# Formal Characterization of Active Databases

Chitta Baral[1] and Jorge Lobo[2]

[1] Department of Computer Sc. Univ. of Texas at El Paso El Paso, TX, 79968, USA
[2] Department of EECS Univ. of Illinois at Chicago Chicago, IL 60607, USA

**Abstract.** In this paper we take a first step towards characterizing active databases. Declarative characterization of active databases allows additional flexibility in studying the effects of different priority criteria between fireable rules, different actions and event definitions, and also to make claims about effects of transaction and prove them without actually executing them. Our characterization is related but different from similar attempts by Zaniolo in terms of making a clear distinction between actual and hypothetical execution of actions and allowing non-determinism. We use the 'choice' construct [SZ90] to characterize the non-determinism that arises when several rules can fire at the same time and the preference between them is not specified. We show through examples how our language allows us to express features of different active database systems.

## 1 Introduction and motivation

The core concept that makes a database active is the concept of an active rule. An active rule is generally composed of three parts: an *event* that causes the rule to be triggered, a *condition* that is checked when the rule is triggered, and an *action* that is executed if the condition is satisfied when the rule is triggered. Based on this simple but powerful idea several database systems have been designed and implemented [CFPB96, DBC96, GJ96, Han96, PS96, SK96, Wid96]. Active rules are used to perform activities such as integrity constraint and view maintenance, monitor services and work-flow management.

Although there has been considerable research and development in this area (as reflected in the recent book edited by Widom and Ceri [WC96]), there has been very little activity in the study of formal foundations. In the 'Future Directions' section of the 'Conclusion' chapter in [WC96], Widom and Ceri express this sentiment by the following:

> Finally, two largely neglected and crucially important areas within the field of active database systems are *formal foundations* and *usability*. Although some very preliminary work has been done in the area of formal foundations [FMT94, Wid92, Zan93], there is no unifying theory underlying active database systems ... A formal foundation for active database

rule languages would provide a very important step in understanding and characterizing the commonalities and differences across systems.

In general, besides the informal definition of an active rule, there is no consensus about the definition of events, how to process rules when several of them are triggered simultaneously, and how to characterize the set of possible actions. The differences arise because the appropriate alternative is determined by the specific application and most of the options are reasonable. These dissimilarities among the systems make a formal or declarative characterization difficult, and at this point, the only characterization that exists for many of the systems is operational.

The goal of this paper is to take a first step towards a formal framework in which we can precisely describe several active database systems. We develop a language called $\mathcal{L}_{active}$ where we can write descriptions of active databases. Our language borrows from the action description language $\mathcal{L}_0$ [BGP96] which has a clear distinction between actual and hypothetical occurrences of actions. In it, the authors emphasize the hypothetical nature of the situation calculus and argue that the action occurrences in the situations $res(a_1, s_0)$ (i.e. the situation that results from the application of action $a_1$ to situation $s_0$) and $res(a_2, s_0)$ should be considered to be hypothetical, since only one action[3] could have really happened in the situation $s_0$. This distinction is overlooked in most formalization of database updates (such as [Rei94, MW88]).

In $\mathcal{L}_{active}$ we allow the description of actions and their effects, definition of events, and various modes of evaluations for the active rules. This gives us the flexibility to express different event definitions in different active database systems (for example, the definitions in Starburst [Wid96] and A-RDL[SK96]) in a single framework.

The semantics of $\mathcal{L}_{active}$ is based on the automata-based semantics of action description languages, and allows us to define an entailment relation between a database description and queries about the state of the database after the execution of a sequence of transactions. Because of its commonality with the recent works on action description languages our language can be easily extended to incorporate additional features such as concurrent actions [BG96] and deductive rules [KL94, Bar95, LR94].

We start with a presentation of the syntax and semantics of the language. We then present a translation that takes an active database description into a logic program. This translation provides us a vehicle to actually compute the entailment relation. We use the 'choice' construct to characterize the non-determinism and ideas from the situation calculus to characterize the dynamic nature of an

[3] We are assuming that our language does not allow concurrent execution of actions.

active database. We conclude the paper with some general remarks on how our language must be generalized to capture many other properties of active database systems.

## 2 Syntax of $\mathcal{L}_{active}$

The alphabet of $\mathcal{L}_{active}$ consists of five disjoint nonempty sets of symbols $\mathcal{F}$, $\mathcal{A}$, $\mathcal{S}$, $\mathcal{E}$, and $\mathcal{M}$ called *fluents*, *actions*, *actual situations*, *events*, and *modes*. Elements of $\mathcal{A}$ will be denoted by (possibly indexed) letters $a$. We will also assume that $\mathcal{S}$ contains situations $s_0, \ldots, s_n$ (for an arbitrary large number $n$) and a special symbol $s_{now}$. The symbols $s_0$ and $s_{now}$ are referred to as *initial* and *current* situations, respectively.

Fluents are data items that can change their values in the database and they can take different forms according to the kind of database being used. For example, in a relational database, fluent names are tuples that can appear in a relation. In an object oriented database it could be the name of an object attribute together with a value in the domain of that attribute. Variables can replace the attribute's value in a fluent name, and they will represent parameters that can be replaced by any value in the domain of the attribute. To minimize the introduction of new notation we will present our examples loosely based on the relation model. A *fluent literal* is a fluent possibly preceded by negation $\neg$. Fluent literals will be denoted by (possibly indexed) letters $f$ and $p$ (possibly preceded by $\neg$). $\neg\neg f$ will be equated with $f$. For any fluent $f$, $\overline{\neg f} = f$, and $\overline{f} = \neg f$. Similar to fluent literals, an *event literal* is an event possibly preceded by negation $\neg$.

We denote sequences[4] of actions by the Greek letter $\alpha$ and its indexed versions. For an action $a$, $\alpha \circ a$, means the sequence of actions where $a$ follows $\alpha$. We also sometimes denote sequence of actions with the Prolog's notation for lists. For example, the list of actions $[a_1, a_2, \ldots, a_n]$, denotes the sequence of actions where $a_1$ is followed by $a_2$ and so on up to $a_n$.

There are six kinds of propositions in $\mathcal{L}_{active}$ called *causal (or effect) laws, executability conditions, fluent facts, user-dictated action occurrences, active rules,* and *event definitions.*

- A *causal law* is an expression of the form

$$a \textbf{ causes } f \textbf{ if } p_1, \ldots, p_n \tag{1}$$

[4] In this paper by sequence we mean a finite sequence, except when is explicitly specified.

where $a$ is an action, and $f, p_1, \ldots, p_n$ ($n \geq 0$) are fluent literals. $p_1, \ldots, p_n$ are called *preconditions* of (1). We will read this law as "$f$ is guaranteed to be true after the execution of an action $a$ in any state of the world in which $p_1 \ldots p_n$ are true".

If $n = 0$, we write the effect law as

$$a \textbf{ causes } f \tag{1a}$$

*Example 1.* The actions common to most database systems are the SQL operations *insert*, *delete* and *update* which we refer to in the rest of the paper by *add*, *del* and *upd* respectively. Following are causal laws describing the effect of these actions.

$add(F)$ **causes** $F$

$del(F)$ **causes** $\neg F$

$upd(F, G)$ **causes** $G$

$upd(F, G)$ **causes** $\neg F$

Many other actions can be specified in $\mathcal{L}_{active}$. We will only remark that actions can have different effects based upon the preconditions of the action. For example, the following action keeps count of the resource allocated to systems $p$ and $q$.[5]

$balance\_load\_pq(N)$ **causes** $p(X)$ **if** $p(Y), q(Z), Y \geq Z, X = Y + N.$

$balance\_load\_pq(N)$ **causes** $q(X)$ **if** $p(Z), q(Y), Y > Z, X = Y + N.$ $\square$

Two causal laws of the form

$a$ **causes** $f$ **if** $p_1, \ldots, p_n$ and
$a$ **causes** $\neg f$ **if** $q_1, \ldots, q_m$

are said to be *contradictory* if $\{p_1, \ldots, p_n\} \cap \{\overline{q_1}, \ldots, \overline{q_m}\} = \emptyset$.

- An *executability condition* is an expression of the form

$$\textbf{executable } a \textbf{ if } q_1, \ldots, q_k \tag{2}$$

where $a$ is an action name, and each of $q_1, \ldots, q_k$ ($k \geq 0$) is a fluent literal. About this proposition we say that it stipulates that $a$ is executable in a situation where $q_1, \ldots, q_k$ are satisfied.

[5] Strictly speaking we will need to define $>$ and $=$ fluents, and the operation $+$, but to simplify the presentation we appeal to the intuition of the reader for the semantics of these predicates.

*Example 2.* Example of executability conditions for inserts, deletes and updates are:

$$\textbf{executable} \;\; add(F) \;\textbf{if}\;\; \neg F$$

$$\textbf{executable} \;\; del(F) \;\textbf{if}\;\; F$$

$$\textbf{executable} \;\; upd(F,G) \;\textbf{if}\;\; F, \neg G$$

An executability condition for $balance\_load\_pq(N)$ could be dictated by a constant $\kappa$ that the total load must not surpass.

$$\textbf{executable} \;\; balance\_load\_pq(N) \;\textbf{if}\;\; p(X), q(Y), \kappa > X + Y + N. \qquad \square$$

- An atomic *fluent fact* is an expression of the form

$$f \;\textbf{at}\;\; s_0 \tag{3}$$

where $f$ is a fluent literal and $s_0$ is the initial situation. The intuitive reading of (3) is "$f$ is observed to be true in the initial situation".

- A *user-dictated occurrence fact* is an expression of the form

$$\alpha \;\textbf{ordered\_at}\;\; s \tag{4}$$

where $\alpha$ is a sequence of actions, and $s$ is a situation constant. It states that "the sequence $\alpha$ of actions was ordered by the user for execution in situation $s$" (we assume that actions in the sequence follow each other immediately). User-dictated facts can be thought to be transactions ordered by users of the database. The transaction is defined by $\alpha$. The situation $s$ determined when the transaction happens. It is assumed that situation $s_i$ precedes situation $s_{i+1}$.

- An *active rule* is an expression of the form

$$\mathbf{r} : e \;\textbf{evaluated\_in\_mode}\; m \;\textbf{under\_conditions}\; p_1, \ldots, p_n \;\textbf{triggers}\; \alpha \tag{5}$$

where $e$ is an event, $m$ is the evaluation mode of the active rule, and $\mathbf{r}$ is the name of the rule. The rule in eqn(5) intuitively states that the event $e$ under conditions $p_1, \ldots, p_n$ triggers $\alpha$. $\alpha$ is referred to as the trigger of this active rule. There are two possible modes of execution $m$, *ignore* and *reconsider*.[6] The modes designate at what points a rule is considered for evaluation. Rules with *ignore* mode are considered only at points where events are generated. If the mode is *reconsidered*, the rule can be triggered: 1) At every point where events are generated, and 2) After an active rule has been fired. Events are generated

[6] We anticipate adding other modes of executions to our language.

by two types of situations, the situation that results after the execution of $\alpha$ in a user-dictated fact, and the situation that results after the evaluation of the active rules. (Note that in the second case, the first rule is triggered at point where an event was generated, any rule can be considered for execution. Subsequent rules in the batch must be in *reconsider* mode.)

If $n = 0$ in (5) we write it as:

$$\mathbf{r} : e \ \textbf{evaluated\_in\_mode} \ m \ \textbf{triggers} \ \alpha \tag{6}$$

*Example 3.* Let us consider the following example from Zaniolo [Zan96], where the database contains the relations:

$$Dept(D\#, Dname, Div, Loc) \text{ and } Emp(E\#, Ename, Jobtitle, Sal, D\#)$$

and there is an active rule which says: "When a tuple is deleted from a department, then delete all employees who were working in the deleted department."

We can represent the above rule in our syntax by the following active rule:

$$\begin{aligned} r_1 : del_e(dept(D\#, Dname, Div, Loc)) \ & \textbf{evaluated\_in\_mode} \ reconsider \\ & \textbf{triggers} \ del\_all\_emps(D\#) \end{aligned}$$

In the above active rule we use the subscript $e$ to distinguish between events and actions that otherwise look similar.

To describe the effect of the action $del\_all\_emps(D\#)$ we will need the following causal law:

$$del\_all\_emps(D\#) \ \textbf{causes} \ \neg Emp(E\#, Ename, Jobtitle, Sal, D\#)$$

In the sequel we will consider the possibility of having $del\_all\_emps(D\#)$ as a complex action defined in terms of the simple actions *add* and *del*. □

Rule execution is driven by the events that take place in the life of the database. The kind of events that can trigger a rule significantly varies from active database to active database, but most of the events can be characterized in terms of (a) the changes that occur in the database during the execution of a sequence of actions, and (b) the sequence of actions itself. This sequence of actions could be a transaction that has been dictated to the database by a user (external actions) or a series of actions generated by the execution of the active rules (internal actions). In $\mathcal{L}_{active}$ events can be defined in terms of the current state of database, the current action to be executed, and the events that have occurred so far. Thus,

- An *event definition* is an expression of the form

$$e \textbf{ after } a \textbf{ if } q_1, \ldots, q_m, e_1, \ldots, e_n \tag{7}$$

where $e$ is an event, $e_i$'s are event literals, and $q_j$'s are fluent literals. It states that the execution of $a$ induces an event $e$, if $q_1, \ldots, q_m$ hold (before $a$ is executed) and $e_1, \ldots, e_n$ were induced (before $a$ is executed). If $n$ and $m$ are both 0, we will write

$$e \textbf{ after } a \tag{8}$$

This simple syntactic structure allows us to define many types of events. Take, for example, the events in Starburst [Wid96]. They are defined in terms of net effects of the transitions instead of the individual operations generating the transitions. Net effects are defined based on the following four premises.

1. If a tuple is first inserted and then updated, then this is considered an insertion of the updated tuple.
2. If a tuple is first updated and then deleted, then this is considered the deletion of the original tuple.
3. If a tuple is updated several times, this is considered an update to the newest value.
4. If a tuple is first inserted and then deleted the actions are ignored.

The four rules can be defined in $\mathcal{L}_{active}$ as

$$\mathit{add}_e(H) \textbf{ after } \mathit{upd}(G,H) \textbf{ if } \mathit{add}_e(G)$$

$$\mathit{del}_e(G) \textbf{ after } \mathit{del}(F) \textbf{ if } \mathit{upd}_e(G,F)$$

$$\mathit{upd}_e(G,I) \textbf{ after } \mathit{upd}(H,I) \textbf{ if } \mathit{upd}_e(G,H)$$

$$\mathit{add}_e(F) \textbf{ after } \mathit{del}(G) \textbf{ if } \mathit{add}_e(F) \quad (F \neq G)$$

Note that these are schemas of event definitions where $G$, $F$ and $H$ range over all the fluents. The inequality in the last line is part of the schema. There are other active database system that use more complicated rules to define net effects. For example, the A-RDL system [SK96] has seventeen rules. They can be easily encoded in $\mathcal{L}_{active}$ using seventeen event definitions.

There are other kinds of events that are not defined in terms of net effects, but in complicated semantic criteria. For example, we can define the event “sharp increase of an object value” if the value of a particular object in the database is updated to a value 10% higher than the current value. The definition of this event in $\mathcal{L}_{active}$ will be:

$$sharp\_increase_e(A) \textbf{ after } upd(value(A,Y), value(A,Z)) \textbf{ if } Z \geq Y + Y * 0.1$$

From the syntax of the language and the above examples it is clear that our language allows us to easily express actions (and their effects) and event definitions beyond the commonly used *add*, *delete* and *update*.

A collection of causal laws, executability conditions, fluent facts, user-dictated action occurrences, active rules and event definitions is called an *active database description* of $\mathcal{L}_{active}$. We will only consider active database descriptions whose propositions do not mention the situation constant $s_{now}$, and which do not have contradictory causal laws.

## 3 Semantics of $\mathcal{L}_{active}$

In this section we introduce a semantics of our active database specification language $\mathcal{L}_{active}$. We assume that states of the database are determined by sets of fluents and that actions executed in a particular state can add and remove such fluents according to the causal laws. The set of all possible behaviors of a dynamic system, satisfying causal laws and executability conditions of a given active database description $D$, can then be described by a transition diagram with states labeled by sets of fluents and transitions labeled by actions. To interpret the facts from $D$ we need to select the initial situation or state together with a path in the diagram describing the actual behavior of the system. In the following subsections we describe an automaton by a partial function from action sequences to states.

### 3.1 States and Causal Interpretations

A *state* is a set of fluents. A *causal interpretation* is a partial function $\Psi$ from sequences of actions to states such that:

(1) The empty sequence [ ] belongs to the domain of $\Psi$ and

(2) $\Psi$ is prefix-closed.[7]

$\Psi([\,])$ is called the initial state of $\Psi$. The partial function $\Psi$ serves as an interpretation of the causal laws and executability conditions of $D$.

Given a fluent $f$ and a state $\sigma$, we say that $f$ *holds* in $\sigma$ ($f$ is *true* in $\sigma$) if $f \in \sigma$; $\neg f$ *holds* in $\sigma$ ($f$ is $false$ in $\sigma$) if $f \notin \sigma$. We say an action $a$ is executable in a

[7] By "prefix closed" we mean that for any sequence of actions $\alpha$ and action $a$, if $\alpha \circ a$ is in the domain of $\Psi$ then so is $\alpha$.

state $\sigma$ if there exists an executability condition of the form (2) with action $a$, such that the preconditions $q_1, \ldots, q_k$ hold in $\sigma$.

Now we define effects of actions determined by causal laws and executability conditions of an active database description $D$.

A fluent literal $f$ is an (immediate) *effect* of (executing) $a$ in $\sigma$ if there is an effect law $a$ **causes** $f$ **if** $p_1, \ldots, p_n$ in $D$ whose preconditions $p_1, \ldots, p_n$ hold in $\sigma$. Let

$E_a^+(\sigma) = \{f : f \in \mathcal{F}$ and $f$ is an effect of $a$ in $\sigma\}$,

$E_a^-(\sigma) = \{f : f \in \mathcal{F}$ and $\neg f$ is an effect of $a$ in $\sigma\}$ and

$Res(a, \sigma) = (\sigma \cup E_a^+(\sigma)) \setminus E_a^-(\sigma)$. $Res$ is referred to as the *transition function.*

The following definition captures the meaning of causal laws and executability conditions of $D$.

**Definition 1.** A causal interpretation $\Psi$ *satisfies* the causal laws and executability conditions of $D$ if for any sequence $\alpha \circ a$ from the language of $D$,

if $E_a^+(\Psi(\alpha)) \cap E_a^-(\Psi(\alpha)) = \emptyset$, $\Psi(\alpha)$ is defined and $a$ is executable in $\Psi(\alpha)$

then $\Psi(\alpha \circ a) = Res(a, \Psi(\alpha))$,

otherwise $\Psi(\alpha \circ a)$ is undefined.

We say $\Psi$ is a *causal model* of $D$ if it satisfies the causal laws and executability conditions of $D$. $\square$

It is easy to see that causal models of active database descriptions are uniquely determined by their initial values, i.e., for any causal models $\Psi_1$ and $\Psi_2$ of an active database description $D$, if $\Psi_1([\,]) = \Psi_2([\,])$ then $\Psi_1 = \Psi_2$.

### 3.2 Situation Assignments, Event Interpretations and Trigger Interpretations

**Definition 2.** Let $\Sigma$ be a mapping from $\mathcal{S}$ to sequences of actions from the language of $D$. This mapping will be called *a situation assignment of $\mathcal{D}$* if it satisfies the following properties:

1. $\Sigma(s_0) = [\,]$;

2. For every $s \in \mathcal{S}$, $\Sigma(s)$ is a prefix of $\Sigma(s_{now})$. □

Intuitively, the set of events that are induced by the execution of an action $a$, is determined by a finite sequence of actions that have occurred previously to $a$, the action $a$, and the current state of the database. More precisely, an *event interpretation* is a mapping $\Xi$ from the set of finite sequences of actions, and the set of states to the power set of events. For an event $e$, if $e \in \Xi(\alpha, \sigma)$, then we say that $e$ is induced by $\alpha$ in state $\sigma$. For an event literal $e'$ such that $e' = \neg e$, where $e$ is an event, we say $e'$ is induced by $\alpha$ in state $\sigma$ if $e \notin \Xi(\alpha, \sigma)$.

**Definition 3.** An event interpretation $\Xi$ is said to satisfy the event definitions of an active database description $D$ if

$\Xi([], \sigma) = \emptyset$, and

$\Xi(\alpha \circ a, \sigma) = \{e$ : there exists an event description of the form (7) such that $q_1, \ldots, q_m$ hold in $Res(\alpha, \sigma)$, and for all $e_i$, $1 \leq i \leq n$, $e_i$ is induced by $\alpha$ in the state $\sigma.\}$ □

Informally, the execution of active rules can be described by the following loop.

1. Events trigger a set of active rules. If the set if empty the loop terminates.
2. One of these rules is selected and executed.
3. Then, if there are *reconsider* rules in the set of triggered rules, we go to step 2.
4. If there are no *reconsider* rules, the events are re-evaluated (based on the actions executed by the triggered rules) and we go to step 1.

Thus, given an initial sequence of actions (that may generate events) and an initial state, a *trigger interpretation* will define the sequence of actions that are triggered by the active rules according to the steps in the loop. Precisely, a trigger interpretation $\mathcal{T}$ will be a function that maps a sequence of actions and a situation into a (not necessarily finite) sequence of actions.

**Definition 4.** A trigger interpretation $\mathcal{T}$ is said to satisfy the active rules of an active database description $D$ if, for any sequence of actions $\alpha$, and a state $\sigma$, if

$$\mathcal{T}(\alpha, \sigma) = \beta_{1,1}, \ldots, \beta_{1,k_1}, \ldots, \beta_{m,1}, \ldots, \beta_{m,k_m}, \ldots$$

then (with the denotation of $\alpha$ by $\alpha_0$ and $\beta_{j,1}, \ldots, \beta_{j,k_j}$ by $\alpha_j$)

1. there exists an active rule of the form (5) with event $e$, conditions $p_1, \ldots, p_n$, and trigger $\beta_{j,1}$ $(j \geq 1)$, such that
   - $e$ is in $\Xi(\alpha_{j-1}, Res(\alpha_0 \circ \ldots \circ \alpha_{j-2}, \sigma))$,[8] and
   - $p_1, \ldots, p_n$, hold in $Res(\alpha_0 \circ \ldots \circ \alpha_{j-1}, \sigma)$.
2. there exists an active rule of the form (5) that has not been used in the triggering of $\beta_{j,1}, \ldots, \beta_{j,l-1}$, with mode *reconsider*, event $e$, conditions $p_1, \ldots, p_n$, and trigger $\beta_{j,l}$ $(j \geq 1, l \geq 2)$, such that
   - $e$ is in $\Xi(\alpha_{j-1}, Res(\alpha_0 \circ \ldots \circ \alpha_{j-2}, \sigma))$, and
   - $p_1, \ldots, p_n$, hold in $Res(\alpha_0 \circ \ldots \circ \alpha_{j-1} \circ \beta_{j,1} \circ \ldots \circ \beta_{j,l-1}, \sigma)$ and

If $\mathcal{T}(\alpha, \sigma) = \beta_{1,1}, \ldots, \beta_{1,k_1}, \ldots, \beta_{m,1}, \ldots, \beta_{m,k_m}$, (i.e. $\mathcal{T}(\alpha, \sigma)$ is a finite sequence) then besides the above two conditions, the following condition is also satisfied:

"There does not exist an active rule of the form (5) with event $e$, and conditions $p_1, \ldots, p_n$, such that the conditions hold in $Res(\alpha_0 \circ \ldots \circ \alpha_m, \sigma)$ and $e$ is in $\Xi(\alpha_m, Res(\alpha_0 \circ \ldots \circ \alpha_{m-1}, \sigma))$." □

**Definition 5.** An *interpretation* $M$ of $\mathcal{L}_{active}$ is a triplet $(\Psi, \Sigma, \Xi)$, where $\Psi$ is a causal model of $D$, $\Sigma$ is a situation assignment of $\mathcal{S}$, and $\Xi$ is an event interpretation. We have the added restriction that $\Sigma(s_{now})$ belong to the domain of $\Psi$. □

**Definition 6.** For any interpretation $M = (\Psi, \Sigma, \Xi)$

(1) ($f$ **at** $s_0$) is true in $M$ (or satisfied by $M$) if $f$ is true in $\Psi([])$;

(2) ($\alpha$ **ordered_at** $s_i$) is true in $M$ if $\Sigma(s_i)$ is finite and $\Sigma(s_{i+1}) = \Sigma(s_i) \circ \alpha \circ \mathcal{T}(\alpha, \Psi(\Sigma(s_i)))$. □

**Definition 7.** An interpretation $M = (\Psi, \Sigma, \Xi)$ is said to be a model of an active database description $D$, if the following conditions are satisfied.

- $\Psi$ is a causal model of $D$.
- All fluent facts in $D$ are true in $M$.
- All user dictated occurrences are true in $M$.
- $\Xi$ satisfies the event definition in $D$.

[8] In this part and part 2 of the definition, when $j = 1$, $Res(\alpha_0 \circ \ldots \circ \alpha_{j-2}, \sigma)$ denotes $\sigma$.

and $\Sigma(s_{now})$ is the minimal sequence that satisfies the above four conditions. $\square$

$\Sigma(s_{now})$ will be called the *actual path* of $M$ and for simplicity will often be denoted by $\Sigma_{now}$.

The *query language* associated with $\mathcal{L}_{active}$ will consist of hypothetical facts of the form

$$f\ \textbf{after}\ \ \alpha\ \textbf{ordered\_at}\ \ s \tag{9}$$

and will be denoted by $\mathcal{L}_0^Q$.

When $s$ in (9) is $s_{now}$, we write the query as:

$$f\ \textbf{after}\ \ \alpha \tag{10}$$

If $\alpha$ is an empty sequence of actions we write the query as:

$$\textbf{currently}\ f \tag{11}$$

**Definition 8.** We say a query $q$ of the form (9) [9] is true in a model $M$ of an active database description $D$ if $f$ is true in $\Psi(\Sigma(s) \circ \alpha \circ \mathcal{T}(\alpha, \Psi(\Sigma(s))))$. $\square$

The following definition describes the set of acceptable conclusions one can obtain from an active database description $D$.

**Definition 9.** An active database description $D$ *entails* a query $q$ (written as $D \models q$) iff $q$ is true in all models of $D$. The set of all facts entailed by $D$ will be denoted by $Cn(D)$.

We will say that the answer given by $D$ to a query $q$ is *yes*, if $D \models q$; *no*, if $D \models \neg q$; and *unknown* otherwise. $\square$

Queries of the form (11) allow us to reason about what is true (and what is false) in the current situation of the database. Notice that this situation might be different in different models of the database description because of the non-determinism created by the active rules. Queries of the form (10) allow us to

[9] We can extend the queries to allow a sequence of user dictated action sequences in the standard way. Syntactically, (9) will then be extended to the form:

$f\ \textbf{after}\ \ \alpha_1, \ldots, \alpha_n\ \textbf{ordered\_at}\ \ s$

do hypothetical reasoning from the current situation and queries of the form (9) allow us to do counter-factual reasoning from the past situations. We may want to know properties of the database if we were to execute the sequence of actions $\alpha$ in situation $s$ instead of the sequence of actions in the user-dictated facts that were specified in the database description. Notice too, that if $\alpha$ is the empty sequence we are asking for properties of the database in real situations that occurred in the past.

## 4 A Translation to Logic Programs

In this section we present a translation $\Pi_s$ of active database descriptions into logic programs with situation calculus notation. For simplicity, we currently assume that all actions are executable in all situations and only try to capture what is true in the current situation. In the sequel we will remove these restrictions.

The logic program $\Pi_s D$ where $D$ is the active database description uses terms of five sorts. (We follow the convention of variables starting with capital letters and constants starting with small letters.)

- *fluents*: Fluent constants are denoted by $f, f'$, and fluent variables are denoted by $F, F', \ldots$.
- *events*: Event constants are denoted by $e, e'$, event variables are denoted by $E, E'$ and by $not(e)$ we denote the negative event of $e$.
- *actions*: Action constants are denoted by $a$, $a_0$, $a_1$, etc, and action variables with $A$, $A'$.
- *situations*: $s_0, \ldots, s_n$ are situation constants, and all other situation constants are constructed by application of the function symbol $res$ to a situation constant and an action. For example, if $a$ denotes an action and $s$ denotes a situation constant then $res(a, s)$ denotes a situation constant. Intuitively the situation constant $res(a, s)$ refers to the situation obtained by executing action $a$ in situation $s$. For convenience, by $[a_1, \ldots, a_m]s_0$ we denote the situation $res(a_m, res(a_{m-1}, \ldots, res(a_1, s_0)) \ldots)$. We use the reversed version of the Prolog's notation for lists; i.e. we have $[a_1, \ldots, a_m]$ to be equivalent to $[[a_1, \ldots, a_{m-1}]|a_m]$.
- *rules*: Rule constants are denoted by $r, r'$, and rule variables are denoted by $R, R', \ldots$

The program $\Pi_s D$ consists of the translations of the individual active rules, the initial state, the events occurring at the initial state, and certain other logic programming rules. The various predicates that are used in the translation have the following intuitive meaning.

- *actual*$(s)$: Intuitively, *actual*$(s)$ is true if $s$ is an actual situation where the sequence of actions in $s$ can either be ordered by a user or triggered by an active rule.

- *holds*$(f, s)$ and *hyp_holds*$(f, s)$: Intuitively, *hyp_holds*$(f, s)$ is true if $f$ holds in the situation (hypothetical or actual) $s$, and *holds*$(f, s)$ is true if $f$ holds in the actual situation $s$.

- We denote a sequence of actions by $\alpha$, $\beta$ and $\gamma$. Hence, $[\alpha]$, denotes a list of actions. We denote a sequence of lists of actions by $\Gamma$. For example, $\Gamma$ could be the sequence $[\alpha_1], \ldots, [\alpha_k]$. The list $[\Gamma]$ is a list of list of actions.

- *flatten*, *append*: These are auxiliary procedures to manipulate sequence of actions. Intuitively, *flatten*$([\Gamma], [\alpha])$ is true if $[\alpha]$ is a flattened list obtained from flattening $[\Gamma]$, which is a list of list of actions. Intuitively, *append*$([\alpha], [\beta], [\gamma])$ means that the list $[\gamma]$ is obtained by appending the list $[\beta]$ to the list $[\alpha]$. (To save space we do not explicitly give rules defining these predicates in our translation.)

- *unfold*: Intuitively, *unfold*$(s, [\Gamma], [\alpha], [\beta]s_0)$ is true if $[\beta]$ is the list of actions from the initial state that corresponds to the situation $s$ followed by the flattening of $[\Gamma]$, followed by $[\alpha]$.

- *event*$(e, [\beta], s)$: Intuitively, *event*$(e, [\beta], s)$ means that the event $e$ is induced by the list of actions $[\beta]$ in situation $s$. The rules that define *event* are given in Step 5.

- *event_pt*, *trigger_pt*: Recall the definition of a trigger interpretation $\mathcal{T}$ satisfying active rules in Definition 4. There we had $\mathcal{T}(\alpha, \sigma) = \beta_{1,1}, \ldots, \beta_{1,k_1}, \ldots, \beta_{m,1}, \ldots, \beta_{m,k_m}$. In this sequence, events are generated after the execution of $\alpha$ and after the execution of $\beta_{i,k_i}$, for $i \geq 1$. However, active rules can be triggered after any $\beta_{i,j}$ or after $\alpha$. Suppose we have a situation $s$ whose corresponding state is $\sigma$, and let $\alpha = a_1, \ldots, a_n$.

  The predicate *event_pt* describes situations where the active database computes events that have happened since the last such computation. In the above example such points are described by the fact

  *event_pt*$(s, [[a_1, \ldots, a_n]])$, and the facts

  *event_pt*$(s, [[a_1, \ldots, a_n], [\beta_{1,1}, \ldots, \beta_{1,k_1}], \ldots, [\beta_{i,1}, \ldots, \beta_{i,k_i}]])$, where, $1 \leq i \leq m$.

  The predicate *trigger_pt* describes situations where the active database triggers. In the above example such points are described by the fact

  *trigger_pt*$(s, [[a_1, \ldots, a_n]], [])$, the facts

  *trigger_pt*$(s, [[a_1, \ldots, a_n]], [\beta_{1,1}, \ldots, \beta_{1,j_1}])$, where $1 \leq j \leq k_1$, and the facts

$trigger\_pt(s, [[a_1, \ldots, a_n], [\beta_{1,1}, \ldots, \beta_{1,k_1}], \ldots, [\beta_{i,1}, \ldots, \beta_{i,k_i}]],$
$[\beta_{i+1,1}, \ldots, \beta_{i+1,j}])$,
where, $1 \leq i \leq m$ and $1 \leq j \leq k_{i+1}$.

- $p\_occurs(r, s, [\Gamma], [\alpha])$ and $occurs(r, s, [\Gamma], [\alpha])$: Intuitively, the predicate $p\_occurs(r, s, [\Gamma], [\alpha])$ is true if the rule $r$ can be fired in the situation $s$ followed by the flattening of $[\Gamma]$ followed by $[\alpha]$. But since there may be many such rules that can fired in this situation, and the semantics of active database system dictates that only one such rule should be actually fired,[10] we denote the firing of this particular rule by the atom $occurs(r, s, [\Gamma], [\alpha])$.
- $ab(f, a, s)$: Intuitively, $ab(f, a, s)$ is true if the execution of action $a$ in situation $s$ may change the value of $f$.
- $last(s), currently(f)$: Intuitively, $last(s)$ is true if $s$ is the situation where the active database system *terminates*, and $currently(f)$ is true if $f$ is true currently, i.e., at the end of the execution of all actions triggered by the active rules.

We now present the translation $\Pi_s D$.

**1. Translating active rules**:

For an active rule of the form (5) the program $\Pi_s D$ contains the following three rules:

$$(1.1)\ p\_occurs(\mathbf{r}, S, [\Gamma|[\beta]], []) \leftarrow event\_pt(S, [\Gamma|[\beta]]), unfold(S, [\Gamma], [], [\gamma]s_0),$$
$$event(e, [\beta], [\gamma]s_0), unfold(S, [\Gamma|[\beta]], [], [\gamma']s_0),$$
$$holds(p_1, [\gamma']s_0), \ldots, holds(p_n, [\gamma']s_0)$$

$$(1.2)\ mode(\mathbf{r}, m) \leftarrow$$

$$(1.3)\ trigger(\mathbf{r}, \alpha) \leftarrow$$

**2. Translating Fluent facts**:

If $f$ is true in the initial state then $\Pi_s D$ contains the following rule.

$$(2.2)\ hyp\_holds(f, s_0) \leftarrow$$

Otherwise $\Pi_s D$ contains the rule:

$$(2.3)\ hyp\_holds(\neg f, s_0) \leftarrow$$

[10] Similar to actions, in this paper we assume that there is no concurrent executions of rules.

**3. Translating user dictated occurrences:**

For an user dictate occurrences of the form (4) the program contains the following rule:

(3) $event\_pt(s, [[\alpha]]) \leftarrow$

**4. Translating causal rules:**

For each causal rule of the form (1) we have the following rules:

(4.1) $hyp\_holds(f, res(a, S)) \leftarrow hyp\_holds(p_1, S), \ldots, hyp\_holds(p_n, S)$

(4.2) $ab(f', a, S) \leftarrow hyp\_holds(p_1, S), \ldots, hyp\_holds(p_n, S)$

where $f$ is the fluent $g$ if $f = \neg g$, or the fluent literal $\neg g$ if $f = g$.

**5. Translating event definitions**

For each event definition of the form (7) we have the following rule:

$$\begin{aligned}(5.1) event(e, [\alpha|a], [\gamma]s_0) \leftarrow\ & hyp\_holds(q_1, [\beta]s_0), \ldots,\\ & hyp\_holds(q_m, [\beta]s_0),\\ & event(e_1, \alpha, [\beta]s_0), \ldots, event(e_n, \alpha, [\beta]s_0),\\ & append([\gamma], [\alpha], [\beta])\end{aligned}$$

To define the truth of event literals we have the following additional rule:

$event(not(e), [\alpha], S) \leftarrow not\ event(e, [\alpha], S)$

**6. Inertia axiom**

(6.1) $hyp\_holds(F, res(A, S)) \leftarrow hyp\_holds(F, S), not\ ab(F, A, S)$

**7. Other Rules**:

The other rules in $\Pi_s D$ are:

(7.1) Additional rules defining $p\_occurs$:

$$\begin{aligned}(7.1.1)\ p\_occurs(R, S, [\Gamma], [\gamma]) \leftarrow\ & p\_occurs(R, S, [\Gamma], [\alpha]),\\ & occurs(R', S, [\Gamma], [\alpha]), trigger(R', [\beta]),\\ & not\ ab_p(R, S, [\Gamma], [\alpha], [\beta]), append([\alpha], [\beta], [\gamma])\end{aligned}$$

(7.1.2) $ab_p(R, S, [\Gamma], [\alpha], [\beta]) \leftarrow occurs(R, S, [\Gamma], [\alpha]), trigger(R, [\beta])$

(7.1.3) $ab_p(R, S, [\Gamma], [\alpha], [\beta]) \leftarrow not\ mode(R, reconsider)$

These three rules take care of the pre-selection of *reconsider* rules. *p_occurs* in (7.1.1) marks the rule $R$ as a potential rule to be triggered if it was already pre-selected by $p_occurs$ in a previous firing, but it has not been used or it is not a rule in *ignore* mode. These two conditions are verified by (7.1.2) and (7.1.3).

(7.2) Rules defining *occurs*:

(7.2.1) $occurs(R, S, [\Gamma], [\alpha]) \leftarrow p\_occurs(R, S, [\Gamma], [\alpha])$
$not\ notoccurs(R, S, [\Gamma], [\alpha])$

(7.2.2) $notoccurs(R, S, [\Gamma], [\alpha]) \leftarrow occurs(R', S, [\Gamma], [\alpha]), neq(R, R')$

The above two rules implement the *choice* operator which was first introduced in [SZ90]. Intuitively, they guarantee that among all the rules that may possibly be fired in a situation $S$, only one gets fired in each stable model of the program and there is one stable model corresponding to each of the rules that may be possibly fired.[11]

(7.3) Rules defining *trigger_pt* and *event_pt*:

(7.3.1) $trigger\_pt(S, [\Gamma], []) \leftarrow event\_pt(S, [\Gamma])$

(7.3.2) $trigger\_pt(S, [\Gamma], [\gamma]) \leftarrow trigger\_pt(S, [\Gamma], [\alpha']), occurs(R, S, [\Gamma], [\alpha']),$
$trigger(R, [\alpha]), append([\alpha'], [\alpha], [\gamma])$

(7.3.3) $exists(S, [\Gamma], [\beta]) \leftarrow p\_occurs(R, S, [\Gamma], [\beta])$

(7.3.4) $event\_pt(S, [\Gamma|[\beta]]) \leftarrow trigger\_pt(S, [\Gamma], [\beta]),$
$not\ exists(S, [\Gamma], [\beta])$

(7.4) Rules defining intermediate termination of triggering:

(7.4.1) $terminates(S, [\Gamma]) \leftarrow event\_pt(S, [\Gamma]), not\ exists(S, [\Gamma], [])$

(7.5) Defining *actual* and *holds*:

(7.5.1) $actual(s_0) \leftarrow$

(7.5.2) $actual([\gamma]s_0) \leftarrow trigger\_pt(S, [\Gamma], [\alpha]), unfold(S, [\Gamma], [\alpha], [\gamma]s_0)$

[11] $neq(X, Y)$ checks that $X$ and $Y$ do not unify.

(7.5.3) $holds(F, [\gamma]s_0) \leftarrow hyp\_holds(F, [\gamma]s_0), actual([\gamma]s_0)$

The above rules make a fluent literal to hold in an actual situation if it hypothetical holds in that situation.

(7.6) Rules defining $unfold$:

(7.6.1) $unfold(s_0, [], [], s_0) \leftarrow$

(7.6.2) $unfold(s_{i+1}, [], [], [\gamma]s_0) \leftarrow unfold(S_i, [], [], [\alpha]s_0), terminate(s_i, [\Gamma]),$
$flatten([\Gamma], [\beta]), append([\alpha], [\beta], [\gamma])$

(7.6.3) $unfold(S, [\Gamma], [\alpha], [\gamma]s_0) \leftarrow unfold(S, [], [], [\beta_1]s_0),$
$flatten([\Gamma|[\alpha]], \beta_2), append([\beta_1], [\beta_2], [\gamma])$

(7.7) Rules defining $last\_sit$ and $last$:

(7.7.1) $not\_last\_sit(S) \leftarrow event\_pt(S', [\Gamma])$

(7.7.2) $last\_sit(S) \leftarrow not\ not\_last\_sit(S)$

(7.7.3) $last([\gamma]s_0) \leftarrow last\_sit(S), terminates(S, [\Gamma]),$
$unfold(S, [\Gamma], [], [\gamma]s_0)$

The above two rules define the last situation.

(7.8) Defining $currently$

(7.8.1) $currently(F) \leftarrow last([\gamma]s_0), holds(F, [\gamma]s_0)$

The above rule defines which fluent literals are true after the active database system terminates.
**End of the Translation**

Notice that the programs generated by the translation are not locally stratified because of the rules in (7.2). However, these rules follow the construction pattern of the choice operator introduced in [SZ90] which has a well-understood semantics based on the stable models of Gelfond and Lifschitz. Thus, $\Pi_s D$ will always have stable models, and it is possible to show a direct correspondence between the models of $D$ and the stable models of $\Pi_s D$. This correspondence allows us to show the following proposition.

**Proposition 10.** *For an active database description $D$,*

$$D \models \textbf{currently}\ f \textit{ iff } \Pi_s D \models currently(f).$$

### 4.1 Incorporating priorities

In the characterization of active database systems in the past section we did not prioritize between the active rules that are simultaneously triggered. We gave equal priority to all of them. However, many systems allow prioritizing between these rules. Several different prioritizing strategies (also referred to as conflict resolution strategies), such as priority, specificity, recency, etc. are used. We now briefly sketch how these strategies can be incorporated in the logic programming translations.

Consider a predicate $preferable$ of arity two that defines a preference relation between two rules. For example, $preferable(R1, R2)$ means that the rule $R1$ is preferable to rule $R2$. At this point we do not worry about how the relation $preferable$ is obtained. (It may be directly given or it may be defined in terms of other conflict resolution parameters such as how recently the rules were used.) We assume that the relation given by $preferable$ is a partial order.

Using the relation $preferable$ we can now define a predicate $pref\_occurs$ as follows:

$$\begin{aligned} not\_pref\_occurs(R, S, [\Gamma], [\alpha]) \leftarrow\ & p\_occurs(R', S, [\Gamma], [\alpha]), \\ & preferable(R', R), neq(R, R') \end{aligned}$$

$$\begin{aligned} pref\_occurs(R, S, [\Gamma], [\alpha]) \leftarrow\ & p\_occurs(R, S, [\Gamma], [\alpha]), \\ & not\ not\_pref\_occurs(R, S, [\Gamma], [\alpha]) \end{aligned}$$

The predicate $occurs$ is now defined in terms of $pref\_occurs$ instead of $p\_occurs$ in the following way:

$$occurs(R, S, [\Gamma], [\alpha]) \leftarrow pref\_occurs(R, S, [\Gamma], [\alpha]), not\ not\_occurs(R, S, [\Gamma], [\alpha])$$

$$not\_occurs(R, S, [\Gamma], [\alpha]) \leftarrow occurs(R', S, [\Gamma], [\alpha]), neq(R, R')$$

## 5 Final remarks and future work

In this paper we have presented a formal framework to describe active databases. Our framework allows the representation of actions and their effects, event definitions, active rules with different execution modes, and reasoning about the value of the database in both actual and hypothetical situations. We intend to substantially add to our framework. In particular we plan to

- extend $\mathcal{L}_{active}$ to allow specification of priorities between rules,

– consider additional execution modes for the active rules,

– allow concurrent and complex actions where a complex action is defined in terms of simple actions,

– allow temporal conditions, and

– compare the use of temporal conditions vs. complex events.

We have some ideas of how to do this based on previous work on concurrent and complex actions [BG96, LLL+94, BK93], and temporal conditions [MLLB96, SW95]. In general, we believe that the methodology of using a transition system or automata is a promising approach to formally describe the semantics of active databases.

## References


[Bar95] C. Baral. Reasoning about Actions : Non-deterministic effects, Constraints and Qualification. In *Proc. of IJCAI 95*, pages 2017–2023, 1995.

[BG96] C. Baral and M. Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming (to appear)*, 1996.

[BGP96] C. Baral, M. Gelfond, and A. Provetti. Representing Actions: Laws, Observations and Hypothesis. *Journal of Logic Programming (to appear)*, 1996.

[BK93] A. Bonner and M. Kifer. Transaction logic programming. In D. S. Warren, editor, *Logic Programming: Proc. of the 10th International Conf.*, pages 257–279, 1993.

[CFPB96] S. Ceri, P. Fraternali, S. Paraboschi, and L. Branca. Active rule management in Chimera. In J. Widom and S Ceri, editors, *Active Database Systems*, pages 151–176. Morgan Kaufmann, 1996.

[DBC96] U. Dayal, A. Buchmann, and S. Chakravarthy. The HiPAC Project. In J. Widom and S Ceri, editors, *Active Database Systems*, pages 177–206. Morgan Kaufmann, 1996.

[FMT94] P. Fraternali, D. Montesi, and L. Tanca. Active database semantics. In *Proc. of the Fifth Australasian Database Conference*, 1994.

[GJ96] N. Gehani and H. Jagadish. Active database facilities in Ode . In J. Widom and S Ceri, editors, *Active Database Systems*, pages 207–232. Morgan Kaufmann, 1996.

[Han96] E. Hanson. The Ariel Project. In J. Widom and S Ceri, editors, *Active Database Systems*, pages 63–86. Morgan Kaufmann, 1996.

[KL94] G. Kartha and V. Lifschitz. Actions with indirect effects: Preliminary report. In *KR 94*, pages 341–350, 1994.

[LLL+94] Y. Lesperance, H. Levesque, F. Lin, D. Marcu, R. Reiter, and R. Scherl. A logical approach to high level robot programming – a progress report. In *Working notes of the 1994 AAAI fall symposium on Control of the Physical World by Intelligent Systems (to appear), New Orleans, LA*, November 1994.

[LR94] F. Lin and R. Reiter. State constraints revisited. *Journal of Logic and Computation*, 4(5):655–678, October 1994.

[MLLB96] G. Mendez, J Llopis, J. Lobo, and C. Baral. Temporal logic and reasoning about actions. In *Common Sense 96*, 1996.

[MW88] S. Manchanda and D. S. Warren. A logic-based language for database updates. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 363–394. Morgan Kaufmann, 1988.

[PS96] S. Potamianos and M. Stonebraker. The POSTGRESS rule system. In J. Widom and S Ceri, editors, *Active Database Systems*, pages 43–62. Morgan Kaufmann, 1996.

[Rei94] R. Reiter. On specifying database updates. *Journal of Logic Programming*, 19,20:1–39, 1994.

[SK96] E. Simon and J. Kiernan. The A-RDL system. In J. Widom and S Ceri, editors, *Active Database Systems*, pages 111–150. Morgan Kaufmann, 1996.

[SW95] P. Sistla and O. Wolfson. Temporal conditions and integrity constraint checking in active database systems. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, San Jose, CA, 1995. ACM Press.

[SZ90] D. Sacca and C. Zaniolo. Stable models and non-determinism in logic programs with negation. In *Proceedings of PODS 1990*, pages 205–217, 1990.

[WC96] J. Widom and S Ceri, editors. *Active Database Systems - Triggers and Rules for advanced database processing*. Morgan Kaufmann, 1996.

[Wid92] J. Widom. A denotational semantics for starburst production rule language. *SIGMOD Record*, 21(3):4–9, 1992.

[Wid96] J. Widom. The Starburst rule system. In J. Widom and S Ceri, editors, *Active Database Systems*, pages 87–110. Morgan Kaufmann, 1996.

[Zan93] C. Zaniolo. A unified semantics for active and deductive databases. In *Proceedings of 1st international workshop on rules in database systems*, pages 271–287. Springer-Verlag, 1993.

[Zan96] C. Zaniolo. Active database rules with transaction-conscious stable models semantics. In *Proceedings of DOOD 1996*, pages 55–72, 1996.