C gets added to A^m in lines 12–14.
3. Precedence rules. Suppose $\mathcal{M} \vdash A \prec B$. There are two cases, depending on whether the last rule used in the derivation is A9 or A11. For A9, an argument similar to the one for disabling rules and A10 shows that an edge from A to B is placed in $G_{\mathcal{M}}$ by line 11. For A11, note that the algorithm adds the appropriate edge to $G_{\mathcal{M}}$ whenever it infers $\overline{AB}$ or $\overline{BA}$.

$\Box$

EXAMPLE 5.1 We can run Algorithm 5.1 on the meta-rules of Example 3.3 for the airline ECA rules, and derive the following positive requirement meta-rules. No new disabling meta-rules can be derived. We do not list the trivial positive requirement meta-rules of the form $A \supset A$.
$c \supset a$, $d \supset a$, $e \supset a$, $f \supset a$, and $g \supset a$.
We also derive the following scheduling meta-rules:
$a \prec c$, $a \prec d$, $a \prec f$, $a \prec g$, $d \prec e$, $e \prec d$. $\Box$

Algorithm 5.1 gives an efficient implication test for positive requirement, disabling, and scheduling meta-rules. Implication of preference rules can be tested using the following result.

Theorem 5.2 *$\mathcal{M} \vdash B > A$ if and only if either $B \in A^+$ or if $\mathcal{M}$ contains a rule $B > C$ such that $C \subseteq A^+$.* $\Box$

Proof: The (if) direction follows from the inference axioms. If $B \in A^+$, it means then $A \supset B$. $B > B$ from axiom A7. Combining these two in axiom A8, we get that $B > A$. if $\mathcal{M}$ derives a rule $B > C$ such that $C \subseteq A^+$, then axiom A8 directly derives $B > A$.

For the (only if), suppose $\mathcal{M} \vdash B > A$. We proceed by induction on the length of a shortest derivation of $B > A$ from $\mathcal{M}$. For the base cases, if $B > A$ has a derivation of length 0, then $B > A \in \mathcal{M}$, satisfying the theorem. If $B > A$ has a derivation of length 1, then the derivation consists of axioms A7 or A8, and in both cases the theorem is verified. Now suppose the shortest derivation of $B > A$ is of length $n > 1$. Consider the last axiom in the derivation. If it is axiom A7, then $\{B\} = A$ and we are done. If it is axiom A8, there is some C such that $C \subseteq A^+$ and $B > C$ has a shortest derivation of length $n-1$ or less. By induction, either $B \in C^+$ or there is a rule $B > Z$ in $\mathcal{M}$ such that $Z \subseteq A^+$. If $B \in C^+$, then $B \in A^+$ and we are done. If there is a rule $B > Z$ in $\mathcal{M}$ such that $Z \subseteq A^+$, note that this rule satisfies the theorem by setting C to Z. □

EXAMPLE 5.2 Given the meta-rules of Example 3.3, and the positive requirement and disabling rules inferred in Example 5.1, we can use Theorem 5.2 to derive the following preference meta-rules (we do not list the trivial preference rules of the form $A > A$).
$a > b$, $a > c$, $a > d$, $a > e$, $a > f$, $a > g$, $b > c$, $b > d$, $b > e$, $b > f$. □

A polynomial time algorithm to test whether a meta-rule follows from a given set of meta-rules can be derived from Algorithm 5.1.

Theorem 5.3 *The implication problem for meta-rules is decidable in polynomial time.* □

Proof: Let n be the the number of rules in the system, i.e. the cardinality of V, and m the size of $\mathcal{M}$. In each iteration of Algorithm A.1, at least either one rule is added to A^p or A^m for some A, or an edge is added to $G_{\mathcal{M}}$, so there are $O(n^2)$ iterations. In each iteration, the work is $O(n^2 m)$, leading to $O(n^4 m)$ running time in a crude implementation. Theorem 5.2 shows that incorporating preference rules does not change the complexity. □

6 Liveness, Determinism and Well Ordering in a Rule System

We now analyze several static properties of a rule system $\mathcal{S} = \langle V, \mathcal{M} \rangle$. Static properties are those that can be determined without looking at a specific input set.

6.1 Liveness

Definition 6.1 (Live Rule): A rule A is said to be *live* in a rule system $\mathcal{S}$ if there exists a fireable set I for which A is in the execution set O. A rule that is not live will be called *dead*. □

A rule that is not live may be deleted from the rule system, along with any meta-rules that are incident upon it, thereby reducing the size of the rule system.

Theorem 6.1 *A rule A is dead in a rule system $\mathcal{S}$ if and only if the meta-rule set $\mathcal{M}$ derives the meta-rule $\overline{A}$, and hence liveness can be checked in time polynomial in size of $\mathcal{M}$.* □

Proof: The (if) part is straightforward. For the (only if), suppose A is dead and let $I = O = A^+$. Since A is dead, this cannot be a model of $\mathcal{S}$. Suppose it violates some positive rule $C \supset D$. Then $C \subseteq A^+$ and $D \notin A^+$, contradicting the transitivity of $\supset$. So it must violate some negative rule $\overline{BC}$. That is, $BC \subseteq A^+$, so we have $A \supset B$. From $A \supset C$ and $\overline{BC}$, by axioms A1 and A2 we get $\overline{AB}$, proving the theorem. □

Another useful piece of information in a rule system is which pairs of rules can occur together in some O set. For example, we need only worry about such pairs of rules when deciding what scheduling meta-rules to specify. The next theorem states that unless we have been explicitly told, or can derive on the basis of what we have been explicitly told, that two rules cannot fire together, there is at least some model in which the two could fire together (not taking into account semantics specific to the application and not analyzing the internals of either rule).

Theorem 6.2 *Given a set of meta-rules $\mathcal{M}$, rules A and B can occur in the set O of some model of $\mathcal{M}$ if and only if $\mathcal{M} \not\vdash \overline{AB}$.* $\Box$

Proof: If $\mathcal{M} \vdash \overline{AB}$, then in any model of $\mathcal{M}$ in which A appears in O, B cannot appear in O, and vice-versa. If both implications are false, consider a model of $\mathcal{M}$ in which, say, $\overline{AB}$ is false. This means A and B are both in O. $\Box$

EXAMPLE 6.1 For our airline example, we did not derive any meta-rule of the form $\overline{A}$. Therefore, by Theorem 6.1, all the rules are live. The only disabling rule derived was $\overline{de}$. Thus, by Theorem 6.2, all other pairs of rules can co-occur in the execution set for some input set.

Suppose we add the meta-rule $b \supset e$ to the airline example. Then we can derive $d \supset e$, which together with $\overline{de}$, leads to $\overline{d}$, so rule $[d]$ is dead. $\Box$

6.2 Determinism

Our task here is to characterize rule systems in which, for every fireable I, the execution set O is uniquely defined by the rules and meta-rules. We want to execute *every* fireable rule that is not blocked by the meta-rules and the other fireable rules, so we are interested in maximal execution sets.

Definition 6.2 (Deterministic Rule System): A rule system $\mathcal{S} = \langle V, \mathcal{M} \rangle$ is *deterministic* if for each input set I there is a unique maximal set O such that for one or more total orders ω on O, $\langle V, I, O, \omega \rangle$ is a model of $\mathcal{M}$. $\Box$

Determinism implies that there exists a unique set of rules that should be selected for firing. We will show that determinism can be checked in polynomial time (This is the most difficult result of the paper). Note that determinism does not imply that there is a unique execution order between the rules. The existence of a unique execution order will be called well-ordering in Section 6.3.

Non-determinism is introduced in a rule system by disabling rules; if rules A and B are fireable, and we have the meta-rule $\overline{AB}$, and no way of preferring A to B, we may have one maximal execution set where A fires and one where B does. The disabling rules must be "disambiguated" by assigning them a preference in order to guarantee determinism. A major result of this paper is that determinism of a rule system can be checked statically and efficiently. Doing so requires some pre-processing outlined below.

Definition 6.3 (Reduced rule system): Given a rule system $\mathcal{S}' = \langle V', \mathcal{M}' \rangle$, the reduced rule system $\mathcal{S} = \langle V, \mathcal{M} \rangle$ is obtained as follows:

- Construct a positive requirement graph G. The graph has one node for each rule in V'. There is an edge from node A to node B if the meta-rule $A \supset B$ is in the set $\mathcal{M}'$.
- Reduce each strongly connected component $\{A_1, \ldots, A_k\}$ of G into a single node $\mathcal{A}$.
- Reduce rule set V by replacing rules $\{A_1, \ldots, A_k\}$ by a single rule symbol $\mathcal{A}$. Reduce the meta-rule set $\mathcal{M}$ by replacing each occurrence of A_i by the new rule symbol $\mathcal{A}$. Eliminate duplicate meta-rules.
- Let the reduced rule set and meta-rule set be V and $\mathcal{M}$.
- Do liveness analysis on $\langle V, \mathcal{M} \rangle$, and eliminate all dead rules from V.

$\Box$

The reduction helps us by getting rid of all positive requirement cycles from the meta-rule set, and allows us to reason about these rules together as a set. The rules in such a cycle either must all be in the execution set, or must all be excluded from the execution set. If any one rule in a cycle is dead, all the rules will be dead. So, the analysis of these can be done faster by using one rule symbol. We also eliminate all dead rules from the system to make further analysis easier.

Definition 6.4 (Minimal disabling set): For the purposes of this section, consider the disabling meta-rule $\overline{AB}$ as indistinguishable from $\overline{BA}$. By axiom A1, this does not change the semantics. Given a reduced system $\mathcal{S} = \langle V, \mathcal{M} \rangle$, let $\mathcal{D}$ be the set of all disabling meta-rules in $\mathcal{M}$, and let $\mathcal{E}$ be all the remaining meta-rules in $\mathcal{M}$. Minimize $\mathcal{D}$ by repeated application of the following step until it no longer applies:

- If a disabling meta-rule $\mu \in \mathcal{D}$ can be derived from $\mathcal{E} \cup \mathcal{D} - \{\mu\}$, then remove μ from the set $\mathcal{D}$.

The resulting set $\mathcal{D}$ is called the minimal disabling set. The meta-rules in $\mathcal{D}$ are called minimal disabling meta-rules. □

Lemma 6.1 *Given a reduced system $\mathcal{S} = \langle V, \mathcal{M} \rangle$ the minimal disabling set $\mathcal{D}$ is unique and well-defined.* □

Proof: The disabling rules can be represented as edges in the reduced positive requirement graph G, which is acyclic. We can remove a disabling edge $\overline{AC}$ whenever there is a positive requirement path from A to B, and another disabling edge $\overline{CB}$. Since the positive requirement graph is acyclic, the process of removing edges will terminate, with an end state where a disabling edge $\overline{BC}$ remains in the graph only if:

- There is no path from B to a node E, such that the disabling edge $\overline{BE}$ was in the initial model $\mathcal{M}$.

In other words, only the disabling rules between nodes as far down in the graph as possible will be left, regardless of the order in which they are eliminated from the disabling set. □

EXAMPLE 6.2 Let $\mathcal{S}$ be defined by $V = ABCD, \mathcal{M} = \{\overline{AB}, \overline{CD}, C > D, A \supset C, B \supset D\}$. This is a reduced system since the positive requirement graph is acyclic. The minimal disabling set is $\{\overline{CD}\}$, since the rule $\overline{AB}$ can be derived as follows: From $A \supset C \wedge \overline{CD}$ we get $\overline{AD}$ by axiom A2. From $B \supset D \wedge \overline{AD}$ we get $\overline{AB}$ by axiom A2. □

EXAMPLE 6.3 For the airline rule system, the positive requirement graph is acyclic, so the reduced system is the same as the original system. Suppose we added the meta-rule $a \supset f$ to the airline rule system. Then rules $[a]$, $[b]$, and $[f]$ all positively require one another, so all three can be coalesced into one.

Further, in the running example there is only one disabling rule $\overline{de}$, and it cannot be derived from any subset of the meta-rule set. Hence the minimal disabling set is $\{\overline{de}\}$. Suppose the airline rule system were augmented with the meta-rules $g \supset d$ and $\overline{ge}$. We observe that $\overline{ge}$ can be derived from $g \supset d$ and $\overline{de}$ by application of rule A2. Hence the minimal disabling set is still $\{\overline{de}\}$.
□

Theorem 6.3 *A reduced rule system $\mathcal{S}$, comprising a set of rules V and a set of meta-rules $\mathcal{M}$, is deterministic if and only if for every minimal disabling rule $\overline{AB}$ in $\mathcal{M}$, either $\mathcal{M} \vdash (A > B)$ or $\mathcal{M} \vdash (B > A)$.* □

Proof:
(If part:) Assume the preference conditions are satisfied for each minimal disabling rule. Suppose there are two distinct possible execution sets, call them O_1 and O_2, for some fireable set I in a rule system $\mathcal{S}$ that satisfies the condition of the theorem. There must be at least one rule A in O_1 not in O_2.

The possible reasons why A cannot be added to O_2, even though it is on O_1 are (if none of these apply, then the rule A can be placed in O_2):

- A conflicts with some rule in O_2. That is, there is a meta-rule $\overline{AB}$ that can be inferred from $\mathcal{M}$ with $B \in O_2$. But A is in O_1, so it cannot conflict with any rule in O_1. Therefore $B \notin O_1$.

(Case 1:) If $\overline{AB}$ is a minimal disabling meta-rule, then by the theorem assumption, we must have either $A > B$ or $B > A$. Also, $O_1 \cup O_2 \subseteq I$, so $A \in I$ and $B \in I$. If $A > B$, then O_2 is not valid and if $B > A$ then O_1 is not valid; a contradiction.

(Case 2:) If $\overline{AB}$ is not a minimal disabling meta-rule, it must have been derived using axioms A2 or A3 on other meta-rule in $\mathcal{M}$. If derived using axiom A3, then rule A is dead, and could not have been in O_1. If derived using axiom A2, then this was derived from some disabling rule $\overline{BC}$ where $A \supset C$ holds, so C must also appear in O_1.

Case (2.a) If the $\overline{BC}$ meta-rule is minimal, then either $B > C$ or $C > B$ holds. We assumed that $B \in O_2$, and $C \in O_1$. Now, since one of the preference relations holds, it cannot be the case that B is included in one execution set, and C in the other execution set, so we get a contradiction.

Case (2.b) If the $\overline{BC}$ meta-rule is not minimal, it must be derivable. Look at its derivation, and reason with the disabling rule used to derive it. We must eventually reach a disabling rule that is minimal, closing the recursion at that point.

- There is a meta-rule $A \supset A'$ that can be inferred from $\mathcal{M}$ where $A \neq A' \notin O_2$. That is, A positively requires some distinct A' which is part of O_1 but not O_2. Repeat the above analysis with A' in place of A. Since the positive requirement graph in the reduced set is acyclic, we must exit this path due to some other meta-rule as else all the rules along this path can be placed in the execution set O_2.
- There is a meta-rule $A' > A$ that can be inferred from $\mathcal{M}$ where $A \neq A' \in I$ and $A' \notin O_2$. That is, a distinct rule A', which is part of O_1 but not O_2, is preferred over A. Repeat the analysis with A' in place of A. We must eventually leave the preference path or cycle with some other meta-rule, as else all the rules along this path can be placed in the execution set O_2.

(Only if part:) The only if part is the easy one for this theorem. Let if possible the system be deterministic, but some minimal disabling meta-rule not have the preference condition satisfied. Let $\overline{AB}$ be one such minimal disabling meta-rule, without either$A > B$ or $B > A$ being derivable from $\mathcal{M}$. Then consider the case where I comprises $A^+ \cup B^+$. No rule or pair of rules in $A^+ \cup B^+$, other than A and B, can have a disabling rule between them, because if they did, then $\overline{AB}$ could be derived from that disabling rule and from the positive requirement meta-rules, and $\overline{AB}$ would not be minimal. However, for $I = A^+ \cup B^+$, there are two possible outputs, $O = I - A$ and $O = I - B$, violating the assumption that the system is deterministic. □

Theorem 6.3 not only characterizes deterministic rule systems, but also provides guidance regarding what must be done to make the system deterministic when it is not. Specifically, it lets the programmer identify particular minimal disabling meta-rules that do not have a preference direction specified on them and add appropriate preference rules to make the system deterministic.

Theorem 6.4 *Given a rule system $\mathcal{S} = \langle V, \mathcal{M} \rangle$, one can check, in time polynomial in size of $\mathcal{M}$, whether the rule system is deterministic.* □

Proof: Given a rule system $\langle V, \mathcal{M} \rangle$, we first construct the reduced rule system. The construction of the positive requirement graph, identification of strongly connected components, and replacement of strongly connected components with a single node can clearly be done in polynomial time. Each meta-rule in the resulting set can be checked for liveness in polynomial time (Theorem 6.1), so the entire construction of the reduced rule system can be completed in polynomial time in $\mathcal{M}$. The minimal disabling set can also be constructed in polynomial time using the graph construction in the proof to Lemma 6.1. Given that implication can be checked in polynomial time, Theorem 6.3 implies that determinism can be checked in polynomial time. □

EXAMPLE 6.4 Consider the rule system $\mathcal{S}$ with $V = ABCD, and \mathcal{M} = \{\overline{AB}, \overline{CD}, C > D, A \supset C, B \supset D\}$. This rule system is deterministic. Indeed, we can list the unique (input/output) sets for all the possible 16 inputs to this system as: $\emptyset/\emptyset$, $A/\emptyset$, $B/\emptyset$, C/C, D/D, $AB/\emptyset$, AC/AC, AD/D, BC/C, BD/BD, CD/C, ABC/AC, ABD/BD, ACD/AC, BCD/C, and $ABCD/AC$.

For the airline rule system, there is a preference meta-rule $D > E$, and the only minimal disabling rule is $\overline{[d][e]}$. Thus, by Theorem 6.3 the airline rule system is deterministic. □

6.3 Well-ordering

Given a deterministic rule system, we still need to be sure that there is a unique total order ω upon each execution set O. Theorem 6.5 below provides us with a necessary and sufficient condition.

Definition 6.5 (Well-ordering): A deterministic rule system $\mathcal{S}$, comprising a set of rules V and a set of meta-rules $\mathcal{M}$, is said to be *well-ordered* if $\mathcal{M}$ implies a unique total order ω for the rules in O. That is, no matter what the input set, if rules A and B appear in the output, they must appear in the same order. □

Theorem 6.5 *A deterministic reduced rule system $\mathcal{S}$, comprising a set of rules V and a set of meta-rules $\mathcal{M}$, is well-ordered iff for every pair of rules $A, B \in V$, $\mathcal{M}$ implies one of:*
1. $A \prec B$, or 2. $B \prec A$. □

Proof: If the requirement of the theorem is satisfied, then for every pair of rules that could occur in O, there is a scheduling rule that establishes a relative order between them. Further, on account of inference axiom A10 and Theorem 4.3, we know that there can be no cycle in the relative order established pairwise between any set of rules that co-occur in O. Therefore, we have a total order of the rules in O. Since this is true for every choice of I, and hence O for the deterministic rule system, it follows that the rule system is well-ordered.

Now suppose that the requirement of the theorem is not satisfied. That is, there exists a pair of rules $A, B \in V$ for which none of the three conditions hold. From Theorem 6.2 we know that there exists at least some I for which A and B both occur in O. Further, there is some execution set in which A appears before B, and some execution set in which B appears before A, so the system is not well-ordered. □

EXAMPLE 6.5 Consider the reduced rule set $V = \{A, B, C\}$, and $\mathcal{M} = \{A \prec B, B \prec C\}$.

This rule system is deterministic. For an input I, the execution set is also I.

However, the system is not well-ordered. For the input $\{A, B, C\}$, the output can be ordered as $(A \rightarrow B \rightarrow C)$. However, for the input $\{A, C\}$, the output cannot be ordered. This is because the intermediate node B in the scheduling meta-rules can be used to infer $A \prec C$ only when B is in the output, and as we just saw, B is not always in the output.

Thus, this system is not well-ordered due to the pair (A, C). The system can be made well ordered by specifying meta-rules. A programmer may also choose to live with a non well-ordered system if he knows that the order between A and C is not important. □

EXAMPLE 6.6 For the airline rule system, Examples 5.1and 5.2, list the complete set of meta-rules that can be derived from the given set of meta-rules in Example 3.3. We repeat the complete set here (we do not list the trivial preference and positive requirement meta-rules of the form $A > A$ and $A \supset A$):

$b \supset a$, $c \supset a$, $d \supset a$, $e \supset a$, $f \supset a$, $g \supset a$, $c \supset b$, $d \supset b$, $e \supset b$, $f \supset b$, $g \supset b$.
$\overline{de}$.
$d > e$, $a > b$, $a > c$, $a > d$, $a > e$, $a > f$, $a > g$, $b > c$, $b > d$, $b > e$, $b > f$.

$a \prec b$, $a \prec c$, $a \prec d$, $a \prec f$, $a \prec g$, $b \prec c$, $b \prec d$, $b \prec e$, $b \prec f$, $b \prec g$, $c \prec f$, $c \prec g$, $d \prec f$, $d \prec g$, $e \prec f$, $e \prec g$, and $f \prec g$.

There are several pairs of rules for which neither the precedence nor the disabling rules hold. For example, (c, d) and (c, e) are two such pairs. Thus this system is not well-ordered. Now, for (c, d), it doesn't really matter what order is used, since multiplication is commutative. However, for (c, e) the order is material. At this point, the rule system designer would write a new scheduling rule, say $c \prec e$. $\square$

7 Computing the Execution Sequence

Thus far we have studied a given rule system statically, without a specific input set. From the static analysis, we have shown how to determine useful properties of the rule system, and utilize these to assist a programmer in specifying additional or different meta-rules.

In this section we consider the problem of actually applying the meta-rules given that a particular event has occurred, and multiple rules have their condition predicates satisfied. That is, given a deterministic rule system $\mathcal{S}$ and a fireable set I, how do we compute the corresponding execution set O, and establish a total order on it?

We assume the set of meta-rules $\mathcal{M}$ is reduced (Definition 6.3), that is, rules that positively require each other have been collapsed into equivalence classes. It is straightforward to translate fireable rules into their class representative before applying the algorithm, and back from class representatives into individual rules after the algorithm is done.

Algorithm 7.1 *The algorithm to compute the execution set consists of the following steps:*

1. *Initialize O equal to I.*
2. *Validate I with respect to the positive requirement meta-rules. For every pair of rules $A \in O, B \notin O$, if $\mathcal{M} \vdash (A \supset B)$, then (1) remove rule A from O, and (2) remove all rules $C \in O$ such that $\mathcal{M} \vdash A > C$.*
3. *Identify pairs of rules A and B in I, such that $\overline{AB}$ is a minimal disabling rule as given by Definition 6.4.*
4. *The system is assumed to be deterministic, so for each rule pair (A, B) identified above, one must be preferred to the other (Theorem 6.3); without loss of generality suppose $\mathcal{M} \vdash A > B$.*
   - *Delete rule B from the execution set O.*
   - *Delete all rules $C \in O$ such that $\mathcal{M} \vdash C \supset B$ or $\mathcal{M} \vdash B > C$.*
5. *Establish an order on O using the scheduling meta-rules. A total order can be established if the system is well-ordered, or if the scheduling rules on the current execution set are enough to force a total order.*

$\diamond$

EXAMPLE 7.1 Let us again consider the airline frequent flier database example. Let a tuple be inserted into the `FlightDetail` database, and as a result, let the rules $I = \{a, b, d, e, g\}$ be fireable. Running Algorithm 7.1, we find that the I set is valid with respect to positive requirement meta-rules. So O is set equal to $I = \{a, b, d, e, g\}$. In the next step, we consider the minimal disabling meta-rule $\overline{de}$, and given the preference rule $d > e$, we drop rule e from O. There is no meta-rule of the form $A \supset e$ or $e > A$ that can be derived, so nothing more needs to be done.

No other minimal disabling rule remains, so the execution set $O = \{a, b, d, g\}$ is the final execution set. As for the order, given the scheduling meta-rules shown in Example 3.3, we can obtain the total order $\omega = (a \to b \to d \to g)$.

As another example, consider another activation of the rules, leading to the input set $I = \{a, b, c, d, g\}$. No rules are dropped due to disabling rules, and we get $O = I = \{a, b, c, d, g\}$. Now two orders are possible, $\omega_1 = (a \to b \to c \to d \to g)$, and $\omega_2 = (a \to b \to d \to c \to g)$. Both of these orders produce the same result, so the rule system designer may well ignore this issue, and let the system chose any one of these orders. $\square$

Algorithm 7.1 has been stated in a manner that makes it easiest to understand. An actual implementation can be made more efficient by using graphs. We outline such an implementation here.

We define a *meta-rule graph* with one node for each live rule in the rule set V, an undirected edge for each essential disabling meta-rule, and three types of directed edges – one for each positive requirement meta-rule (from node A to node B for meta-rule $A \supset B$), one for each preference meta-rule (from node B to node A for meta-rule $A > B$), and one for each scheduling meta-rule (from node A to node B for meta-rule $A \prec B$).

When building the meta-rule graph, we only need to consider the transitive reduction of the set of positive requirement meta-rules, and we can ignore the scheduling rules that are derived from inference axiom [A8]. The meta-rule graph is reduced along the cycles involving the positive requirement edges, and the reduced graph is topologically sorted on the positive requirement edges.

When given a particular input set I, we first "project" the meta-rule graph on the set of nodes in I. The projection retains only the nodes of the graph that are in I, only the positive requirement edges in the graph that emanate from a node in I (if the destination is not in I, create a special node marked NULL), and only the scheduling, preference, and disabling edges in the graph that are between two nodes in I. Next, we traverse the projected meta-rule graph in reverse topological order (with respect to the positive requirement meta-rules), and for each node

- If there are any outgoing preference edges to NULL, delete this node, all ancestors of this node, and all nodes with preference edges to any deleted nodes.
- If there are any disabling edges incident on this node, check for preference direction on the disabling edge. Delete the node that is less preferred, all its ancestors, and all nodes with preference edges to deleted nodes.

The nodes that remain in the graph, in the topological order according to the scheduling edges, is the desired execution sequence in the execution set.

Theorem 7.1 *Algorithm 7.1 computes the execution set for a deterministic rule system in time polynomial in the number of rules in the input set I.* □

Proof: Since rule implication can be tested in polynomial time. We know that no more than $O(|I|^2)$ pairs of AB can be found. For each such pair, the time spent is at most $O(I)$ to check for additional deletions that may follow, and time polynomial in the size of $\mathcal{M}$ to check for implications. □

Theorem 7.1 guarantees us that given a set of fireable rules, determining which ones to execute, and in which order, is not too expensive. In fact, with the graph structures, we can compute the execution set in time linear in the number of meta-rules. However, one may still worry about even this polynomial (in number of rules fireable) cost in high performance applications. Our experience in the applications we have studied is that there are only a few canonical types of updates, with corresponding sets of fireable rules. One can compute the execution sequence once for each such combination, and store these in a table. Thereafter, whenever a (typical) event causes a set of rules to become fireable, one only has to look up this table to find the proper execution sequence.

Algorithm 7.1 can be modified to choose all possible execution sets when the given rule system is non-deterministic. However, the number of execution sets can be exponential in the number of rules.

8 Related Work

Active databases and rule systems have both seen a large volume of research literature. That multiple triggers (or rules) may qualify for firing simultaneously has been recognized by many people. There are three primary ways in which this is dealt with. The first is to fire all triggers that qualify [DBB+88, GJ91] in parallel. While such a mechanism has the benefits of being clean to understand, the action parts must commute for every pair of rules that may simultaneously execute, and this places a restriction on what can be programmed into the action parts of rules. The second is to use criteria such as recency, specificity [WF90, SLR88] or "certainty factors" [Dav80]. We believe this is not a good idea, since one is hard-coding application dependent heuristics into a system. A comprehensive taxonomy of different criteria for this purpose is presented in [SI89], each suitable under a different circumstance.

The third possibility is to use explicitly stated priorities between rules. [SHP88, Han89] require an absolute priority level for each rule. Starburst [WCL91] permits explicitly specified relative priorities, but uses default priorities (derived by application of a heuristic criterion) where no explicit priorities have been specified. [ACL91] discusses the management of these interactions, and the incremental maintenance of priorities derived using transitivity from those explicitly stated. Rule-based query rewrite optimizers [PHH92] use hard-coded priorities between query rewrite rules. The design of such optimizers could benefit by using our meta-rules.

[SKdM92] allows explicit specification of a "control expression", in a regular expression like syntax, to consider just one rule, or all rules, as desired, at a time from the fireable set. However, there is no notion of turning off certain rules explicitly in certain contexts, and default priorities are heuristically derived where none is explicitly specified. A similar approach has been adopted by [IN88]. They add to Datalog programs a regular expression on the alphabet of rules which allows specification of which rules are evaluated after which rules, in parallel with other rules, etc. They show how to translate a program and an expression into a locally stratified program with the same semantics.

Parallel production systems consider the entire set of eligible rules in parallel rather than one at a time. For instance, the Parullel language [SDW91] permits specification of "reduction" meta-rules in higher-order logic, using rule names as predicates and predicate names as symbols, to eliminate rules otherwise eligible for firing. A similar strategy is adopted by CREL [CKB91].

Though our scheduling meta-rules rules are similar to relative priorities, we do not rely totally upon them. In fact, we use three other types of rules to control the exact set of rules to be executed, and only then use scheduling meta-rules. To our knowledge, no one has performed an analysis of meta-rules or priorities specified to determine properties of the rule system, such as whether a unique execution order is guaranteed under all circumstance with no need to resort to heuristics.

One problem with our approach is that the programmer who introduces a new rule may find it difficult to determine its proper relationship to existing rules and come up with the appropriate meta-rules. When the number of rules becomes large, one mechanism suggested (e.g. in DEC's XCON system [SBJ87]) to manage the complexity of possible interactions is to organize the rules hierarchically into rule groups. Interactions are then limited by group boundaries. We believe that such structuring of rules in large rule systems is valuable, and results in more comprehensible rule definitions. However, such structuring does not eliminate the burden of managing interactions between rules, it merely makes the management easier through better organization.

9 Conclusions

We have presented an intuitive declarative means for a programmer to specify and manage interactions between multiple rules in an active database. We have developed a formal axiomatization for such a specification, and have shown a sound and complete algorithm to derive implications from any set of such specified meta-rules. We have developed efficient tests to check whether a given rule set is guaranteed to have a well-defined unique execution set, and a unique execution order, irrespective of which specific rules have their condition predicates satisfied simultaneously. We also provide tests for other desirable properties of rule sets, such as liveness and co-occurrence.

Often, programmers have knowledge of what is inside particular condition predicates. They can use this knowledge to determine, for instance, that if the condition for rule A is satisfied, then the condition for rule B is satisfied as well, or that the condition predicates for rules A and B cannot be satisfied at the same time. Such "assertions" on the part of the programmer, can also be encoded in the form of positive requirement and disabling meta-rules and used both to manage interactions and to verify correctness at run time.

While we have illustrated our ideas using an object-oriented active database model, the ideas are applicable to other active database models, and indeed to other rule systems as well. For example, consider the Starburst rule system [WF90] for relational databases. The set V could represent the set of rules that are activated by one of the update events on one relation, and the set I could represent the subset of V for which the condition evaluates to true. If one does not want to evaluate the conditions of all activated rules (or if one uses event-action rules), the set I is made equal to the set V.

We believe that as trigger facilities become more prevalent in databases, and are used more extensively, managing interactions between rules gains importance. In this paper, we have proposed a declarative mechanism for controlling the interaction of multiple rules, without the need to use global variables that are set by one rule and checked by others. Further, we have developed a very complete and powerful analysis system that can not only infer properties of the rule system, but can also guide the programmer in disambiguating the rule system by pointing him towards rules that have ambiguous behavior. These rule analysis techniques are inspired by a real billing application in AT&T, very similar to the problem of computing frequent flyer miles upon each flight transaction. A production system based on our rule analysis technique is being implemented.

Acknowledgments

We thank the reviewers for comments that led to many improvements in this paper. The second author was partially supported by the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre.

References

[ACL91] Rakesh Agrawal, Roberta Cochrane, and Bruce Lindsay. On maintaining priorities in a production rule system. In *Proceedings of the Seventeenth International Conference on Very Large Databases (VLDB)*, pages 479–487, Barcelona, Spain, September 3-6 1991.

[AG89] Rakesh Agrawal and Narain Gehani. Ode (object database and environment): the language and the data model. In *Proceedings of ACM SIGMOD 1989 International Conference on Management of Data*, pages 36–45, Portland, OR, May 1989.

[CKB91] D.P. Miranker C.M. Kuo and J.C. Browne. On the performance of the CREL system. *Journal of Parallel and Distributed Computing*, 13(4):424–441, December 1991.

[Dav80] R. Davis. Meta-rules: Reasoning about control. *Artificial Intelligence*, 15:179–222, 1980.

[DBB+88] Umeshwar Dayal, Barbara Blaustein, Alex Buchmann, U. Chakravarthy, Meichun Hsu, Rivka Ladin, Dennis R. McCarthy, Arnon Rosenthal, S. Sarin, Michael J. Carey, Miron Livny, and R. Jauhari. The Hipac project: Combining active databases and timing constraints. *ACM-SIGMOD Record*, 17(1):51–70, March 1988.

[GJ91] Narain Gehani and H. V. Jagadish. Ode as an active database: Constraints and triggers. In *Proceedings of the Seventeenth International Conference on Very Large Databases (VLDB)*, pages 327–336, Barcelona, Spain, September 3-6 1991.

[Han89] E. N. Hanson. An initial report on the design of Ariel. *Sigmod Record*, 18(3):12–29, September 1989.

[IN88] Tomasz Imielinski and Shamim Naqvi. Explicit control of logic programs through rule algebra. In *Proceedings of the Seventh Symposium on Principles of Database Systems (PODS)*, pages 103–115, Austin, TX, 1988.

[PHH92] Hamid Pirahesh, Joseph M. Hellerstein, and Waqar Hasan. Extensible/rule based query rewrite optimization in Starburst. In *Proceedings of ACM SIGMOD 1992 International Conference on Management of Data*, pages 39–48, San Diego, CA, June 2-5 1992.

[SDW91] S. J. Stolfo, H. M. Dewan, and Ouri Wolfson. The Parullel parallel rule language. In *ICPP*, pages II–36–II–45, 1991.

[SHP88] Michael Stonebraker, E. N. Hanson, and S. Potamianos. The Postgres rule manager. *IEEE Transactions on Software Engineering*, 14(7):897–907, July 1988.

[SI89] Timos K. Sellis and Yannis E. Ioannidis. Conflict resolution of rules assigning values to virtual attributes. In *Proceedings of ACM SIGMOD 1989 International Conference on Management of Data*, pages 205–214, Portland, OR, 1989.

[SKdM92] E. Simon, J. Kiernan, and C. de Maindreville. Implementing high level active rules on top of a relational DBMS. In Li-Yan Yuan, editor, *Proceedings of the Eighteenth International Conference on Very Large Databases*, pages 315–326, Vancouver, Canada, August 23-27 1992.

[SLR88] Timos K. Sellis, C. C. Lin, and L. Raschid. Implementing large production systems in a DBMS environment: Concepts and algorithms. In *Proceedings of ACM SIGMOD 1988 International Conference on Management of Data*, pages 404–412, Chicago, IL, 1988.

[WCL91] Jennifer Widom, Roberta Jo Cochrane, and Bruce Lindsay. Implementing set-oriented production rules as an extension to Starburst. In *Proceedings of the Seventeenth International Conference on Very Large Databases (VLDB)*, pages 275–285, Barcelona, Spain, September 3-6 1991.

[SBJ87] E. Soloway, J. Bachant, and K. Jensen. Assessing the maintainability of XCON-in-R1: Coping with the problems of a very large rule-base. *Proc. Natl. Conf. of the American Association for Artificial Int.*, July 1987.

[WF90] Jennifer Widom and Sheldon J. Finkelstein. Set-oriented production rules in a relational database system. In *Proceedings of ACM SIGMOD 1990 International Conference on Management of Data*, pages 259–270, Atlantic City, NJ, May 23-25 1990.