Dynamics of Active Database Rules: Models and Refinements

Carlo Zaniolo
Computer Science Department
University of California
Los Angeles, CA 90024
zaniolo@cs.ucla.edu

Abstract Semantics represents a major problem area for active databases inasmuch as (i) there is no formal framework for defining the abstract semantics of active rules, and (ii) the various systems developed so far have ad-hoc operational semantics that are widely different from each other. This situation contributes to the difficulty of predicting the run-time behavior of sets of rules: thus, ensuring the termination of a given set of rules is currently recognized as a major research issue. This situation hampers the applicability of this powerful technology in critical application areas.

In this paper, we introduce a *durable change* semantics for active database rules; this semantics improves Starburst's *deferred activation* notion with concepts taken from Postgres and Heraclitus and the semantic foundations of deductive databases. We provide a formal logic-based model for this transaction-oriented semantics, show that it is amenable to efficient implementation, and prove that it solves the non-termination problem.

1 Introduction

Several active database languages and systems have been developed so far — a very incomplete list include [?, ?, ?]. Furthermore, active rules are now becoming a part of several commercial databases and of the SQL3 proposed standards. Indeed, active databases represent a powerful new technology that finds important application in the market place. However, this new technology is faced with several technical challenges; among these the lack of uniform and clear semantics stands out as one of the most and pressing and difficult problems [?]. The lack of formal models for characterizing the abstract semantics of active systems is a first facet of this problem. The second facet is represented by the differences between the many operational semantics proposed and implemented by the various systems in an ad-hoc fashion, with little progress towards unification and convergence. The result is that the behavior of complex rule sets is very difficult to predict, and critical questions such as confluence and termination are extremely hard to answer. These questions must be answered before active rules can be trusted with critical functions in an information system. Improved semantics and formal models for active databases, will solve these problem, and foster generalizations and simplifications that enhance the implementability, generality and intuitive appeal of active rules. The results presented in this paper illustrate the feasibility of achieving many of the desiderata of such an ideal scenario.

Let us consider the typical ECA rules of active databases:

$$\text{Event, Condition} \rightarrow \text{Action}$$

The basic structure of these rules make them different from those used in in expert systems shells such as CLIPS and OPS5 (which follow a Condition → Action pattern) or those used by deductive databases (that follow a Condition → Condition pattern). But in addition to these differences, active rules are also unique inasmuch as their meaning is intertwined with the concept of database transactions. Therefore, an active database system must specify, whether the Action part of the rule is to fired in the same transaction as the Event (*coupled semantics*) or as a separate transaction (*decoupled semantics*) [?]. Most systems adopt the coupled semantics, inasmuch as this is more effective at enforcing integrity constraints via active rules, and this framework will be adopted in this papers as well. Furthermore, while in the *immediate* interpretation of coupled semantics, rules are fired as soon as Event is detected, the *deferred* semantics used in Starburst is more transaction-conscious [?], inasmuch as it takes into account the fact that individual actions might be of no consequence before the transaction commits. For instance, the insertion of a record r, followed by the deletion of the same r within a transaction is *ephemeral*, inasmuch as the net effect of these two opposite actions (i.e., their composition) is null. Inasmuch as ephemeral actions leave no trace once the transaction completes, they should be disregarded by transaction-conscious ECA rules, which should instead be triggered only by actions that are durable, i.e., persist till the end of the transaction.

Thus, a critical contribution of Starburst is the notion that rules should be deferred until a *rule processing point*, which, by default, occurs at the end of the transaction, after regular actions have completed but before the transaction commits. At rule processing point, the net effect of all accumulated actions is computed using *composition* rules; for instance the net effect of an insert followed by an update on the same tuple is a modified insert [?].

While the basic idea of deferred semantics is both elegant and profound, there is the complication that many competing rules are firable at once at rule processing point. At this time, the many changes requested by the transaction (often on several relations) could trigger several rules with incompatible action requests in their heads. The Starburst designers recognized the complexity of the situation, and the fact that different firing orders might lead to different results or to non-terminating programs [?]. Their proposed solution calls for a very smart designer, who after studying the rules to ensure termination steers the system clear of problems through explicit assignment of inter-rule precedence. This approach has several drawbacks, including the fact that termination analysis is exceedingly difficult, and that rules cannot be added or deleted independent of each other. Therefore, in this paper we take deferred semantics a step further and show that, with a simple extension, the system can solve the rule termination and priority-assignment problems for the designer.

2 An Example

Consider the following example. We have three relations:

```
Dept (D#, DName, Div, Loc)
EMP(E#, Ename, JobTitle, SAL, Dept)
HPaid(JobTitle)
```

`HPaid` is actually a derived relation, which stores those job titles for which there are, or have been employees, who make more than $100,000. This concrete view is maintained via the rules EMP_INSERT and EMP_UPDATE specified as follows:

> Rules EMP_INSERT and EMP_UPDATE. *Upon an insertion into* `EMP` *or an update to* `EMP`*, the new* `SAL` *is checked, and if it exceeds $100,000, then the* `JobTitle` *of this employee is added to* `HPaid`*, assuming that it was not there already.*

There is also a foreign key constraint between `EMP` and `Dept`. This is supported by a rule DEPT_DELETE that propagates deletions from `Dept` to `EMP` :

> Rule DEPT_DELETE: *When a tuple is deleted from* `Dept`*, then delete all employees who were working in the deleted department.*

Now assume that our transaction has executed the following actions (in the order listed):

- Change the location of every department who was in LA (Los Angeles) into SM (Santa Monica).
- Delete the department with `D#` = 1300 from the database.
- Give a raise of $4,000 to all employees whose `Job_Title` is `analyst`.

Say that in the initial database there are only two departments with location 'LA', say, one with `D#` = `1300` and the other with `D#` = `2500`. Then, the Starburst *composition* semantics prescribes that the update on the `Dept` tuple with `D# =1300` followed by the deletion of the same is equivalent to the deletion the original tuple. The update on the department tuple with `D#` = `2500` instead remains till the end of the transaction . Thus, the update on the tuple with `D# =1300`, which does not persist until the end of the transaction, will be called *ephemeral* while the second will be called *durable*. Therefore, at rule processing point only two change remain and the following two rules can be activated:

- Rule DEPT_DELETE can be triggered by the resulting deletion of department with `D#` = `1300` (and `Loc` = '`LA`').
- Rule EMP_UPDATE can be triggered by the +$4,000 salary update gotten by analysts.

The issue of which of the two rules above should be fired first is left by Starburst up to programmer, who can direct the system by assigning explicit precedence to the rules. However, consider the situation in which EMP_UPDATE is fired before, or even at the same time as, DEPT_DELETE. Say that there is only one network specialist, Bob White who now makes $98,000. Then, with the $4,000 raise the new salary exceeds the $100,000 threshold and rule EMP_UPDATE might add a new entry into `HPaid`. However, if Bob White happens to work for department 1300, then there is a problem. Once the rule DEPT_DELETE fires the Bob White tuple is deleted and the $4000 salary-raise becomes ephemeral, and, therefore, the addition of `analyst` to `HPaid` becomes totally unjustified— it should never have happened.

Therefore, we propose a semantics whereby only durable-change events can fire rules—ephemeral-change events cannot. This restriction produces a natural ordering between rules; in our case, the DEPT_DELETE rule must be fired before EMP_UPDATE. Once the rules are processed in this order the update

Figure 1: Three entries in the delta tables at rule-processing point

```
updDept⁰(2500, ims, 1000, 'LA',
         2500, ims, 1000, 'SM').
delDept⁰(1300, media, 1000, 'LA').
updEMP⁰ (E2309,'Bob White',analyst,98000,1300,
         e2309,'Bob White',analyst,102000,1300).
```

on of Bob White's record will be removed, and the second rule will not fire at all.

There is also an obvious implication upon the termination problem, since the composition rules have the following property: every event that is followed by a later event on the same tuple is ephemeral. Therefore, each durable event on a tuple t is the final event on t. Since only final events can trigger rules, the computation cannot fall into an infinite loop.

In the rest of the paper we give a logic-based formalization to these simple concepts.

3 Active Rules as Deductive Rules

We will now define a logic-based semantics for active rules with priority assignment. We use the notion of delta relations from Heraclitus [?], which we extend to handle updates along with inserts, deletes (the term "changes" will be used here to denote any of these three). Thus, for each relation $\mathtt{R(X)}$ in the database schema, where $\mathtt{X}$ denotes the attribute vector for $\mathtt{R}$, we also keep three additional relations (*delta relations*):

$$\mathtt{insR},\ \mathtt{delR},\ \mathtt{updR}$$

The delta relations $\mathtt{insR(X)}$ and $\mathtt{delR(X)}$ have the same attributes as the original $\mathtt{R(X)}$. However, $\mathtt{updR}$ has an old-value and a new-value for each attributes of $\mathtt{R}$, and can therefore be viewed according to the scheme $\mathtt{updR(Xold, Xnew)}$. A sample of the initial value delta relations just before the rule-processing point is shown in Figure 1. The same tuple cannot appear in more than one delta table of the same relation.

We will use Datalog$_{1S}$ to model the state changes occurring in the various relations [?, ?]. In Datalog$_{1S}$, tables and predicates are allowed to have an additional argument or column called the *stage argument*. The values in the stage argument are taken from the domain $0,\ 0+1,\ 0+1+1,\ ...$, i.e., the integers generated by using the postfix successor function $+1$; thus, the integer 3 is represented as $0+1+1+1$. Alternatively using the normal functional notation, the successor of $\mathtt{J}$ is denoted $\mathtt{s(J)}$—this notation is at the root of the name Datalog$_{1S}$. The merits of Datalog$_{1S}$ for modeling temporal and dynamic systems have been described in several papers [?, ?, ?]. Therefore, delta predicates with stage argument $\mathtt{J}$ have the form $\mathtt{insR(J,X)}$, $\mathtt{delR(J,X)}$ and $\mathtt{updR(J,Xold,Xnew)}$. For notational convenience, we will instead write the stage argument as a superscript: $\mathtt{insR^J(X)}$, $\mathtt{delR^J(X)}$ and $\mathtt{updR^J(Xold,Xnew)}$.

In addition to the delta relations, several auxiliary predicates are needed for each $\mathtt{R}$ in our database schema. In particular we need:

- *Initial Relation:* $\mathtt{iniR(X)}$. This stores the value of $\mathtt{R}$ at the beginning of the transaction. It does not have a stage argument since it remains constant during the whole transaction.
- *Delta Relations:* $\mathtt{insR^J(X)}$, $\mathtt{delR^J(X)}$, $\mathtt{updR^J(Xold,Xnew)}$.
- *Current Relations:* $\mathtt{curR^J(X)}$ represents the current content of relation $\mathtt{R}$, as seen within the transaction. It is computed from the initial relation and the delta relations.
- *Action Request Relations:* $\mathtt{rinR(X)}$, $\mathtt{rdeR^J(X)}$, $\mathtt{rupR^J(Xold,Xnew)}$. These contain the actions on $\mathtt{R}$ produced by fired active rules. Their union yields the *change request relation* $\mathtt{chrR^J(X)}$. The union of these for all R in the schema produces the request relation $\mathtt{req^J}$.
- *Durable-change Relations:* $\mathtt{dinR^J(X)}$, $\mathtt{ddeR^J(X)}$, $\mathtt{dupR^J(Xold,Xnew)}$. These contain all the changes assumed durable at step $\mathtt{J}$.
- *A Current Level relation:* $\mathtt{levl^J(Nr)}$ with $\mathtt{Nr}$ the name of a rule. This predicate is used to enforce priorities between rules by denoting the rules that are currently active. The priorities between rules is represented by a binary $\mathtt{prec}$ relation.

We begin by computing the current value of the database as shown in Figure 2, using frame axioms. The current value of relation for $\mathtt{R(X)}$ is obtained by first subtracting from its initial value $\mathtt{iniR}$ the tuples deleted and the old values of tuples updated, and then adding the tuples inserted and the new values of tuples updated (for both delta relations and durable-change relations):

$$\begin{array}{ll}
\mathtt{curR^J(X)} \leftarrow & \mathtt{iniR(X), levl^J(\_)}, \\
& \neg\mathtt{delR^J(X)}, \neg\mathtt{updR^J(X,New)}, \\
& \neg\mathtt{ddeR^J(X)}, \neg\mathtt{dupR^J(X,New)}. \\
\mathtt{curR^J(X)} \leftarrow & \mathtt{insR^J(X)}. \\
\mathtt{curR^J(X)} \leftarrow & \mathtt{updR^J(Old,X)}. \\
\mathtt{curR^J(X)} \leftarrow & \mathtt{dinR^J(X)}. \\
\mathtt{curR^J(X)} \leftarrow & \mathtt{dupR^J(Old,X)}.
\end{array}$$

The next set of rules, called *action request* rules, capture the behavior of the actual active rules in the system.

Figure 2: The current state of the database via frame axioms

$$
\begin{aligned}
\mathtt{curDept}^{\mathtt{J}}(\mathtt{D},\mathtt{N},\mathtt{Dv},\mathtt{L}) \leftarrow\ & \mathtt{iniDept}(\mathtt{D},\mathtt{N},\mathtt{DvL}), \mathtt{levl}^{\mathtt{J}}(\_),\\
& \neg\mathtt{delDept}^{\mathtt{J}}(\mathtt{D},\mathtt{N},\mathtt{Dv},\mathtt{L}), \neg\mathtt{updDept}^{\mathtt{J}}(\mathtt{D},\mathtt{N},\mathtt{Dv},\mathtt{L},\_,\_,\_,\_),\\
& \neg\mathtt{ddeDept}^{\mathtt{J}}(\mathtt{D},\mathtt{N},\mathtt{Dv},\mathtt{L}), \neg\mathtt{dupDept}^{\mathtt{J}}(\mathtt{D},\mathtt{N},\mathtt{Dv},\mathtt{L},\_,\_,\_,\_).\\
\mathtt{curDept}^{\mathtt{J}}(\mathtt{D},\mathtt{N},\mathtt{Dv},\mathtt{L}) \leftarrow\ & \mathtt{insDept}^{\mathtt{J}}(\mathtt{D},\mathtt{N},\mathtt{Dv},\mathtt{L}).\\
\mathtt{curDept}^{\mathtt{J}}(\mathtt{D},\mathtt{N},\mathtt{Dv},\mathtt{L}) \leftarrow\ & \mathtt{updDept}^{\mathtt{J}}(\_,\_,\_,\_,\mathtt{D},\mathtt{N},\mathtt{Dv},\mathtt{L}).\\
\mathtt{curDept}^{\mathtt{J}}(\mathtt{D},\mathtt{N},\mathtt{Dv},\mathtt{L}) \leftarrow\ & \mathtt{dinDept}^{\mathtt{J}}(\mathtt{D},\mathtt{N},\mathtt{Dv},\mathtt{L}).\\
\mathtt{curDept}^{\mathtt{J}}(\mathtt{D},\mathtt{N},\mathtt{Dv},\mathtt{L}) \leftarrow\ & \mathtt{dupDept}^{\mathtt{J}}(\_,\_,\_,\_,\mathtt{D},\mathtt{N},\mathtt{Dv},\mathtt{L}).
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{curEMP}^{\mathtt{J}}(\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno}) \leftarrow\ & \mathtt{iniEMP}(\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno}), \mathtt{levl}^{\mathtt{J}}(\_)\\
& \neg\mathtt{delEMP}^{\mathtt{J}}(\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno}), \neg\mathtt{updEMP}^{\mathtt{J}}(\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno},\_,\_,\_,\_),\\
& \neg\mathtt{ddeEMP}^{\mathtt{J}}(\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno}), \neg\mathtt{dupEMP}^{\mathtt{J}}(\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno},\_,\_,\_,\_).\\
\mathtt{curEMP}^{\mathtt{J}}(\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno}) \leftarrow\ & \mathtt{insEMP}^{\mathtt{J}}(\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno}).\\
\mathtt{curEMP}^{\mathtt{J}}(\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno}) \leftarrow\ & \mathtt{updEMP}^{\mathtt{J}}(\_,\_,\_,\_,\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno}).\\
\mathtt{curEMP}^{\mathtt{J}}(\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno}) \leftarrow\ & \mathtt{dinEMP}^{\mathtt{J}}(\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno}).\\
\mathtt{curEMP}^{\mathtt{J}}(\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno}) \leftarrow\ & \mathtt{dupEMP}^{\mathtt{J}}(\_,\_,\_,\_,\mathtt{Eno},\mathtt{N},\mathtt{Jt},\mathtt{Dno}).
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{curHPaid}^{\mathtt{J}}(\mathtt{Jt}) \leftarrow\ & \mathtt{iniHPaid}(\mathtt{Jt}), \mathtt{levl}^{\mathtt{J}}(\_), \neg\mathtt{delHPaid}^{\mathtt{J}}(\mathtt{Jt}), \neg\mathtt{updHPaid}^{\mathtt{J}}(\mathtt{Jt},\_).\\
& \neg\mathtt{ddeHPaid}^{\mathtt{J}}(\mathtt{Jt}), \neg\mathtt{dupHPaid}^{\mathtt{J}}(\mathtt{Jt},\_).\\
\mathtt{curHPaid}^{\mathtt{J}}(\mathtt{Jt}) \leftarrow\ & \mathtt{insHPaid}^{\mathtt{J}}(\mathtt{Jt}).\\
\mathtt{curHPaid}^{\mathtt{J}}(\mathtt{Jt}) \leftarrow\ & \mathtt{updHPaid}^{\mathtt{J}}(\_,\mathtt{Jt}).\\
\mathtt{curHPaid}^{\mathtt{J}}(\mathtt{Jt}) \leftarrow\ & \mathtt{dinHPaid}^{\mathtt{J}}(\mathtt{Jt}).\\
\mathtt{curHPaid}^{\mathtt{J}}(\mathtt{Jt}) \leftarrow\ & \mathtt{dupHPaid}^{\mathtt{J}}(\_,\mathtt{Jt}).
\end{aligned}
$$

Figure 3: Translations of Active Rules

DEPT_DELETE:

$$
\begin{aligned}
\mathtt{rdeEMP}^{\mathtt{J}}(\mathtt{En},\mathtt{E},\mathtt{JT},\mathtt{S},\mathtt{Dn}) \leftarrow\ & \mathtt{delDept}^{\mathtt{J}}(\mathtt{Dn},\mathtt{N},\mathtt{V},\mathtt{L}), \mathtt{curEMP}^{\mathtt{J}}(\mathtt{En},\mathtt{E},\mathtt{JT},\mathtt{S},\mathtt{Dn}), \mathtt{levl}^{\mathtt{J}}(\mathtt{dd}),\\
& \neg\mathtt{lchDept}^{\mathtt{J}}(\mathtt{Dn},\mathtt{N},\mathtt{V},\mathtt{L}).
\end{aligned}
$$

EMP_INSERT:

$$
\begin{aligned}
\mathtt{rinHPaid}^{\mathtt{J}}(\mathtt{Jt}) \leftarrow\ & \mathtt{insEMP}^{\mathtt{J}}(\mathtt{En},\mathtt{N},\mathtt{Jt},\mathtt{S},\mathtt{Dn}), \mathtt{S} > 100000, \neg\mathtt{curHPaid}^{\mathtt{J}}(\mathtt{Jt}), \mathtt{levl}^{\mathtt{J}}(\mathtt{ei}),\\
& \neg\mathtt{lchEMP}^{\mathtt{J}}(\mathtt{En},\mathtt{E},\mathtt{JT},\mathtt{S},\mathtt{Dn}).
\end{aligned}
$$

EMP_UPDATE:

$$
\begin{aligned}
\mathtt{rinHPaid}^{\mathtt{J}}(\mathtt{Jt}) \leftarrow\ & \mathtt{updEMP}^{\mathtt{J}}(\mathtt{En},\_,\mathtt{Jt},\mathtt{So},\_,\mathtt{En},\_,\mathtt{Jt},\mathtt{Sn},\_), \mathtt{Sn} > 100000, \neg\mathtt{curHPaid}^{\mathtt{J}}(\mathtt{Jt}), \mathtt{levl}^{\mathtt{J}}(\mathtt{eu}),\\
& \neg\mathtt{lchEMP}^{\mathtt{J}}(\mathtt{En},\mathtt{E},\mathtt{JT},\mathtt{S},\mathtt{Dn}).
\end{aligned}
$$

Figure 4: Changes assumed durable in firing the rules of Figure 3

DEPT_DELETE:

$$
\begin{aligned}
\mathtt{ddeDept}^{\mathtt{J}}(\mathtt{Dn},\mathtt{N},\mathtt{V},\mathtt{L}). \leftarrow\ & \mathtt{delDept}^{\mathtt{J}}(\mathtt{Dn},\mathtt{N},\mathtt{V},\mathtt{L}), \mathtt{curEMP}^{\mathtt{J}}(\mathtt{En},\mathtt{E},\mathtt{JT},\mathtt{S},\mathtt{Dn}), \mathtt{levl}^{\mathtt{J}}(\mathtt{dd}),\\
& \neg\mathtt{lchDept}^{\mathtt{J}}(\mathtt{Dn},\mathtt{N},\mathtt{V},\mathtt{L}).
\end{aligned}
$$

EMP_INSERT:

$$
\begin{aligned}
\mathtt{dinEMP}^{\mathtt{J}}(\mathtt{En},\mathtt{N},\mathtt{Jt},\mathtt{S},\mathtt{Dn}) \leftarrow\ & \mathtt{insEMP}^{\mathtt{J}}(\mathtt{En},\mathtt{N},\mathtt{Jt},\mathtt{S},\mathtt{Dn}), \mathtt{S} > 100000, \neg\mathtt{curHPaid}^{\mathtt{J}}(\mathtt{Jt}), \mathtt{levl}^{\mathtt{J}}(\mathtt{ei}),\\
& \neg\mathtt{lchEMP}^{\mathtt{J}}(\mathtt{En},\mathtt{E},\mathtt{JT},\mathtt{S},\mathtt{Dn}).
\end{aligned}
$$

EMP_UPDATE:

$$
\begin{aligned}
\mathtt{dupEMP}^{\mathtt{J}}(\mathtt{E},\mathtt{N},\mathtt{Jt},\mathtt{S},\mathtt{D},\mathtt{En},\mathtt{Nn},\mathtt{Jtn},\mathtt{Sn},\mathtt{Dn}) \leftarrow\ & \mathtt{updEMP}^{\mathtt{J}}(\mathtt{E},\mathtt{N},\mathtt{Jt},\mathtt{S},\mathtt{D},\mathtt{En},\mathtt{Nn},\mathtt{Jtn},\mathtt{Sn},\mathtt{Dn}), \mathtt{Sn} > 100000,\\
& \neg\mathtt{curHPaid}^{\mathtt{J}}(\mathtt{Jt}), \mathtt{levl}^{\mathtt{J}}(\mathtt{eu}), \neg\mathtt{lchEMP}^{\mathtt{J}}(\mathtt{En},\mathtt{E},\mathtt{JT},\mathtt{S},\mathtt{Dn}).
\end{aligned}
$$

Obviously events are represented by the tuples of the delta relations, while conditions must be evaluated against the current relations. Finally the actions in the head of active rules are modeled by action requests. For instance, an immediate translation of rule DEPT_DELETE is:

$$\begin{aligned}\mathtt{rdeEMP^{J+1}(En, E, JT, S, Dn)} \leftarrow\ & \mathtt{delDept^{J}(Dn, N, V, L)},\\ & \mathtt{curEMP^{J}(En, E, JT, S, Dn)}, \mathtt{levl^{J}(dd)}.\end{aligned}$$

This rule specifies that all the employees working in a certain department must be deleted once their department is in the delta relation. The rule will fire only if its proper level of precedence, i.e., only if $\mathtt{levl^{J}(dd)}$ is true, where $\mathtt{dd}$ is just a shorter name for DEPT_DELETE.

To express the durable-change semantics we need to add an additional goal $\neg\mathtt{lchDept}$ to ensure that the event triggering the rule is a durable one and will not be obliterated by later change requests (“later” refer to stage values larger than the current ones). Thus our original rule becomes:

$$\begin{aligned}\mathtt{rdeEMP^{J+1}(En, E, JT, S, Dn)} \leftarrow\ & \mathtt{delDept^{J}(Dn, N, V, L)},\\ & \mathtt{curEMP^{J}(En, E, JT, S, Dn)}, \mathtt{levl^{J}(dd)},\\ & \neg\mathtt{lchDept^{J}(Dn, N, V, L)}.\end{aligned}$$

This rule specifies that all the employees working in a certain department must also be deleted if their department is deleted. The translation of active rules for the example at hand is shown in Figure 3.

At each step, the firing of active rules might generate several action requests on $\mathtt{R}$. These have the form $\mathtt{rinR}$, $\mathtt{rdeR}$, $\mathtt{rupR}$, respectively for tuples inserted, deleted or updated. Thus we have three rules as follows:

$$\begin{aligned}\mathtt{chrR^{J}(X)} \leftarrow\ & \mathtt{rinR^{J}(X)}.\\ \mathtt{chrR^{J}(X)} \leftarrow\ & \mathtt{rdeR^{J}(X)}.\\ \mathtt{chrR^{J}(X)} \leftarrow\ & \mathtt{rupR^{J}(X, New)}.\end{aligned}$$

From these, we can now derive $\mathtt{lchR^{I}(X)}$ for values of $\mathtt{I}$ preceding the current stage value of $\mathtt{J}$. (Say that the $<$ relation between stage values is part of Datalog$_{1S}$, or alternatively that we define recursive rules to achieve the same effect.)

$$\mathtt{lchR^{I}(X)} \leftarrow\ \mathtt{chrR^{J}(X)}, \mathtt{I} < \mathtt{J}.$$

Now, we have to use the composition rules to compose the action requests with old deltas yielding new deltas (of course new and old deltas are denoted by their respective stage values of $\mathtt{J+1}$ and $\mathtt{J}$). Basically there are three cases:

1. The action request $\mathtt{rinR(X)}, \mathtt{rde(X)}, \mathtt{rup(X, \_)}$, does not compose with any object in the delta tables. In this case the action request is simply entered in the delta tables. Thus:

$$\begin{aligned}\mathtt{insR^{J+1}(X)} \leftarrow\ & \mathtt{rinR^{J}(X)},\ \neg\mathtt{insR^{J}(X)},\\ & \neg\mathtt{delR^{J}(X)}, \neg\mathtt{upd^{J}(\_, X)}.\\ \mathtt{delR^{J+1}(X)} \leftarrow\ & \mathtt{rdeR^{J}(X)}, \neg\mathtt{insR^{J}(X)},\\ & \neg\mathtt{delR^{J}(X)}, \neg\mathtt{upd^{J}(\_, X)}.\\ \mathtt{updR^{J+1}(X, Y)} \leftarrow\ & \mathtt{rupR^{J}(X, Y)}, \neg\mathtt{insR^{J}(X)},\\ & \neg\mathtt{delR^{J}(X)}, \neg\mathtt{upd^{J}(\_, X)}.\end{aligned}$$

2. The second case concerns delta tuples that are neither moved to durable-change tables nor affected by the last action requests. These tuples are simply copied into the next-state delta tables. We also have added a $\mathtt{wt4^{J}}$ predicate to ensure that these rules do not fire until the current change-requests have been computed:

$$\begin{aligned}\mathtt{insR^{J+1}(X)} \leftarrow\ & \mathtt{insR^{J}(X)},\ \mathtt{wt4^{J}},\\ & \neg\mathtt{dinR^{(}X)}, \neg\mathtt{chrR^{J}(X)}.\\ \mathtt{delR^{J+1}(X)} \leftarrow\ & \mathtt{delR^{J}(X)},\ \mathtt{wt4^{J}},\\ & \neg\mathtt{ddeR^{(}X)}, \neg\mathtt{chrR^{J}(X)}.\\ \mathtt{updR^{J+1}(X, Y)} \leftarrow\ & \mathtt{updR^{J}(X, Y)},\ \mathtt{wt4^{J}},\\ & \neg\mathtt{dupR^{(}X, Y)}, \neg\mathtt{chrR^{J}(X)}.\end{aligned}$$

3. This is the situation where an object in the delta tables at stage $\mathtt{J}$ must be composed with action requests to yield an entry in the delta table at stage $\mathtt{J+1}$. In this case we have to apply the composition rules as follows:

$$\begin{aligned}\mathtt{\%null} \leftarrow\ & \mathtt{insR^{J}(X)},\ \mathtt{rdeR^{J}(X)}.\\ \mathtt{insR^{J+1}(Xnew)} \leftarrow\ & \mathtt{insR^{J}(X)},\ \mathtt{rupR^{J}(X, Xnew)}.\\ \mathtt{error} \leftarrow\ & \mathtt{insR^{J}(X)},\ \mathtt{rinR^{J}(X)}.\\ \mathtt{error} \leftarrow\ & \mathtt{delR^{J}(X)},\ \mathtt{rdeR^{J}(X)}.\\ \mathtt{error} \leftarrow\ & \mathtt{delR^{J}(X)},\ \mathtt{rupR^{J}(X, Y)}.\\ \mathtt{\%null} \leftarrow\ & \mathtt{delR^{J}(X)},\ \mathtt{rinR^{J}(X)}.\\ \mathtt{delR^{J+1}(X)} \leftarrow\ & \mathtt{updR^{J}(X, Y)},\ \mathtt{rdeR^{J}(Y)}.\\ \mathtt{error} \leftarrow\ & \mathtt{updR^{J}(\_, X)},\ \mathtt{rinR^{J}(X)}.\\ \mathtt{updR^{J+1}(Xold, Xnew)} \leftarrow\ & \\ & \mathtt{updR^{J}(Xold, X)},\ \mathtt{rupR^{J}(X, Xnew)}.\end{aligned}$$

Figure 5: Change Requests and later changes

$$\begin{aligned}\mathtt{chrDept^{J}(En, E, JT, S, DN)} \leftarrow\ & \mathtt{rinDept^{J}(En, E, JT, S, DN)}.\\ \mathtt{chrDept^{J}(En, E, JT, S, DN)} \leftarrow\ & \mathtt{rdeDept^{J}(En, E, JT, S, DN)}.\\ \mathtt{chrDept^{J}(En, E, JT, S, DN)} \leftarrow\ & \mathtt{rupdDept^{J}(En, E, JT, S, DN, \_, \_, \_)}.\\ \mathtt{lchDept^{I}(En, E, JT, S, DN)} \leftarrow\ & \mathtt{chrDept^{J}(En, E, JT, S, DN)}, \mathtt{I} < \mathtt{J}.\end{aligned}$$

$$\begin{aligned}\mathtt{chrEMP^{J}(En, N, Jt, S, Dn)} \leftarrow\ & \mathtt{rinEMP^{I}(En, N, Jt, S, Dn)}.\\ \mathtt{chrEMP^{J}(En, N, Jt, S, Dn)} \leftarrow\ & \mathtt{rdeEMP^{I}(En, N, Jt, S, Dn)}.\\ \mathtt{chrEMP^{J}(En, N, Jt, S, Dn)} \leftarrow\ & \mathtt{rupEMP^{I}(En, N, Jt, Dn, S, \_, \_, \_)}.\\ \mathtt{lchEMP^{J}(En, N, Jt, S, Dn)} \leftarrow\ & \mathtt{chrEMP^{I}(En, N, Jt, S, Dn)}, \mathtt{I} < \mathtt{J}.\\ \mathtt{chrHPaid^{J}(Jt)} \leftarrow\ & \mathtt{rinHPaid^{I}(Jt)}.\\ \mathtt{chrHPaid^{J}(Jt)} \leftarrow\ & \mathtt{rdeHPaid^{I}(Jt)}.\\ \mathtt{chrHPaid^{J}(Jt)} \leftarrow\ & \mathtt{rupHPaid^{I}(Jt, \_)}.\\ \mathtt{lchHPaid^{J}(Jt)} \leftarrow\ & \mathtt{lchHPaid^{J}(Jt)}, \mathtt{I} < \mathtt{J}.\end{aligned}$$

Figure 7: Delta tuples copied into the next-state table without any change (Case 2)

$$\begin{array}{ll}
\texttt{insDept}^{\texttt{J}+1}(\texttt{D},\texttt{N},\texttt{Div},\texttt{L}) \leftarrow & \texttt{insDept}^{\texttt{J}}(\texttt{D},\texttt{N},\texttt{Div},\texttt{L}), \neg\texttt{dinDept}^{\texttt{J}}(\texttt{D},\texttt{N},\texttt{Div},\texttt{L}), \neg\texttt{chrDept}^{\texttt{J}}(\texttt{D},\texttt{N},\texttt{Div},\texttt{L}),\\
\texttt{delDept}^{\texttt{J}+1}(\texttt{D},\texttt{N},\texttt{Div},\texttt{L}) \leftarrow & \texttt{delDept}^{\texttt{J}}(\texttt{D},\texttt{N},\texttt{Div},\texttt{L}), \neg\texttt{ddeDept}^{\texttt{J}}(\texttt{D},\texttt{N},\texttt{Div},\texttt{L}), \neg\texttt{chrDept}^{\texttt{J}}(\texttt{D},\texttt{N},\texttt{Div},\texttt{L}),\\
\texttt{updDept}^{\texttt{J}+1}(\texttt{Do},\texttt{No},\texttt{Dvo},\texttt{Lo},\texttt{Dn},\texttt{Nn},\texttt{Dvn},\texttt{Ln}) \leftarrow & \texttt{updDept}^{\texttt{J}}(\texttt{Do},\texttt{No},\texttt{Dvo},\texttt{Lo},\texttt{Dn},\texttt{Nn},\texttt{Dvn},\texttt{Ln}),\\
& \neg\texttt{dupDept}^{\texttt{J}}(\texttt{Do},\texttt{No},\texttt{Dvo},\texttt{Lo},\texttt{Dn},\texttt{Nn},\texttt{Dvn},\texttt{Ln}), \neg\texttt{chrDept}^{\texttt{J}}(\texttt{D},\texttt{N},\texttt{Div},\texttt{L})
\end{array}$$

$$\begin{array}{ll}
\texttt{insEMP}^{\texttt{J}+1}(\texttt{En},\texttt{N},\texttt{Jt},\texttt{Dn}) \leftarrow & \texttt{insEMP}^{\texttt{J}}(\texttt{En},\texttt{N},\texttt{Jt},\texttt{Dn}), \neg\texttt{dinEMP}^{\texttt{J}}(\texttt{En},\texttt{N},\texttt{Jt},\texttt{Dn}), \neg\texttt{chrEMP}^{\texttt{J}}(\texttt{En},\texttt{N},\texttt{Jt},\texttt{Dn}),\\
\texttt{delEMP}^{\texttt{J}+1}(\texttt{D},\texttt{N},\texttt{Div},\texttt{L}) \leftarrow & \texttt{delEMP}^{\texttt{J}}(\texttt{D},\texttt{N},\texttt{Div},\texttt{L}), \neg\texttt{ddeEMP}^{\texttt{J}}(\texttt{En},\texttt{N},\texttt{Jt},\texttt{Dn}), \neg\texttt{chrEMP}^{\texttt{J}}(\texttt{En},\texttt{N},\texttt{Jt},\texttt{Dn}),\\
\texttt{updEMP}^{\texttt{J}+1}(\texttt{Do},\texttt{No},\texttt{Dvo},\texttt{Lo},\texttt{Dn},\texttt{Nn},\texttt{Dvn},\texttt{Ln}) \leftarrow & \texttt{updEMP}^{\texttt{J}}(\texttt{Do},\texttt{No},\texttt{Dvo},\texttt{Lo},\texttt{Dn},\texttt{Nn},\texttt{Dvn},\texttt{Ln}),\\
& \neg\texttt{dupEMP}^{\texttt{J}}(\texttt{Do},\texttt{No},\texttt{Dvo},\texttt{Lo},\texttt{Dn},\texttt{Nn},\texttt{Dvn},\texttt{Ln}), \neg\texttt{chrEMP}^{\texttt{J}}(\texttt{En},\texttt{N},\texttt{Jt},\texttt{Dn}).
\end{array}$$

$$\begin{array}{ll}
\texttt{insHPaid}^{\texttt{J}+1}(\texttt{Jt}) \leftarrow & \texttt{insDept}^{\texttt{J}}(\texttt{Jt}), \neg\texttt{dinHPaid}^{\texttt{J}}(\texttt{Jt}), \neg\texttt{chrHPaid}^{\texttt{J}}(\texttt{Jt}),\\
\texttt{delHPaid}^{\texttt{J}+1}(\texttt{Jt}) \leftarrow & \texttt{delDept}^{\texttt{J}}(\texttt{Jt}), \neg\texttt{ddeHPaid}^{\texttt{J}}(\texttt{Jt}), \neg\texttt{rdeHPaid}^{\texttt{J}}(\texttt{Jt}).\\
\texttt{updHPaid}^{\texttt{J}+1}(\texttt{Jto},\texttt{Jtn}) \leftarrow & \texttt{delDept}^{\texttt{J}}(\texttt{Jto},\texttt{Jtn}), \neg\texttt{dupHPaid}^{\texttt{J}}(\texttt{Jto},\texttt{Jtn}), \neg\texttt{chrHPaid}^{\texttt{J}}(\texttt{Jtn}).
\end{array}$$

The set of rules for updating the delta relations for the example at hand is shown if Figures 6, 7 and 8.

Finally, we have the prec table that describes the (inverse) priority between rules and ensures that only the rules at the correct precedence level will fire. An entry $\texttt{prec}(\texttt{r}_1, \texttt{r}_2)$ denotes that a rule at level $\texttt{r}_2$ should fire only after all rules at level $\texttt{r}_1$ have stopped firing (if $\texttt{r}_2$ is non-recursive, then it can only fire at one stage value; if $\texttt{r}_2$ is recursive, then it can fire at successive stage values while keeping at the same precedence level). Therefore, the first two rules in Figure 9 specify that, if there has been some action request we keep the same level; otherwise we move to the rules at the next precedence level. Naturally, $\texttt{req}^{\texttt{J}}$ is defined as the disjunction of all possible action requests. When we reach the last level in prec, for a stage value of say m, then, $\texttt{levl}^{\texttt{m+1}}$ is never set to true and we thus reached the end of the computation. The third rule in Figure 9 specifies that at the first step of the computation the precedence to be used to select the rules should be the first (bottom) in the prec tree.

For each database, $\mathcal{D}$, the program containing the rules so generated, augmented with the facts describing the content of the database at the beginning of the transaction, will be called the *durable-delta* program for $\mathcal{D}$.

4 Declarative Semantics and Operational Semantics

At this point, our reader is probably puzzled by the many rules that our Datalog$_{1S}$ model has generated starting from a rather simple example. It is therefore important to point out that most of these rules, namely all those in Figures 2, 6, 7, 8 and 9 are needed to express Starburst's deferred evaluation with composition semantics using Heraclitus' delta relation approach. Any complete formalization of these complex operations is bound to be a lengthy one. For instance, the usage of relational algebra or relational calculus would lead to even longer formulas—and in fact most topical papers only provide semi-formal English-based descriptions of these. The examples of active rules, normally expressed in SQL or QUEL-like syntax in such papers, however, find a very simple expression in our framework—Figure 3.

Finally, all rules but those that define action requests and durable changes, can be generated directly from the schema, and they obey highly repetitive patterns. Therefore the use of a meta-level notation, where variables represent relation names and their attribute lists, would cut down dramatically in the number of final rules generated. Nevertheless, we have restricted ourselves to the basic Datalog$_{1S}$ representation, because we want to apply the standard stable model semantics to the problem at hand.

Figure 9: Moving to the next level till no more

$$\begin{array}{ll}
\texttt{levl}^{\texttt{J+1}}(\texttt{X}) \leftarrow & \texttt{levl}^{\texttt{J}}(\texttt{X}), \texttt{wt1}^{\texttt{J+1}}, \neg\texttt{req}^{\texttt{J}}, \neg\texttt{error}. \\
\texttt{levl}^{\texttt{J+1}}(\texttt{Y}) \leftarrow & \texttt{levl}^{\texttt{J}}(\texttt{X}), \texttt{wt1}^{\texttt{J+1}},\ \texttt{req}^{\texttt{J}}, \neg\texttt{error}, \\
 & \texttt{prec}(\texttt{X}, \texttt{Y}). \\
\texttt{levl}^{0}(\texttt{X}) \leftarrow & \texttt{wt1}^{0}, \texttt{prec}(\texttt{nil}, \texttt{X}). \\
\texttt{req}^{\texttt{J}} \leftarrow & \texttt{chrDdep}^{\texttt{J}}(\_,\_,\_,\_). \\
\texttt{req}^{\texttt{J}} \leftarrow & \texttt{chrEMP}^{\texttt{J}}(\_,\_,\_,\_). \\
\texttt{req}^{\texttt{J}} \leftarrow & \texttt{chrHPaid}^{\texttt{J}}(\_). \\
\texttt{wt4}^{\texttt{J}} \leftarrow & \texttt{wt3}^{\texttt{J}}.\ \texttt{\%same strtm as chrR} \\
\texttt{wt3}^{\texttt{J}} \leftarrow & \texttt{levl}^{\texttt{J}}(\_). \\
\texttt{wt1}^{\texttt{J+1}} \leftarrow & \texttt{wt4}^{\texttt{J}}.\ \ \texttt{\%a new stage value} \\
\texttt{wt1}^{0}. & \qquad\quad \texttt{\%begin ruleprocessing}
\end{array}$$

Observe that all these rules, but the active rules and the durable change rules, can be given a simple operational interpretation. These safe rules can, for instance, be translated into equivalent relational algebra expressions. Then the overall computation proceeds in a bottom-up fashion from level J to level J+1 (in fact, if we remove the $\texttt{lchR}^{\texttt{J}}$ goals, the whole program becomes XY-stratified and thus efficiently computed [?]).

In our durable-changes policy, however, we use the negation of $\texttt{lchR}^{\texttt{J}}$ as a goal to predict the absence of conflicting *future events*. This feature puts us beyond the scope of any operational semantics, and in the realm of declarative semantics based on the notion of stable models (Definition 2). Therefore, we can now define our durable-change semantics for active databases as follows:

Definition 1 *Let $\mathcal{D} = (\mathcal{S}, \mathcal{C}, \mathcal{A})$ be a database where:*

- $\mathcal{S}$ *denotes a set of schema relations*
- $\mathcal{C}$ *denotes the current content of the database*
- A *denotes a set of active rules on* S

Let P denote the durable-delta program for $\mathcal{D}$. If P has a stable model semantics, then $\mathcal{D}$ is said to obey a durable-change semantics.

For all its superior conceptual benefits this new declarative semantics is bound to remain of little practical consequence until we can translate it into some efficient operational semantics. In general, stable models represent an egregious basis for efficient implementation since computing stable models is NP-hard [?]. Even more restrictive subclasses of programs, such as locally stratified programs or those that have well-founded models, might not yield computation procedures that can be realistically used for active database applications. At this point, therefore, our reader might suspect of having being led to the quagmire of current non-monotonic reasoning research whereby: *'The semantics we like cannot be implemented efficiently...'*. Fortunately, in this case, a careful assignment of priorities to rules and events, will take us out that quagmire and to the solid grounds of very efficient operational semantics. As described more formally next, this can be done by reconciling the stable model semantics with an efficient inflationary-fixpoint computation.

Let r be a rule of a logic program P and let $h(r)$, $gp(r)$ and $gn(r)$, respectively, denote the head of r, the set of positive goals of r and the set of negated goals of r without the negation sign. For instance, if $r : a \leftarrow b, \neg c, \neg d$, then $h(r) = a$, $gp(r) = \{b\}$ and $gn(r) = \{c, d\}$. In the following, P denotes a logic program with negated goals, I and N are subsets of P's Herbrand Base B_P (here, I represents the set of atoms that are true, and N represents those that are false); $ground(P)$ represents the Herbrand instantiation of P.

Definition 2 *Let P be a logic program, and let I and N be subsets of B_P. The* immediate positive-consequence operator *for P given N is defined as:*

$$\Gamma_{P(N)}(I) = \{h(r) \mid r \in ground(P),\ gp(r) \subseteq I,\ gn(r) \subseteq N\}$$

While Γ can also be viewed as a two-place function (on I and N), in the following definition, we view it as a function of I only, inasmuch as N is kept constant. The following characterization of two-valued stable models follows directly from the one given in [**?**]:

Definition 3 *Let P be a logic program with Herbrand base B_P and $\overline{M} = B_P - M$. Then, M is a stable model for P iff:*

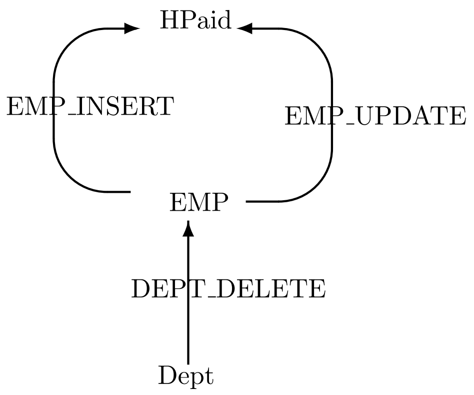
$$\Gamma^{\uparrow\omega}_{P(\overline{M})}(\emptyset) = M$$

Thus M is a stable model if it can be obtained as the ω power of the positive consequence operator, where the set of false atoms is kept constant and equal to the set of atoms not in M. Using this last definition, it is easy to check whether a model M is stable in polynomial time, by simply letting the set of false atoms to be $\overline{M} = B_P - M$. In actual computations, however, the set of false atoms is not known a priori, and educated guesses must be made in the course of the computation when firing rules with negated goals. For instance, it is customary to use a *naive immediate consequence operator*, defined as follows ($\overline{I} = B_P - I$):

$$T_P(I) = \Gamma_{P(\overline{I})}(I).$$

$T_P^{\uparrow\omega}(\emptyset)$ yields the least model for positive programs where T_P is continuous. However, for programs with negated goals, this operator makes the naive closed-world assumption that every atom that is currently not in I is false. However, as successive powers T_P are computed, larger and larger sets I are constructed, and the original assumptions about negated facts are frequently contradicted. Therefore, for most programs with negation, $T_P^{\uparrow\omega}(\emptyset)$ does not yield a stable model, or not even a minimal model. Fortunately, our durable-delta programs offer a very useful exception to this general rule. Let us begin with the concept of *Event Precedence Graph (EPG)*:

Figure 10: The EPG for the rules of Figure 3



Definition 4 *Let P be a durable-delta program. The Event Precedence Graph (EPG) for P is a directed labeled graph that has as nodes the relation names of the database schema. The graph contains an arc from relation* $\mathtt{R}_1$ *to relation* $\mathtt{R}_2$ *with label α iff there is an active rule α having as goals either* $\mathtt{insR}_1, \mathtt{delR1}$*, or* $\mathtt{updR}_1$ *and having either* $\mathtt{rinR}_2, \mathtt{rdeR}_2$*, or* $\mathtt{rupR}_2$ *as its head.*

The EPG for the example at hand is shown in Figure 10. We will now discuss the treatment of acyclic EPG graphs: the treatment of graphs with cycles is discussed in the next section. The *Canonical Rule Precedence Assignment* for an EPG graph is defined as follows:

- Nodes with zero in-degree are assigned level 0
- The arcs departing from a node of level $j \geq 0$ are assigned level j.
- Every node that is the end-node of one or more arcs, is assigned the maximum level of such arcs, plus 1.

Thus, in our example `Dept` (and the rules triggered by its changes) are at level 0, `EMP` is at level 1 and `HPaid` is at level 2. In order to avoid using integers outside the stage argument, we will represent level through a binary precedence relation `prec`. For the example at hand, for instance, we have

$$\mathtt{prec(nil, dd)} \quad \mathtt{prec(dd, ei)} \quad \mathtt{prec(dd, eu)}$$

Thus, `prec` is a graph having as nodes the abbreviated rule names. For each rule r at level 0 there is an arc from a special node node *nil* to r; for each rule r at level j there must be an arc connecting some rule at level $j-1$ to r. Then we have the following theorem:

Theorem 1 *Let P denote the durable-change program, with acyclic EPG graph, and canonical rule precedence assignment. Then, P has a stable model which is equal to $T_P^{\uparrow\omega}(\emptyset)$.*

Proof. It suffices to show every $\texttt{lchR}^{\texttt{J}}$ atom assumed false to fire a rule instance r is not in $T_P^{\uparrow\omega}(\emptyset)$. Indeed , durable-change rules can fire only at their canonical level—i.e., at a level where rules that could affect their triggering events have already fired. Also these rules can never fire again since the EPG is acyclic. □

Observe for example the computation for the example at hand. The computation begins with the exit rules in Figure 9, setting $\texttt{wt1}^{\texttt{J}}$ and then $\texttt{levl}^0(\texttt{dd})$ to true. Thus, the rules of Figure 2 compute database state at level 0, by combining the database before the transaction with the the net effect of all actions till the rule-processing point. The durable changes are also evaluated at this point, assuming as a default that all $\neg\texttt{lchR}$ goals are true. While this assumption is incorrect, no arm follows from it, since only rules enabled by $\texttt{levl}^{\texttt{J}}$ can fire. For instance, for stage value of 0, only the `dd` rule can fire, and its firing event is entered in the durable-change table. The action requested by DEPT_DELETE rule is the deletion of the last tuple in Figure 1 (the `analyst` tuple) which is thus removed by the composition rules. As the computation proceeds with a stage value of 1, no rule fires; thus the delta relations are copied unchanged to next stage value of 2, and $\texttt{levl}^2(\texttt{ei})$ and $\texttt{levl}^2(\texttt{eu})$ are set to true and the first two rules can fire; but with no change was left in the delta tables for this level, the computation proceeds by setting level $\texttt{levl}^3(\texttt{ei})$. There is no candidate triggering event at this level either, and we are now at the top of the EPG graph. Thus $\texttt{wl}^4$ is set to true while $\texttt{levl}^4$ remains false. Thus the computation terminates yielding a stable model for our durable delta program.

Upon successful termination, all remaining entries in the delta relations and all the entries accumulated in the durable-change relations, are written back into stable storage as the transaction commits.

4.1 Recursive Rules

In the previous example, the durable-delta program is recursive, but the EPG is acyclic. Let us now consider the situation where the EPG is cyclic, which corresponds to the situation where the set of active rules alone are recursive. For instance, assume that we have a hierarchy of organizations each identified by a D#; the column `Div` in the `Dept` relation, now denotes the organization to which the department is reporting. Then, we have an active rule DEPT_DEL_PROP which, once an organization is deleted, deletes all organizations reporting to it. The logical counterpart of such a rule is:

DEPT_DEL_PROP:

$$\begin{aligned}\texttt{rdeDept}^{\texttt{J}}(\texttt{Dc},\texttt{N},\texttt{Dp},\texttt{Loc}) \leftarrow\ & \texttt{delDept}^{\texttt{J}}(\texttt{Dp},\_,\_,\_),\\ & \texttt{curDept}^{\texttt{J}}(\texttt{Dc},\texttt{N},\texttt{Dp},\texttt{Loc}),\\ & \texttt{levl}^{\texttt{J}}(\texttt{ddp}),\neg\texttt{lchDept}^{\texttt{J}}(\texttt{Dp},\_,\_,\_).\end{aligned}$$

This last rule introduces a loop from `Dept` to `Dept` the EPG graph of Figure 10. While, programs with acyclic EPGs always have stable models, not all cyclic programs have one. Take, for instance, the following rule that reacts to a tuple with `nil` value being inserted in `HPaid` by deleting the same tuple:

COUNTER_ACTION:

$$\begin{aligned}\texttt{rdeHPaid}^{\texttt{J}+1}(\texttt{nil}) \leftarrow\ & \texttt{insHPaid}^{\texttt{J}}(\texttt{nil}),\\ & \texttt{levl}^{\texttt{J}}(\texttt{ca}),\neg\texttt{lchHPaid}^{\texttt{J}}(\texttt{nil}).\end{aligned}$$

Say now that our delta tables contain `inHPaid(nil)`, and our durable-delta program P has no active rule, but COUNTER_ACTION, affecting this entry in the delta relation. Then, if we assume the insertion of `HPaid(nil)` to be durable, we must fire COUNTER_ACTION, requesting the deletion of this delta tuple—making the original insertion ephemeral. Conversely, if we consider the initial insertion ephemeral, then we cannot fire the rule, thus making the insertion of `HPaid(nil)` durable. This is contradiction means that our program P (which also include the durable change rules not listed above) does not have a stable model, much in the way in which a program containing the rule $a \leftarrow \neg a$ cannot have a stable model. The COUNTER_ACTION rule is therefore disallowed in our durable-delta semantics; to provide the same operational effect as this rule we propose the use of 'instead' rules from Postgres [?], which are given a modified logical translation [?].

Therefore, durable-delta programs with cyclic EPG might not have stable model semantics; moreover, deciding if a given durable-delta program has a stable model is as complex a problem as deciding whether an arbitrary program has a stable model; [1]

Therefore, it appears that cyclic EPG pose an insuperable obstacle to the implementation of our durable change semantics. Fortunately, we can take advantage of the roll-back mechanism of transactions, whereby a computation that has incurred in errors or semantic constraint violations can be simply aborted, while the database is returned to the initial consistent state. We have already used `error` conditions in composition semantics, where, e.g., an insert followed by another insert on the same tuple produces an error. Once the `error` predicates becomes true then the transaction aborts. For a cyclic EPG, therefore,

[1]It suffices to write active rules for an NP-complete problem—e.g., to decide whether a graph has a Hamiltonian circuit.

we can monitor the computation as it takes place, and once we detect that this will not generate a stable model, we can simply abort the computation. As we describe next, this policy can be implemented efficiently—consistently with the fact that checking that a model is stable is PTIME.

Let G be a directed graph, and S be a strong component for G. The *contraction* of S in G yields a new graph G' obtained by (i) eliminating all the arcs of S and merging the nodes of S into one node, say N_S, and (ii) replacing each arc $A \rightarrow B$ by $N_S \rightarrow B$ if $A \in S$, and by $A \rightarrow N_S$ if $B \in S$. The graph obtained from G by contracting all its maximal strong components of G is unique and will be called the *acyclic contraction* of G.

The canonical rule precedence assignment for a cyclic EPG is then constructed as follows: first compute the canonical assignment for its acyclic contraction, and then set all arcs (rules) in a strong component S to the same level as N_S.

For the example at hand, the addition of rule DEPT_DEL_PROP to those of Figure 4, adds a loop on `Dept`; then DEPT_DEL_PROP is assigned to level 0 and the levels of the remaining rules does not change, although the computation of $T_P^{\uparrow\omega}(\emptyset)$ is changed by this rule. Say, for instance, that the database contains the following `Dept` tuples:

```
iniDept (2500, ims, 1000, ‘LA‘)
iniDept (1300, media, 1000, ‘LA‘).
iniDept (2300, prodc, 1300 , ‘LA‘).
```

Then, the computation begin with `levl(dd)` and `lev(ddp)` being set to true and the rules DEPT_DELETE and DEPT_DEL_PROP being triggered by the first tuple in the delta table of Figure 1. The rule DEPT_DELETE triggers a deletion on `EMP` which composes with the last entry from the delta table, and removes it as in the non-recursive case. The recursive rule DEPT_DEL_PROP instead generates a new request on `Dept rdeDept(2300, prodc, 1300, ‘LA‘)`. This does not compose with any current request, and it is entered as `delDept(2300, prodc, 1300, ‘LA‘)` in the delta relation. Now, the stage value is increased, but the precedence level is not changed, and will remain the same until all the requests at this level have been exhausted. At this point the request `delDept(2300, prodc, 1300, ‘LA‘)` is assumed durable, (the durable change rules have been omitted for brevity). Next, the rule DEPT_DEL_PROP can no longer fire since the condition part of the rule fails. Thus, and the computation moves to the next precedence level where it continues as in the non-recursive case.

From the various examples proposed in the literature, it appears that $T_P^{\uparrow\omega}(\emptyset)$ succeeds in computing a stable model for most durable-delta programs of practical interest. However, precautions must be taken against rules such as the COUNTER_ACTION rule where there is no stable model, or even situations where $T_P^{\uparrow\omega}(\emptyset)$ cannot find it. To this end, we add the following rules:

$$\begin{array}{ll} \texttt{fail\_sc} \leftarrow & \texttt{dinDept}^{\texttt{J}}(\texttt{X},\texttt{Y},\texttt{Z},\texttt{W}), \texttt{lchDept}^{\texttt{J}}(\texttt{X},\texttt{Y},\texttt{Z},\texttt{W}) \\ \texttt{fail\_sc} \leftarrow & \texttt{ddeDept}^{\texttt{J}}(\texttt{X},\texttt{Y},\texttt{Z},\texttt{W}), \texttt{lchDept}^{\texttt{J}}\texttt{t}(\texttt{X},\texttt{Y},\texttt{Z},\texttt{W}) \\ \texttt{fail\_sc} \leftarrow & \texttt{dupDept}^{\texttt{J}}(\texttt{X},\texttt{Y},\texttt{Z},\texttt{W}), \\ & \texttt{lchDept}^{\texttt{J}}(\texttt{X},\texttt{Y},\texttt{Z},\texttt{W},\_,\_,\_,\_). \end{array}$$

We need to add a similar rule for each event in a strongly connected component of the EPG, only. Whenever any such a rule fires $T_P^{\uparrow\omega}(\emptyset)$ is no longer a stable model. In this case, `fail_sc` the transaction can be aborted using the following rule:

$$\texttt{error} \leftarrow \ \texttt{fail\_sc}$$

Observe that, as per the rules of Figure 9, `error` immediately terminates computation of the model M and aborts the transaction. Then, M is a stable mode iff and only iff `fail_sc` $\notin$ `M`, i.e., when the error has been produced by the composition rules rather than by a violation of the stability condition. Independent of its cause, `error` always results in an immediate transaction-abort.

5 Termination

When `fail_sc` does not occur $T_P^{\uparrow\omega}(\emptyset)$ produces a stable model. The main question that remains open is whether it terminates after a finite number of steps, or only an infinite computation to the first ordinal can yield the stable model. Using the durable change semantics, and the Datalog$_{1S}$ formalism, we can now derive a simple a practical solution to this problem, that is in normally of very intractable nature.

Since in Datalog$_{\mathrm{1S}}$ functions symbols are confined to an argument, $T_P^{\uparrow\omega}(\emptyset)$ defines a computation that either terminates or becomes ultimately periodic

Definition 5 *A function f on natural numbers is said to be* ultimately periodic *with period (n, k), where n and k are non-negative integers, if for all $j \geq n$ we have $f(j + k) = f(j)$.*

Let $M = T_P^{\uparrow\omega}(\emptyset)$, and let M^J denote the set of atoms in M with stage value equal to J. For a Datalog$_{\mathrm{1S}}$ program P, M^J can be viewed as a function that maps an integer J to the set of atoms in $T_P^{\uparrow\omega}(\emptyset)$ that have stage argument J. Then, we have the following theorem [**?**]:

Lemma 1 *Let P be a* Datalog$_{\mathrm{1S}}$ *program. Then one of the following two cases must hold:*

1. **[Finite Set of Stage Values]** *There exist an integer n such that, for $J > n$: $M^J = \emptyset$*

2. [**Periodic Behavior**] *The set of stage values is not finite, but there exist two integers n and k such that for every $J > n$: $M^{J+k} = M^J$.*

We now have the following theorem:

Theorem 2 *Let P be a durable-delta program. If P is* Datalog$_{1S}$*, then there exists an integer n such that for every $J > n$, $M^J = \emptyset$. Then $M = \bigcup_{1 \le j \le n} M^J$ is the stable model of P iff* `fail_sc` $\notin$ `M`.

Proof: It suffices to prove that the computation is not eventually periodic. Indeed, assume that the computation becomes periodic after n with periodicity k. Then if M contains a $dinR^j(X)$ with $j > n$ then it must also contain $dinR^{j+k}(X)$. Observe that the latter requires a $insR^{j+k}(X)$ to in delta relation—and this requires that M contains some $chrR^{j+h}(X)$ for $0 < h \le k$. Then, $lchR^j(X)$ is true, and that is a contradiction, as error is generated and the computation terminates. Similar considerations hold for $ddeR^{j+k}(X)$ and $dupR^{j+k}(X)$. $\Box$

Therefore, only a finite number of distinct stage values is possible for durable-delta programs where the computation can be stopped at the first `n` for which `levl`$^{\mathtt{n}}$ is not set to true. If that occurs at the m-step of the computation [2] of $T_P^{\uparrow\omega}(\emptyset)$ then we have that: $M = \bigcup_{1 \le j \le m} T_P^{\uparrow j}(\emptyset)$.

Therefore, the durable-change semantics solves the difficult termination problem, whenever the durable-delta program is Datalog$_{1S}$. In practical terms, this means that the original active rules must be free of interpreted functions; i.e., the values of the rule head are not constructed using arithmetic or aggregates. This is the case for the majority of rules taken from real-life examples. While space limitations prevent us from discussing the more general case, it suffices to say that termination can be ensured in the most general terms by making the composition rules more strict through the inclusion of key constraints. In fact, the inclusion of these constraints suggest various refinements on the basic durable change semantics, including a greater role for non-deterministic computations. These will be discussed in future papers.

6 Conclusion

This paper presented several new results. A first novelty is the notion of durable-change semantics, which ensures termination of active rule programs by making their behavior more consistent with transaction semantics. This result, obtained using the Datalog$_{1S}$ framework, provides a tangible proof that the power of active databases, that was previously considered impervious to formal treatment, can in fact be tamed and improved with the help of the semantics of deductive databases. (Nor benefits flow in only one direction, since this paper provides a rare example of a successful derivation of efficient operational semantics for a problem characterized by stable-model semantics).

In my recent research, I have been pursuing the thesis that a conceptual unity underlies the areas of active databases, temporal databases and deductive databases [**?**, **?**, **?**]. The results of this paper, bring further support to this thesis, and, hopefully, will promote the confluence of these three areas of database research.

Acknowledgments

Thanks are due to Antonio Brogi for many improvements.

References

[1] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, *Proc. 5th Int. Conf. on Logic Programming*, MIT Press, 1988.

[2] M.L. Stonebraker, A. Jhingran, J. Goh, and S. Potamianos. On rules, procedure, cacheing and views in data base systems. In *ACM SIGMOD Int. Conf. on Management of Data*, pages 281–290, 1990.

[3] J. Chomicki, Temporal deductive databases, *Temporal Databases: Theory, Design and Implementation*, A. Tansel et al. (eds), Benjamin/Cummings, 1993.

[4] J. Chomicki, "Polynomial-time Computable Queries in Temporal Deductive Database Systems," *PODS* 1990.

[5] S. Ghandeharizadeh, R. Hull and D. Jacobs, "On Implementing a Language for Specifying Active Database Execution' Models, *Procs. Int. Conf. on Very Large Databases,* 1993.

[6] Widom J., "The Starburst Active Database Rule System", To appear in IEEE Trans. On Knowledge and Data Engineering.

[7] U. Dayal, E.N. Hanson, and J. Widom *Active Database Systems,* "Modern Database Systems, W. Kim (ed.), Addison Wesley, 1995.

[8] Y. Motakis, and C. Zaniolo, *Composite Temporal Events in Active Databases: a Formal Semantics*, submitted for publication.

[9] J.S. Schlipf, The expressive powers of logic programming semantics, *Proc. ACM-PODS*, 1990, 196-204.

[10] Zaniolo, C., N. Arni, K. Ong, "Negation and Aggregates in Recursive Rules: the $\mathcal{LDL}$++ Approach", *Proc. 3rd Int. Conf. on Deductive and O-O DBs, DOOD-93*, Phoenix, AZ, Dec 6-8, 1993.

[11] C. Zaniolo, "A unified semantics for active and deductive databases", In *Procs. 1st Int. Workshop on Rules in Database Systems*, pages 271–287, Springer-Verlag, 1993

[12] C. Zaniolo, "Active Database Rules with Transaction-Conscious Stable-Model Semantics," Technical Report, UCLA CS Dept., May 1995.

[2] It is easy to show that $m = 5 \times n + 1$.