In: SIGMOD Record, vol. 21 no. 3, pages 4–9, September 1992

A Denotational Semantics for the Starburst Production Rule Language

Jennifer Widom

IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120

widom@almaden.ibm.com

Abstract

Researchers often complain that the behavior of database production rules is difficult to reason about and understand, due in part to the lack of formal declarative semantics. It has even been claimed that database production rule languages inherently cannot be given declarative semantics, in contrast to, e.g., deductive database rule languages. In this short paper we dispute this claim by giving a denotational semantics for the Starburst database production rule language.

1 Introduction

Production rules in database systems allow specification of data manipulation operations that are executed automatically whenever certain events occur or conditions are met, e.g. [Han89,MD89, SJGP90,WF90]. A wide variety of semantics have been proposed for database production rule languages; see, e.g. [HW93, Sel89]. Most of these semantics are based to some extent on the *recognize-act* cycle of the *OPS5* production rule language [BFKM85]. In all cases, the semantics are described informally or, at best, as an algorithm for rule execution. The lack of formal declarative semantics for database production rule languages has been discussed at some length, particularly in the context of understanding and reasoning about rule behavior [AWH92,DD91,KdMS92,RL91]. In response to this problem, some work has been done in extending deductive database rule languages, which have a clean declarative semantics [CGT90], to include production rule capabilities. In [RL91], a declarative semantics is given for such an extension, but the extension is not as powerful as most integrated database production rule languages. In [KdMS92], a more powerful extension is considered, but no declarative semantics is specified.

In this short paper we give a denotational semantics for the Starburst rule language [WCL91, WF90], showing that a formal declarative semantics is possible for database production rules. Although the semantics is (of course) tailored for the Starburst rule language, a similar semantics should be definable for other similar database rule languages, e.g. [Han89,SJGP90]. In general, a denotational semantics for a conventional programming language is defined as a *meaning function* that takes any program in the language and produces the (input-output) function computed by that program [Sto77]. Database production rules are processed in response to user modifications on persistent data, and the effect of rule processing is additional modifications to that data [HW93]. Hence, a denotational semantics for a database production rule language is defined as a meaning function that takes any set of rules and produces the function that maps a set of modifications and a database state into the new database state that results from processing those rules.

In Section 2 we give an informal description of the Starburst rule language, which serves both to introduce the language and to illustrate the informal style in which such languages (including this one) typically are specified. Section 3 contains the formal denotational semantics, with relevant domains defined in Section 3.1, supporting functions in Section 3.2, and the meaning function in Section 3.3. This semantics is defined under the assumption that rule selection (choosing which rule to consider first when multiple rules are triggered) is deterministic; Section 3.4 modifies the semantics for nondeterministic rule selection.

2 Informal Description of the Starburst Rule Language

We give an informal description of the set-oriented, SQL-based Starburst production rule language. For numerous examples see [WCL91,WF90]. The description given here is similar to that in, e.g. [WCL91]. A more detailed but still mostly informal definition of the language is given in [WF90]. Prior to this paper, the closest to a formal specification is the execution model given in [AWH92] to prove properties of rule analysis.

Starburst production rules are based on the notion of *transitions*. A transition is a database state change resulting from execution of a sequence of data manipulation operations. Rules consider only the *net effect* of transitions, meaning that: (1) if a tuple is updated several times, only the composite update is considered; (2) if a tuple is updated then deleted, only the deletion is considered; (3) if a tuple is inserted then updated, this is considered as inserting the updated tuple; (4) if a tuple is inserted then deleted, this is not considered at all.

The syntax for defining a rule is:

> **create rule** *name* **on** *table*
> **when** *transition predicate*
> [**if** *condition*]
> **then** *action*
> [**precedes** *rule-list*]
> [**follows** *rule-list*]

The *transition predicate* specifies one or more triggering operations on the rule's *table*: **inserted**, **deleted**, or **updated**($c_1, \ldots, c_n$), where $c_1, \ldots, c_n$ name columns of the rule's *table*. The rule is triggered by a given transition if at least one of the specified operations occurred in the net effect of the transition. The optional *condition* specifies an SQL predicate. The *action* specifies an arbitrary sequence of SQL data manipulation operations to be executed when the rule is triggered and its condition is true. The optional **precedes** and **follows** clauses are used to induce a partial ordering on the set of defined rules. If a rule r_1 specifies a rule r_2 in its **precedes** list, or if r_2 specifies r_1 in its **follows** list, then r_1 is higher than r_2 in the ordering. (We also say that r_1 has *precedence* or *priority* over r_2.) When no direct or transitive ordering is specified between two rules, their order is arbitrary. Cyclic orderings are not permitted.

A rule's condition and action may refer to the current state of the database through top-level or nested SQL **select** operations. In addition, rule conditions and actions may refer to *transition tables*, which are logical tables reflecting the changes to the rule's table that have occurred during the triggering transition. At the end of a given transition, transition table **inserted** in a rule refers to those tuples of the rule's table that were inserted by the transition, transition table **deleted** refers to those tuples that were deleted, and transition tables **new-updated** and **old-updated** refer to the new and old values (respectively) of the updated tuples. A rule may refer only to transition tables corresponding to its triggering operations; note that a rule is triggered iff one or more of the corresponding transition tables is non-empty.

Rules are activated at *rule assertion points.* There is an assertion point at the end of each transaction, and there may be additional user-specified assertion points within a transaction (but not within SQL operations). We consider the semantics of rule processing at an arbitrary assertion point. The state change resulting from the user-generated database operations executed since the last assertion point (or start of the transaction) create the first relevant transition, and some set of rules are triggered by this transition. A triggered rule r is chosen from this set for *consideration.* Rule r must be chosen so that no other triggered rule has precedence over r. If r has a condition, then it is checked. If r's condition is false, then another triggered rule is chosen for consideration. Otherwise, if r has no condition or its condition is true, then r's action is executed. After execution of r's action, all rules not yet considered are triggered only if their transition predicates hold with respect to the composite transition created by the initial transition and subsequent execution of r's action. That is, these rules see r's action as if it were executed as part of the initial transition. Rules already considered (including r) have already "processed" the initial transition; thus, they are triggered again only if their transition predicates hold with respect to the transition created by r's action. From the new set of triggered rules, a rule r' is chosen for consideration such that no other triggered rule has precedence over r'. Rule processing continues in this fashion.

At an arbitrary time in rule processing, a given rule is triggered if its transition predicate holds with respect to the (composite) transition since the last time it was considered. If it has not yet been considered, it is triggered if its transition predicate holds with respect to the transition since the last rule assertion point or start of the transaction. The values of transition tables in rule conditions and actions always reflect the rule's triggering transition. When there are no triggered rules with true conditions, rule processing terminates.

3 Denotational Semantics for the Starburst Rule Language

We take as given a semantics for rule conditions and actions, which in this case is SQL predicates and operations.[1] In the informal description of Section 2, the choice of which rule to consider when multiple highest-priority rules are triggered is said to be "arbitrary". We assume initially that this rule selection is performed by a deterministic algorithm [ACL91], and we take the semantics of

[1]It may be presumptuous to assume a semantics for SQL, but we do not intend to tackle this issue here.

this algorithm as given. In Section 3.4 we modify our semantics for the case when rule selection nondeterministically chooses any eligible rule.

3.1 Domains

- Let $\mathcal{S}$ be the domain of *database states.*

  If s is a state in $\mathcal{S}$, then $s = \{t_1, t_2, \ldots, t_n\}$ where each t_i is a tuple. We assume that tuples include unique, non-reusable identifiers, and for simplicity (without loss of generality) we assume that a tuple t_i's identifier also identifies t_i's table.

- Let Δ be the domain of *sets of database changes.*

  If δ is a set of changes in Δ, then $\delta = [I, D, U]$. $I = \{\langle tid_1, v_1\rangle, \ldots, \langle tid_i, v_i\rangle\}$ where each tid_x is the identifier of a (inserted) tuple and each v_x is the null value. $D = \{\langle tid_1, v_1\rangle, \ldots, \langle tid_j, v_j\rangle\}$ where each tid_x is the identifier of a (deleted) tuple and each v_x is a (value of the deleted) tuple with identifier tid_x. $U = \{\langle tid_1, v_1\rangle, \ldots, \langle tid_k, v_k\rangle\}$ where each tid_x is the identifier of a (updated) tuple and each v_x is a (old value of the updated) tuple with identifier tid_x. No tuple identifier appears more than once in $I \cup D \cup U$.

- Let $\mathcal{R}$ be the domain of *production rules.*

  If r is a rule in $\mathcal{R}$, then r is a function that takes as arguments a set of changes δ and a database state s. It returns a boolean value, a new set of changes, and a new database state. That is:

  $$r: \ \Delta \times \mathcal{S} \rightarrow \{\mathit{true}, \mathit{false}\} \times \Delta \times \mathcal{S}$$

  $r(\delta, s)\downarrow 1 = \mathit{true}$ iff r is triggered by the changes in δ (using the obvious definition).[2] If $r(\delta, s)\downarrow 1 = \mathit{false}$ then $r(\delta, s)\downarrow 2 = [\emptyset, \emptyset, \emptyset]$ and $r(\delta, s)\downarrow 3 = s$. If $r(\delta, s)\downarrow 1 = \mathit{true}$ then $r(\delta, s)\downarrow 2$ and $r(\delta, s)\downarrow 3$ are the results of executing rule r starting with database state s and using δ for r's transition tables (with the assumed semantics of SQL). "Executing" rule r includes evaluating its condition and, if true, executing its action. Note that if r's condition is false then $r(\delta, s)\downarrow 2 = [\emptyset, \emptyset, \emptyset]$ and $r(\delta, s)\downarrow 3 = s$.

- Let $\mathcal{O}$ be the domain of *rule orderings.*

  If o is an ordering in $\mathcal{O}$, then $o = \{r_i > r_j, r_k > r_l, \ldots\}$ where each r_x is in $\mathcal{R}$ and $>$ is transitive and irreflexive (but not necessarily total).

- Let $\mathcal{RC}$ be the domain of *sets of rule-changes pairs.*

  $\mathcal{RC} \subset P(\mathcal{R} \times \Delta)$, where P is the *powerset* operator. (I.e. if A is a set, then $P(A)$ is the set of all subsets of A.) If rc is a set of rule-changes pairs in $\mathcal{RC}$, then $rc = \{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}$, and no rule appears more than once in rc.

[2] We use $\downarrow i$ to denote the ith element in a sequence.

3.2 Supporting Functions

For each function, we give an informal description, a specification of the argument and result domains, and the function definition. Function definitions are given using the *lambda-calculus* [HS86]. Briefly, $\lambda x_1,..,x_n.E$ denotes a function that takes argument values $v_1,..,v_n$ and returns the result of evaluating expression E with all free occurrence of x_i in E replaced by v_i, $1 \leq i \leq n$. Note that E may itself be a λ-expression.

- Function *Run-Rule* takes a rule r_i, a database state s, and a set of rule-changes pairs rc. It returns (1) the new database state resulting from executing rule r_i starting with database state s and using r_i's changes in rc for r_i's transition tables, and (2) the new set of rule-changes pairs that contains the net effect of r_i's changes with each set of changes in rc (except for r_i's, which is replaced by $[\emptyset, \emptyset, \emptyset]$). *Run-Rule* is undefined on and is never applied to arguments in which r_i does not appear in rc.

  $\mathit{Run\text{-}Rule}: \ \mathcal{R} \times \mathcal{S} \times \mathcal{RC} \rightarrow \mathcal{S} \times \mathcal{RC}$

  $$\mathit{Run\text{-}Rule} = \lambda\ r_i,\ s,\ \{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}.$$
  $$\langle\ r_i(\delta_i, s)\downarrow 3,\ \mathit{Add\text{-}Changes}(r_i(\delta_i, s)\downarrow 2, \{\langle r_1, \delta_1\rangle, \ldots, \langle r_i, [\emptyset, \emptyset, \emptyset]\rangle, \ldots, \langle r_n, \delta_n\rangle\}\ \rangle$$

- Function *Add-Changes* takes a set of changes δ and a set of rule-changes pairs rc. It returns the modified set of rule-changes pairs that contains the net effect of δ with each set of changes in rc.

  $\mathit{Add\text{-}Changes}: \ \Delta \times \mathcal{RC} \rightarrow \mathcal{RC}$

  $$\mathit{Add\text{-}Changes} = \lambda\ \delta,\ \{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}.\ \{\langle r_1, \mathit{Net\text{-}Effect}(\delta_1, \delta)\rangle, \ldots, \langle r_n, \mathit{Net\text{-}Effect}(\delta_n, \delta)\rangle\}$$

- Function *Net-Effect* takes two sets of changes and returns the set of changes that is their net effect. (This is similarly defined in [WF90].)

  $\mathit{Net\text{-}Effect}: \ \Delta \times \Delta \rightarrow \Delta$

  $$\mathit{Net\text{-}Effect} = \lambda\ [I_1, D_1, U_1], [I_2, D_2, U_2].$$
  $$[\,(I_1 -^* D_2) \cup I_2,\ D_1 \cup (D_2 -^* I_1),\ (U_1 -^* D_2) \cup^* (U_2 -^* I_1)\,]$$

  where $S_1 -^* S_2$ is defined as $\{\langle tid, v\rangle \in S_1 \mid tid$ does not appear in $S_2\}$ and $S_1 \cup^* S_2$ is defined as $S_1 \cup (S_2 -^* S_1)$.

- Function *Choose-Triggered* takes a set of rule-changes pairs rc and a rule ordering o. It returns a rule r in rc that is triggered by its changes in rc and is such that no other rule in rc with precedence over r in o is triggered by its changes in rc. *Choose-Triggered* is undefined on and is never applied to a set rc containing no triggered rules.

  $\mathit{Choose\text{-}Triggered}: \ \mathcal{RC} \times \mathcal{O} \rightarrow \mathcal{R}$

  $$\mathit{Choose\text{-}Triggered} = \lambda\ \{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\},\ o.\ \mathit{Select}(\mathit{Eligible}(\{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}, o))$$

- Function *Eligible* takes a set of rule-changes pairs *rc* and a rule ordering o. It returns all rules r in *rc* that are triggered by their changes in *rc* and are such that no other rule in *rc* with precedence over r in o is triggered by its changes in *rc*.

  Eligible : $\mathcal{RC} \times \mathcal{O} \rightarrow P(\mathcal{R})$ (recall that P is the powerset operator)

  $$\begin{aligned} \textit{Eligible} = \lambda\ \{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\},\ o.& \\ \{\, r_i \mid 1 \leq i \leq n\ \wedge\ r_i(\delta_i, \emptyset)\downarrow 1 = \textit{true}\ \wedge& \\ \{r_j \mid 1 \leq j \leq n\ \wedge\ r_j(\delta_j, \emptyset)\downarrow 1 = \textit{true}\ \wedge\ r_j > r_i \in o\} = \emptyset \,\}& \end{aligned}$$

- Function *Select* takes a set of rules and deterministically chooses one.

  Select : $P(\mathcal{R}) \rightarrow \mathcal{R}$

  Select is undefined on and is never applied to the empty set. As mentioned above, we take as given a definition for this function. In Section 3.4 we modify the semantics for the case when this function is nondeterministic.

3.3 The Meaning Function

The semantics is denoted by meaning function $\mathcal{M}$. $\mathcal{M}$ takes a set of rules $R \in P(\mathcal{R})$ and a rule ordering $o \in \mathcal{O}$ such that all rules in o also are in R. The meaning of R and o, denoted $\mathcal{M}[\![R, o]\!]$, is a function, call it ϕ. Function ϕ takes a set of changes δ and a database state s. It returns the new database state that results from processing the rules in R starting with initial changes δ and state s, and using the ordering in o. If rule processing does not terminate then ϕ returns $\perp$ (*bottom*). $\mathcal{M}$ is defined as follows.

$$\mathcal{M} :\ P(\mathcal{R}) \times \mathcal{O} \rightarrow \Delta \times \mathcal{S} \rightarrow \mathcal{S} \cup \{\perp\}$$

$$\mathcal{M}\,[\![\{r_1, r_2, \ldots, r_n\}, o]\!] = \lambda\ \delta,\ s.\, \mathcal{M}'(o)\,(\langle s, \textit{Distrib}(\delta, \{r_1, r_2, \ldots, r_n\})\rangle)$$

Function *Distrib* takes a set of changes δ and a set of rules R. It returns the set of rule-changes pairs that results from distributing δ to each rule in R.

$$\textit{Distrib} :\ \Delta \times P(\mathcal{R}) \rightarrow \mathcal{RC}$$

$$\textit{Distrib} = \lambda\ \delta,\ \{r_1, r_2, \ldots, r_n\}.\ \{\langle r_1, \delta\rangle, \ldots, \langle r_n, \delta\rangle\}$$

Finally, $\mathcal{M}'$ takes an ordering o and returns the least fixed point of a function F. F takes a database state s and a set of rule-changes pairs *rc*. It returns s if no rules in *rc* are triggered by their changes in *rc*. Otherwise, it calls function *Choose-Triggered* to choose a rule r_i, then applies itself to the new state and set of rule-changes pairs that result from calling function *Run-Rule* with r_i, s, and *rc*.

$$\mathcal{M}' :\ \mathcal{O} \rightarrow \mathcal{S} \times \mathcal{RC} \rightarrow \mathcal{S}$$

$$\begin{aligned} \mathcal{M}' = \lambda\ o.\, \textit{Least-Fixed-Point}(\lambda\ F.& \\ \lambda\ \langle s,\ \{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}\rangle\ .& \\ \text{if } \textit{Eligible}(\{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}, o) = \emptyset \text{ then } s& \\ \text{else let } r_i = \textit{Choose-Triggered}(\{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}, o) \text{ in}& \\ F(\textit{Run-Rule}(r_i, s, \{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\})))& \end{aligned}$$

3.4 Modified Semantics for Nondeterministic Rule Selection

When rule selection is nondeterministic, the function ϕ produced by meaning function $\mathcal{M}$ returns the set of all database states (rather than a single state) that can result from processing the rules in R starting with initial changes δ and state s, and using the ordering in o. If rule processing cannot terminate (regardless of which rules are selected) then ϕ returns the empty set. $\mathcal{M}$ is defined as follows.

$$\mathcal{M}: \; P(\mathcal{R}) \times \mathcal{O} \rightarrow \Delta \times \mathcal{S} \rightarrow P(\mathcal{S})$$
$$\mathcal{M}\,[\![\{r_1, r_2, \ldots, r_n\}, o]\!] = \lambda\ \delta,\ s.\, \mathcal{M}'(o)\,(\langle s, \mathit{Distrib}(\delta, \{r_1, r_2, \ldots, r_n\})\rangle)$$

Function *Distrib* is defined as in Section 3.3. $\mathcal{M}'$ applied to an ordering o is the least fixed point of a function F that takes a database state s and a set of rule-changes pairs rc. F returns $\{s\}$ if no rules in rc are triggered by their changes in rc. Otherwise, F calls function *Eligible* (from Section 3.2) to find all eligible triggered rules, then takes the union of the results of applying itself, for each eligible rule r_i, to the new state and set of rule-changes pairs that result from calling function *Run-Rule* with r_i, s, and rc. Note that function *Choose-Triggered* (and consequently function *Select*) is not used here.

$$\mathcal{M}': \; \mathcal{O} \rightarrow \mathcal{S} \times \mathcal{RC} \rightarrow P(\mathcal{S})$$
$$\begin{aligned}
\mathcal{M}' = \lambda\ o.\ & \mathit{Least\text{-}Fixed\text{-}Point}(\lambda\ F. \\
& \quad \lambda\ \langle s,\ \{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}\rangle\ . \\
& \qquad \text{if } \mathit{Eligible}(\{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}, o) = \emptyset \text{ then } \{s\} \\
& \qquad \text{else let } R' = \mathit{Eligible}(\{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}, o) \text{ in} \\
& \qquad\quad \textstyle\bigcup_{r_i \in R'} F(\mathit{Run\text{-}Rule}(r_i, s, \{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\})))
\end{aligned}$$

Note that this semantics does not distinguish between the case in which rule processing always terminates and the case in which rule processing may or may not terminate (depending on which rules are selected): In both cases, the function produced by $\mathcal{M}$ returns a non-empty set of database states. We can extend our semantics to include this distinction. Let there be an additional domain $\mathcal{T}$ containing one element called *term*: $\mathcal{T} = \{\mathit{term}\}$. We modify the function ϕ produced by meaning function $\mathcal{M}$ to return the set of all database states that can result from processing the rules in R along with either *term* or $\perp$. If *term* is returned then rule processing always terminates; if $\perp$ is returned then rule processing may not terminate.

$$\mathcal{M}: \; P(\mathcal{R}) \times \mathcal{O} \rightarrow \Delta \times \mathcal{S} \rightarrow P(\mathcal{S}) \times (\mathcal{T} \cup \{\perp\})$$
$$\mathcal{M}\,[\![\{r_1, r_2, \ldots, r_n\}, o]\!] = \lambda\ \delta,\ s.\, \mathcal{M}'(o)\,(\langle s, \mathit{Distrib}(\delta, \{r_1, r_2, \ldots, r_n\})\rangle)$$

We modify the function F in $\mathcal{M}'$ to return $\langle \{s\}, \mathit{term}\rangle$ if no rules are triggered. Otherwise, F calls function *Eligible* to find all eligible triggered rules, then takes the "composition" (defined below) of the results of applying itself, for each eligible rule r_i, to the new state and set of rule-changes pairs that result from calling function *Run-Rule* with r_i, s, and rc.

$$\mathcal{M}': \; \mathcal{O} \rightarrow \mathcal{S} \times \mathcal{RC} \rightarrow P(\mathcal{S}) \times (\mathcal{T} \cup \{\perp\})$$

$$\begin{array}{l} \mathcal{M}' = \lambda\, o.\, \textit{Least-Fixed-Point}(\lambda\, F. \\ \qquad\qquad \lambda\, \langle s,\, \{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}\rangle\, . \\ \qquad\qquad\qquad \text{if } \textit{Eligible}(\{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}, o) = \emptyset \text{ then } \langle\{s\}, \textit{term}\rangle \\ \qquad\qquad\qquad \text{else let } R' = \textit{Eligible}(\{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}, o) \text{ in} \\ \qquad\qquad\qquad\quad \bigcirc_{r_i \in R'} F(\textit{Run-Rule}(r_i, s, \{\langle r_1, \delta_1\rangle, \ldots, \langle r_n, \delta_n\rangle\}))) \end{array}$$

where $\langle S_1, t_1\rangle \circ \langle S_2, t_2\rangle$ is defined as $\langle S_1 \cup S_2,\, t\rangle$, with $t = \bot$ if $t_1 = \bot$ or $t_2 = \bot$ and $t = \textit{term}$ otherwise.

Acknowledgements

A discussion with Louiqa Raschid inspired me to work this out. Alex Aiken and Ed Wimmers made very insightful contributions. Bill Cody provided useful comments.

References

[ACL91] R. Agrawal, R.J. Cochrane, and B. Lindsay. On maintaining priorities in a production rule system. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, pages 479–487, Barcelona, Spain, September 1991.

[AWH92] A. Aiken, J. Widom, and J.M. Hellerstein. Behavior of database production rules: Termination, confluence, and observable determinism. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 59–68, San Diego, California, June 1992.

[BFKM85] L. Brownston, R. Farrell, E. Kant, and N. Martin. *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming.* Addison-Wesley, Reading, Massachusetts, 1985.

[CGT90] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer-Verlag, Berlin, 1990.

[DD91] U. Dayal and K.R. Dittrich. Active database systems. In *Seventeenth International Conference on Very Large Data Bases* (tutorial), Barcelona, Spain, September 1991.

[Han89] E.N. Hanson. An initial report on the design of Ariel: A DBMS with an integrated production rule system. *SIGMOD Record, Special Issue on Rule Management and Processing in Expert Database Systems*, 18(3):12–19, September 1989.

[HS86] J.R. Hindley and J.P. Seldin. *Introduction to Combinators and λ-Calculus.* Cambridge University Press, Cambridge, England, 1986.

[HW93] E.N. Hanson and J. Widom. An overview of production rules in database systems. *The Knowledge Engineering Review*, 8(2):121–143, June 1993.

[KdMS92] J. Kiernan, C. de Maindreville, and E. Simon. Supporting deductive and active rules on top of a relational DBMS. Research report, INRIA, Le Chesnay, France, 1992.

[MD89] D.R. McCarthy and U. Dayal. The architecture of an active database management system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 215–224, Portland, Oregon, May 1989.

[RL91] L. Raschid and J. Lobo. Semantics for update rule programs and implementation in a relational database management system. Technical Report UMIACS-TR-91-140, Institute for Advanced Computer Studies, University of Maryland, 1991.

[Sel89] T. Sellis, editor. *Special Issue on Rule Management and Processing in Expert Database Systems,* SIGMOD Record 18(3), September 1989.

[SJGP90] M. Stonebraker, A. Jhingran, J. Goh, and S. Potamianos. On rules, procedures, caching and views in data base systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 281–290, Atlantic City, New Jersey, May 1990.

[Sto77] J.E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. The MIT Press, Cambridge, Massachusetts, 1977.

[WCL91] J. Widom, R.J. Cochrane, and B.G. Lindsay. Implementing set-oriented production rules as an extension to Starburst. In *Proceedings of the Seventeenth International Conference on Very Large Data Bases*, pages 275–285, Barcelona, Spain, September 1991.

[WF90] J. Widom and S.J. Finkelstein. Set-oriented production rules in relational database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 259–270, Atlantic City, New Jersey, May 1990.