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 ina 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.

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.

8 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, the issue of managing multiple trigger rules is going to be one of considerable 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.

6 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 5.3), 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 6.1 *The algorithm to compute the output 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 5.4.*
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 5.2); without loss of generality suppose $\mathcal{M} \vdash A > B$.*
   - *Delete rule B from the output 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 output set are enough to force a total order.*

$\diamond$

EXAMPLE 6.1 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 6.1, 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 output set $O = \{a, b, d, g\}$ is the final output set. As for the order, given the scheduling meta-rules shown in Example 3.3, we can obtain the total order $\omega = (a \rightarrow b \rightarrow d \rightarrow g)$.

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, $\omega_1 = (a \rightarrow b \rightarrow c \rightarrow d \rightarrow g)$, and $\omega_2 = (a \rightarrow b \rightarrow d \rightarrow c \rightarrow g)$. 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. $\square$

Theorem 6.1 *Algorithm 6.1 computes the output set for a deterministic rule system in time polynomial in the number of rules in the input set I.* $\square$

Theorem 6.1 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 output set in time linear in the number of meta-rules.

Algorithm 6.1 can be modified to choose all possible output sets when the given rule system is nondeterministic. However, the number of output sets can be exponential in the number of rules.

7 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]

- Do liveness analysis on $\langle V, \mathcal{M} \rangle$, and eliminate all dead rules from V.

□

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 output set, or must all be excluded from the output 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 5.4 (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 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 5.1 *Given a reduced system $\mathcal{S} = \langle V, \mathcal{M} \rangle$ the minimal disabling set $\mathcal{D}$ is unique and well-defined.* □

EXAMPLE 5.2 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 5.3 For the airline rule system, the positive requirement graph is acyclic, so the reduced system is the same as the original system. Further, there is only one disabling rule $\overline{[d][e]}$, and it cannot be derived from any subset of the meta-rule set. Hence the minimal disabling set is $\{\overline{[d][e]}\}$. □

Theorem 5.2 *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 $\mathcal{M} \vdash (B > A)$.* □

Theorem 5.3 *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.* □

EXAMPLE 5.4 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 5.2 the airline rule system is deterministic. □

5.3 Well-ordering

Given a deterministic rule system, we still need to be sure that there is a unique total order ω upon each output set O. Theorem 5.4 below provides us with a necessary and sufficient condition.

Definition 5.5 (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 5.4 *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. $B \prec A$. □

EXAMPLE 5.5 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 output 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. □

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 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.* □

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.

Theorem 4.4 *The implication problem for meta-rules is decidable in polynomial time.* □

5 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.

5.1 Liveness

Definition 5.1 (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 5.1 *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 $\mathcal{M}$.* □

EXAMPLE 5.1 For our airline example, we did not derive any meta-rule of the form $\overline{A}$. Therefore, by Theorem 5.1, all the rules are live. The only disabling rule derived was $\overline{[d][e]}$. Thus, by Theorem 5.1, all other pairs of rules can co-occur in the output set for some input set. □

5.2 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 5.2 (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 $\mathcal{M}$. □

Determinism implies that there exists a unique set of rules that should be selected for firing. 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 5.3.

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 5.3 (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 $\mathcal{M}$.

- 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 output set O for every fireable set I?
- Are some of the meta-rules in $\mathcal{M}$ redundant?

We will answer these questions in Section 5. 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 $\mathcal{M}$. 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.

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.* □

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

[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.

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. 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 A and B 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 $\overline{[d][e]}$. At this point, we prefer to execute rule $[d]$ rather than rule $[e]$, that is, $[d] > [e]$.

Scheduling rules : $A \prec B$; If A and B both are 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. □

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. □

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. μ is of the form $A \supset B$ and if $A \in O$, then $B \in O$.
2. μ is of the form $\overline{AB}$ and $\neg(A \in O \ \& \ B \in O)$.
3. μ is of the form $B > A$ and if $AB \subseteq I \ \& \ A \in O$, then $B \in O$.
4. μ is of the form $A \prec B$ and if A and B both appear in 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 $\mu \in \mathcal{M}$. □

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

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

is a model. The term $(a \rightarrow b \rightarrow c \rightarrow d \rightarrow g)$ specifies the total order ω between the rules in the output set. □

3.3 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? What are all such *dead* rules?[1]

[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.

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.

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). The subset of I that actually executes is denoted as O, or the output 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].

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 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 rule A executes, then rule B must execute as well.

ing whether there will always be a unique set of rules selected for execution from any possible fireable set.

We begin with a motivating example in Section 2 from a frequent flier application that shares several common features with the billing application within AT&T. 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 5 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 6 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 7, and concluding remarks are presented in Section 8.

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. 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 ;
```

Managing Conflicts between Rules

(Extended Abstract)

H. V. Jagadish
AT&T Research
jag@research.att.com
://www.research.att.com/home/jag@research

Alberto O. Mendelzon*
University of Toronto
mendel@db.toronto.edu
://db.toronto.edu:8020/aomweb/aom.html

Inderpal Singh Mumick
Bell Laboratories
mumick@research.att.com
://www.research.att.com/home/mumick@research

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, and is presently being implemented within the application.

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. 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 be guaranteed. We present our techniques within the active database framework.

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 determin-

*Work of Alberto Mendelzon was partially supported by NSERC and ITRC.