

Transaction Logic with (Complex) Events

Ana Sofia Gomes and José Júlio Alferes*

CENTRIA - Dep. de Informática, Faculdade Ciências e Tecnologias
Universidade Nova de Lisboa

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

This work deals with the problem of combining reactive features, such as the ability to respond to events and define complex events, with the execution of transactions over general Knowledge Bases (KBs).

With this as goal, we build on Transaction Logic (\mathcal{TR}), a logic precisely designed to model and execute transactions in KBs defined by arbitrary logic theories. In it, transactions are written in a logic-programming style, by combining primitive update operations over a general KB, with the usual logic programming connectives and some additional connectives e.g. to express sequence of actions. While \mathcal{TR} is a natural choice to deal with transactions, it remains the question whether \mathcal{TR} can be used to express complex events, but also to deal simultaneously with the detection of complex events and the execution of transactions. In this paper we show that the former is possible while the latter is not. For that, we start by illustrating how \mathcal{TR} can express complex events, and in particular, how SNOOP event expressions can be translated in the logic. Afterwards, we show why \mathcal{TR} fails to deal with the two issues together, and to solve the intended problem propose Transaction Logic with Events, its syntax, model theory and executional semantics. The achieved solution is a non-monotonic extension of \mathcal{TR} , which guarantees that every complex event detected in a transaction is necessarily responded.

To appear in Theory and Practice of Logic Programming (TPLP)

KEYWORDS: reactivity, complex events, transaction logic

1 Introduction

Reactivity stands for the ability to detect complex changes (also denoted as events) in the environment and react automatically to them according to some pre-defined rules. This is a pre-requisite of many real-world applications, such as web-services providing different services depending on external information, multi-agent systems adapting their knowledge and actions according to the changes in the environment, or monitoring systems reacting to information detected by their sensors and issuing actions automatically in response to it. In reactive systems, e.g. in those based on Event-Condition-Action (ECA) languages (Alferes et al. 2011; Bry et al. 2006; Chomicki et al. 2003), the reaction triggered by the detection of a complex event may itself be a complex action, formed e.g. by the sequential execution of several basic actions. Moreover, we sustain that sometimes reactive systems are also required to execute *transactions* in response to events. For example, consider an airline web-service scenario where an external event arrives stating that a partner airline is on strike for a given time period. Then, the airline must address this event by e.g.

* The authors thank Michael Kifer for the valuable discussions in a preliminary version of this work. The first author was supported by the grant SFRH/BD/64038/2009 and by project ERRO (PTDC/EIA-CCO/121823/2010). The second author was supported by project ASPEN PTDC/EIA-CCO/110921/2009

rescheduling flights with alternative partners or refund tickets for passengers who do not accept the changes. Clearly, some transactional properties regarding these actions must be ensured: viz. it can never be the case that a passenger is simultaneously not refunded nor have an alternative flight; or that she is completely refunded and has a rescheduled flight.

Although the possibility of executing transactions is of crucial importance in many of today's systems, and a must e.g. in database systems, most reactive languages do not deal with it. Some exceptions exist, but are either completely procedural and thus lack from a clear declarative semantics (as e.g. in (Papamarkos et al. 2006)), or have a strong limitation on the expressivity of either the actions or events (as e.g. in (Zaniolo 1995; Lausen et al. 1998)).

In this paper we propose Transaction Logic with Events, \mathcal{TR}^{ev} , an extension of \mathcal{TR} (Bonner and Kifer 1993) integrating the ability to reason and execute transactions over very general forms of KBs, with the ability to detect complex events. For this, after a brief overview of \mathcal{TR} , we show how it can be used to express and reason about complex events, and in particular, how it can express most SNOOP event operators (Adaikkalavan and Chakravarthy 2006) (Section 2). We proceed by showing why \mathcal{TR} alone is not able to deal with both the detection of complex events and the execution of transactions, and, in particular, why it does not guarantee that all complex events detected during the execution of a transaction are responded within that execution. For solving this problem, we define \mathcal{TR}^{ev} , its language and model theory (Section 3.1), as well as its executional semantics (Section 3.2).

2 Using \mathcal{TR} to express complex events

In this section we briefly recall \mathcal{TR} 's syntax and semantics with minor syntactic changes from the original, to help distinguish between actions and event occurrences, something that is useful ahead in the paper when extending \mathcal{TR} to deal with reactive features and complex events.

Atoms in \mathcal{TR} have the form $p(t_1, \dots, t_n)$ where p is a predicate symbol and t_i 's are terms (variables, constants, function terms). For simplicity, and without loss of generality (Bonner and Kifer 1998), we consider Herbrand instantiations, as usual. To build complex formulas, \mathcal{TR} uses the classical connectives $\wedge, \vee, \neg, \leftarrow$ and the connectives \otimes, \Diamond denoting serial conjunction and hypothetical execution. Informally, $\phi \otimes \psi$ is an action composed of an execution of ϕ followed by an execution of ψ ; and $\Diamond\phi$ tests if ϕ can be executed without materializing the changes. In general, formulas are viewed as (the execution of) transactions, where, $\phi \wedge \psi$ is the simultaneous execution of ϕ and ψ ; $\phi \vee \psi$ the non-deterministic choice of executing ϕ or ψ . $\phi \leftarrow \psi$ is a rule saying that one way to execute of ϕ is by executing ψ . As in classical logic, \wedge and \leftarrow can be written using \vee and \neg (e.g. $\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi)$). Finally, we also use the connective ; as it is useful to express common complex events. $\phi ; \psi$ says that ψ is true after ϕ but possibly interleaved with other occurrences, and it can be written in \mathcal{TR} syntax as: $\phi \otimes \text{path} \otimes \psi$ where $\text{path} \equiv (\varphi \vee \neg\varphi)$ is a tautology that holds in paths of arbitrary size (Bonner and Kifer 1998).

For making possible the separation between the theory of states and updates, from the logic that combines them in transactions, \mathcal{TR} considers a pair of oracles – \mathcal{O}^d (data oracle) and \mathcal{O}^t (transition oracle) – as a parameter of the theory. These oracles are mappings that assume a set of state identifiers. \mathcal{O}^d is a mapping from state identifiers to a set of formulas that hold in that state, and \mathcal{O}^t is a mapping from pairs of state identifiers to sets of formulas that hold in the transition of those states. These oracles can be instantiated with a wide variety of semantics, as e.g. relational databases, well-founded semantics, action languages, etc. (Bonner and Kifer 1993). For example, a relational database can be modeled by having states represented as sets of

ground atomic formulas. Then, the data oracle simply returns all these formulas, i.e., $\mathcal{O}^d(D) = D$, and for each predicate p in the KB, the transition oracle defines $p.ins$ and $p.del$, representing the insertion and deletion of p , respectively. Formally, $p.ins \in \mathcal{O}^t(D_1, D_2)$ iff $D_2 = D_1 \cup \{p\}$ and, $p.del \in \mathcal{O}^t(D_1, D_2)$ iff $D_2 = D_1 \setminus \{p\}$. SQL-style bulk updates can also be defined by \mathcal{O}^t .

Example 1 (Moving objects - \mathcal{TR})

As a \mathcal{TR} 's illustration, assume the prior relational database oracles and the action $move(O, X, Y)$ defining the relocation of object O from position X into position Y . In such a KB, states are defined using the predicates $location(O, P)$ saying that object O is in position P , and $clear(X)$ stating that X is clear to receive an object. In \mathcal{TR} , the move (trans)action can be expressed by:

$$\begin{aligned} move(O, X, Y) &\leftarrow location(O, X) \otimes clear(Y) \otimes localUpdt(O, X, Y) \\ localUpdt(O, X, Y) &\leftarrow location(O, X).del \otimes location(O, Y).ins \otimes clear(Y).del \otimes clear(X).ins \end{aligned}$$

\mathcal{TR} 's theory is built upon the notion of sequences of states denoted as *paths*. Formulas are evaluated over paths, and truth in \mathcal{TR} means *execution*: a formula is said to succeed over a path, if that path represents a valid execution for that formula. Although not part of the original \mathcal{TR} , here paths' state transitions are labeled with information about what (atomic occurrences) happen in the transition of states. Precisely, paths have the form $\langle D_0 \xrightarrow{O_1} D_1 \xrightarrow{O_2} \dots \xrightarrow{O_k} D_k \rangle$, where D_i 's are states and O_i 's are labels (used later to annotate atomic event occurrences).

As usual, satisfaction of complex formulas is based on interpretations. These define what atoms are true in what paths, by mapping every path to a set of atoms. However, only the mappings compliant with the specified oracles are interpretations:

Definition 1 (Interpretation)

An interpretation is a mapping M assigning a set of atoms (or \top^1) to every path, with the following restrictions (where D_i 's are states, and φ a formula):

1. $\varphi \in M(\langle D \rangle)$ if $\varphi \in \mathcal{O}^d(D)$
2. $\{\varphi, \mathbf{o}(\varphi)\} \subseteq M(\langle D_1 \xrightarrow{\mathbf{o}(\varphi)} D_2 \rangle)$ if $\varphi \in \mathcal{O}^t(D_1, D_2)$

In point 2 we additionally (i.e., when compared to the original definition) force $\mathbf{o}(\varphi)$ to belong to the same path where the primitive action φ is made true by the oracle, something that later (in Section 3) will help detect events associated with primitive actions, like “on insert/delete”.

Next, we define operations on paths, and satisfaction of complex formulas over general paths.

Definition 2 (Path Splits, Subpaths and Prefixes)

Let π be a k -path, i.e. a path of length k of the form $\langle D_1 \xrightarrow{O_1} \dots \xrightarrow{O_{k-1}} D_k \rangle$. A *split* of π is any pair of subpaths, π_1 and π_2 , such that $\pi_1 = \langle D_1 \xrightarrow{O_1} \dots \xrightarrow{O_{i-1}} D_i \rangle$ and $\pi_2 = \langle D_i \xrightarrow{O_i} \dots \xrightarrow{O_{k-1}} D_k \rangle$ for some i ($1 \leq i \leq k$). In this case, we write $\pi = \pi_1 \circ \pi_2$.

A subpath π' of π is any subset of states and annotations of π where both the order of the states and their annotations is preserved. A prefix π_1 of π is any subpath of π sharing the initial state.

¹ For not having to consider partial mappings, besides formulas, interpretations can also return the special symbol \top . The interested reader is referred to (Bonner and Kifer 1993) for details.

Definition 3 (\mathcal{TR} Satisfaction of Complex Formulas)

Let M be an interpretation, π a path and ϕ a formula. If $M(\pi) = \top$ then $M, \pi \models_{\mathcal{TR}} \phi$; else:

1. **Base Case:** $M, \pi \models_{\mathcal{TR}} \phi$ iff $\phi \in M(\pi)$ for every event occurrence ϕ
2. **Negation:** $M, \pi \models_{\mathcal{TR}} \neg\phi$ iff it is not the case that $M, \pi \models_{\mathcal{TR}} \phi$
3. **Disjunction:** $M, \pi \models_{\mathcal{TR}} \phi \vee \psi$ iff $M, \pi \models_{\mathcal{TR}} \phi$ or $M, \pi \models_{\mathcal{TR}} \psi$.
4. **Serial Conjunction:** $M, \pi \models_{\mathcal{TR}} \phi \otimes \psi$ iff there exists a split $\pi_1 \circ \pi_2$ of π s.t. $M, \pi_1 \models_{\mathcal{TR}} \phi$ and $M, \pi_2 \models_{\mathcal{TR}} \psi$
5. **Executional Possibility:** $M, \pi \models_{\mathcal{TR}} \Diamond\phi$ iff π is a 1-path of the form $\langle D \rangle$ for some state D and $M, \pi' \models_{\mathcal{TR}} \phi$ for some path π' that begins at D .

Models and logical entailment are defined as usual. An interpretation models/satisfies a set of rules if each rule is satisfied in every possible path, and an interpretation models a rule in a path, if whenever it satisfies the antecedent, it also satisfies the consequent.

Definition 4 (Models, and Logical Entailment)

An interpretation M is a *model* of a formula ϕ iff for every path π , $M, \pi \models_{\mathcal{TR}} \phi$. M is a model of a set of rules P (denoted $M \models_{\mathcal{TR}} P$) iff it is a model of every rule in P .

ϕ is said to logically entail another formula ψ iff every model of ϕ is also a model of ψ .

Logical entailment is useful to define general equivalence and implication of formulas that express properties like “transaction ϕ is equivalent to transaction ψ ” or “whenever transaction ψ is executed, ψ' is also executed”. Moreover, if instead of transactions, we view the propositions as representing event occurrences, this entailment can be used to express complex events. For instance, imagine we want to state a complex event *alarm*, e.g. triggered whenever event ev_1 occurs after both ev_2 and ev_3 occur simultaneously. This can be expressed in \mathcal{TR} as:

$$\mathbf{o}(alarm) \leftarrow (\mathbf{o}(e_2) \wedge \mathbf{o}(e_3)); \mathbf{o}(e_1) \quad (1)$$

In every model of this formula, whenever there is a (sub)path where both $\mathbf{o}(e_2)$ and $\mathbf{o}(e_3)$ are true, followed by a (sub)path where $\mathbf{o}(e_1)$ holds, then $\mathbf{o}(alarm)$ is true in the *whole* path.

Other complex event definitions are possible, and in fact we can encode most of SNOOP (Adaikkalavan and Chakravarthy 2006) operators in \mathcal{TR} . This is shown in Theorem 1 where, for a given history of past event occurrences, we prove that if an event expression is true in SNOOP, then there is a translation into a \mathcal{TR} formula which is also true in that history. Since a SNOOP history is a set of atomic events associated with discrete points in time, the first step is to build a \mathcal{TR} path expressing such history. We construct it as a sequence of state identifiers labeled with time, where time point i takes place in the transition of states $\langle s_i, s_{i+1} \rangle$, and only consider interpretations M over such a path that are *compatible* with SNOOP’s history, i.e. such that, for every atomic event that is true in a time i , M makes the same event true in the path $\langle s_i, s_{i+1} \rangle$.

Theorem 1 (SNOOP Algebra and \mathcal{TR})

Let E be a SNOOP algebra expression without periodic and aperiodic operators, H be a history containing the set of all SNOOP primitive events $e_j^i[t_1]$ that have occurred over the time interval t_1, t_{max} , and $\langle s_1, \dots, s_{max+1} \rangle$ be a path with size $t_{max} - t_1 + 1$. Let τ be the following function:

- Primitive:** $\tau(E) = \mathbf{o}(E)$ where E is a primitive event
Sequence: $\tau(E_1; E_2) = \tau(E_1) \otimes \text{path} \otimes \tau(E_2)$
Or: $\tau(E_1 \vee E_2) = \tau(E_1) \vee \tau(E_2)$
AND: $\tau(E_1 \Delta E_2) = [(\tau(E_1) \otimes \text{path}) \wedge (\text{path} \otimes \tau(E_2))] \vee [(\tau(E_2) \otimes \text{path}) \wedge (\text{path} \otimes \tau(E_1))]$
NOT: $\tau(\neg(E_3)[E_1, E_2]) = \tau(E_1) \otimes \neg\tau(E_3) \otimes \tau(E_2)$

Then, $[t_i, t_f] \in E[H] \Rightarrow \forall M \text{ compatible with } H, M, \langle s_{t_i}, \dots, s_{t_{f+1}} \rangle \models_{\mathcal{TR}} \tau(E)$, where, cf. (Adaikkalavan and Chakravarthy 2006), $E[H]$ is the set of time intervals (t_i, t_f) where E occurs over H in an unrestricted context, and where M is compatible with H if, for each $e_j^i[t_i] \in H$: $M, \langle s_{t_i}, s_{t_{i+1}} \rangle \models_{\mathcal{TR}} \mathbf{o}(e_j)$.

Besides the logical entailment, \mathcal{TR} also provides the notion of executional entailment for reasoning about properties of a *specific* execution path.

Definition 5 (Executional Entailment)

Let P be a set of rules, ϕ a formula, and $D_0 \xrightarrow{O_1} \dots \xrightarrow{O_n} D_n$ a path.

$P, (D_0 \xrightarrow{O_1} \dots \xrightarrow{O_n} D_n) \models \phi$ (\star) iff for every model M of P , $M, \langle D_0 \xrightarrow{O_1} \dots \xrightarrow{O_n} D_n \rangle \models \phi$.

Additionally, $P, D_0 \models \phi$ holds, if there is a path $D_0 \xrightarrow{O_1} \dots \xrightarrow{O_n} D_n$ that makes (\star) true.

$P, (D_0 \xrightarrow{O_1} \dots \xrightarrow{O_n} D_n) \models \phi$ says that a successful execution of transaction ϕ respecting the rules in P , can change the KB from state D_0 into D_n with a sequence of occurrences O_1, \dots, O_n . E.g., in the Example 1 (with obvious abbreviations), the statement $P, (\{cl(t), l(c, o)\} \xrightarrow{\mathbf{o}(l(c, o).del)} \{cl(t)\} \xrightarrow{\mathbf{o}(l(c, t).ins)} \{cl(t), l(c, t)\} \xrightarrow{\mathbf{o}(cl(t).del)} \{l(c, t)\} \xrightarrow{\mathbf{o}(cl(o).ins)} \{l(c, t), cl(o)\}) \models move(c, o, t)$ means that a possible result of executing the transaction $move(c, oven, table)$ starting in the state $\{clear(table), loc(c, oven)\}$ is the path with those 5 states, ending in $\{loc(c, table), clear(oven)\}$.

This entailment has a corresponding proof theory (Bonner and Kifer 1993) which, for a subset of \mathcal{TR} , is capable of *constructing* such a path given a program, a \mathcal{TR} formula, and an initial state. I.e. a path where the formula can be executed. If no such path exists, then the transaction fails, and nothing is built after the initial state.

3 \mathcal{TR}^{ev} : combining the execution of transactions with complex event detection

Reactive languages need to express behaviors like: “on *alarm* do action a_1 followed by action a_2 ”, where the actions $a_1 \otimes a_2$ may define a transaction, and *alarm* is e.g. the complex event in (1). Clearly, \mathcal{TR} can individually express and reason about transaction $a_1 \otimes a_2$, and its complex event. So, the question is whether it can deal with both simultaneously. For that, two important issues must be tackled: 1) how to model the triggering behavior of reactive systems, where the occurrence of an event drives the execution of a transaction in its response; 2) how to model the transaction behavior that prevents transactions to commit until all occurring events are responded.

Regarding 1), (Bonner et al. 1993) shows that simple events can be triggered in \mathcal{TR} as:

$$\begin{aligned} p &\leftarrow body \otimes ev \\ ev &\leftarrow \mathbf{r}(ev) \end{aligned} \tag{2}$$

With such rules, in all paths that make p true (i.e., in all executions of transaction p) the event ev is triggered/fired (after the execution of some arbitrary *body*), and ev ’s response, $\mathbf{r}(ev)$, is executed. Note that, both $\mathbf{r}(ev)$ and *body* can be defined as arbitrary formulas.

But, this is just a very simple and specific type of event: atomic events that are explicitly triggered by a transaction defined in the program. In general, atomic events can also arrive as external events, or because some primitive action is executed in a path (e.g. as the database triggers - “on insert/on delete”). Triggering external events in \mathcal{TR} can be done by considering the paths that make the external event true. E.g., if one wants to respond to an external event ev from an initial state, all we need to do is find the paths π starting in that state, s.t. $P, \pi \models ev$, where P includes the last rule from (2) plus the rules defining ev ’s response.

The occurrences of primitive actions can be tackled by Point 2 of Def. 1, and the occurrence of complex events can be defined as prescribed in Section 2. However, the above approach of

(Bonner et al. 1993) does not help for driving the execution of an event response when such occurrences become true. For instance, the ECA-rule before could be stated as:

$$\begin{aligned}\mathbf{o}(alarm) &\leftarrow (\mathbf{o}(e_2) \wedge \mathbf{o}(e_3)); \mathbf{o}(e_1) \\ \mathbf{r}(alarm) &\leftarrow a_1 \otimes a_2\end{aligned}$$

But this does not drive the execution of $\mathbf{r}(alarm)$ when $\mathbf{o}(alarm)$ holds; one has further to force that whenever $\mathbf{o}(alarm)$ holds, $\mathbf{r}(alarm)$ must be made true subsequently. Of course, adding a rule $\mathbf{r}(alarm) \leftarrow \mathbf{o}(alarm)$ would not work: such rule would only state that, one alternative way to satisfy the response of alarm is to make its occurrence true. And for that, it would be enough to satisfy $\mathbf{o}(alarm)$ to make $\mathbf{r}(alarm)$ true, which is not what is intended.

Clearly, this combination implies two different types of formulas with two very different behaviors: the *detection* of events which are tested for occurrence w.r.t. a past history; and the *execution* of transactions as a response to them, which intends to construct paths where formulas can succeed respecting transactional properties. This has to be reflected in the semantics and these formulas should be evaluated differently accordingly to their nature.

Regarding 2), as in database triggers, transaction's execution must depend on the events triggered. Viz., an event occurring during a transaction execution can delay that transaction to commit/succeed until the event response is successfully executed, and the failure of such response should imply the failure of the whole transaction. Encoding this behavior requires that, if an event occurs during a transaction, then its execution needs to be *expanded* with the event response. Additionally, this also precludes transactions to succeed in paths where an event occurs and is not responded (even if the transaction would succeed in that path if the event did not existed).

For addressing these issues, below we define \mathcal{TR}^{ev} . This extension of \mathcal{TR} evaluates event formulas and transaction formulas differently, using two distinct relations (respectively $\models_{\mathcal{TR}}$ and \models), and occurrences and responses are syntactic represented w.r.t. a given event name e , as $\mathbf{o}(e)$ and $\mathbf{r}(e)$, respectively. In this context, \models requires transactions to be satisfied in expanded paths, where every occurring event (made true by $\models_{\mathcal{TR}}$) is properly responded.

3.1 \mathcal{TR}^{ev} Syntax and Model Theory

To make possible a different evaluation of events and transactions, predicates in \mathcal{TR}^{ev} are partitioned into transaction names (\mathcal{P}_t), event names (\mathcal{P}_e), and oracle primitives ($\mathcal{P}_{\mathcal{O}}$) and, as with \mathcal{TR} , we work with the Herbrand instantiation of the language.

Formulas in \mathcal{TR}^{ev} are partitioned into transaction formulas and event formulas. *Event formulas* denote formulas meant to be *detected* and are either an event occurrence, or an expression defined inductively as $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \otimes \psi$, or $\phi; \psi$ where ϕ and ψ are event formulas. An *event occurrence* is of the form $\mathbf{o}(\varphi)$ s.t. $\varphi \in \mathcal{P}_e$ or $\varphi \in \mathcal{P}_{\mathcal{O}}$. Note that, we preclude the usage of \Diamond in event formulas, as it would make little sense to detect occurrences based on what could possibly be executed.

Transaction formulas are formulas that can be *executed*, and are either a transaction atom, or an expression defined inductively as $\neg\phi$, $\Diamond\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, or $\phi \otimes \psi$. A *transaction atom* is either a transaction name (in \mathcal{P}_t), an oracle defined primitive (in $\mathcal{P}_{\mathcal{O}}$), the response to an event ($\mathbf{r}(\varphi)$ where $\varphi \in \mathcal{P}_{\mathcal{O}} \cup \mathcal{P}_e$), or an event name (in \mathcal{P}_e). The latter corresponds to the (trans)action of *explicitly* triggering an event directly in a transaction as in (2) or as an external event. As we shall see (Def. 7) explicitly triggering an event changes the path of execution (by asserting the information that the event has happened in the current state) and, as such, is different from simply inferring (or detecting) what events hold given a past path.

Finally, rules have the form $\varphi \leftarrow \psi$ and can be transaction or (complex) event rules. In a transaction rule φ is a transaction atom and ψ a transaction formula; in an event rule φ is an event occurrence and ψ is a event formula. A *program* is a set of transaction and event rules.

Importantly, besides the data and transition oracles, \mathcal{TR}^{ev} is also parametric on a *choice* function defining what event should be selected at a given time in case of conflict. Since defining what event should be picked from the set of occurring events depends on the application in mind, \mathcal{TR}^{ev} does not commit to any particular definition, encapsulating it in function *choice*.

As a reactive system, \mathcal{TR}^{ev} receives a series of external events which may cause the execution of transactions in response. This is defined as $P, D_0 \models e_1 \otimes \dots \otimes e_k$, where D_0 is the initial KB state and $e_1 \otimes \dots \otimes e_k$ is the sequence of external events that arrive to the system. Here, we want to find the path $D_0 \xrightarrow{O_1} \dots \xrightarrow{O_n} D_n$ encoding a KB evolution that responds to $e_1 \otimes \dots \otimes e_k$.

As mentioned, triggering explicit events is a transaction formula encoding the *action* of making an occurrence explicitly true. This is handled by the definition of interpretation, in a similar way to how atomic events defined by oracles primitives are made true:

Definition 6 (\mathcal{TR}^{ev} interpretations)

A \mathcal{TR}^{ev} interpretation is a \mathcal{TR} interpretation that additionally satisfies the restriction: 3) $\mathbf{o}(e) \in M(\langle D^{\mathbf{o}(e)} \rightarrow D \rangle)$ if $e \in \mathcal{P}_e$

We can now define the satisfaction of complex formulas, and then models of a program. Event formulas are evaluated w.r.t. the relation $\models_{\mathcal{TR}}$ specified in Def. 3. Transaction formulas are evaluated w.r.t. the relation \models which requires formulas to be true in *expanded paths*, in which every occurring event is responded (something dealt by $\exp_M(\pi)$, defined below).

Definition 7 (Satisfaction of Transaction Formulas and Models)

Let M be an interpretation, π a path, ϕ transaction formula. If $M(\pi) = \top$ then $M, \pi \models \phi$; else:

1. **Base Case:** $M, \pi \models p$ iff $\exists \pi'$ prefix of π s.t. $p \in M(\pi')$ and $\pi = \exp_M(\pi')$, for every transaction atom p where $p \notin \mathcal{P}_e$.
2. **Event Case:** $M, \pi \models e$ iff $e \in \mathcal{P}_e$, $\exists \pi'$ prefix of π s.t. $M, \pi' \models_{\mathcal{TR}} \mathbf{o}(e)$ and $\pi = \exp_M(\pi')$.
3. **Negation:** $M, \pi \models \neg \phi$ iff it is not the case that $M, \pi \models \phi$
4. **Disjunction:** $M, \pi \models \phi \vee \psi$ iff $M, \pi \models \phi$ or $M, \pi \models \psi$.
5. **Serial Conjunction:** $M, \pi \models \phi \otimes \psi$ iff $\exists \pi'$ prefix of π and some split $\pi_1 \circ \pi_2$ of π' such that $M, \pi_1 \models \phi$ and $M, \pi_2 \models \psi$ and $\pi = \exp_M(\pi')$.
6. **Executional Possibility:** $M, \pi \models \Diamond \phi$ iff π is a 1-path of the form $\langle D \rangle$ for some state D and $M, \pi' \models \phi$ for some path π' that begins at D .

An interpretation M is a *model* of a transaction formula (resp. event formula) ϕ iff for every path π , $M, \pi \models \phi$ (resp. $M, \pi \models_{\mathcal{TR}} \phi$). M is a model of a program P (denoted $M \models P$) iff it is a model of every (transaction and complex event) rule in P .

$\exp_M(\pi)$ is a function that, given a path with possibly unanswered events, expands it with the result of responding to those events. Its definition must enforce have some procedural nature: it must start by detecting which are the unanswered events; pick one of them, according to a given *choice* function; then expand the path with the response of the chosen event. The response to this event, computed by operator \mathcal{R}_M defined below, may, in turn, generate the occurrence of further events. So, \mathcal{R}_M must be iterated until no more unanswered events exist.

Definition 8 (Expansion of a Path)

For a path π_1 and an interpretation M , the response operator $\mathcal{R}_M(\pi_1)$ is defined as follows:

$$\mathcal{R}_M(\pi_1) = \begin{cases} \pi_1 \circ \pi_2 & \text{if } \text{choice}(M, \pi_1) = e \text{ and } M, \pi_2 \models \mathbf{r}(e) \\ \pi_1 & \text{if } \text{choice}(M, \pi_1) = \epsilon \end{cases}$$

The expansion of a path π is $\exp_M(\pi) = \uparrow \mathcal{R}_M(\pi)$.

In general it may not be possible to address all events in a finite path, and thus, \mathcal{R}_M may not have a fixed-point. In fact, non-termination is a known problem of reactive systems, and is often undecidable for the general case (Bailey et al. 2004). However, if termination is possible, then a fixed-point exists and each iteration of \mathcal{R}_M is an approximation of the expansion operator \exp_M .

This definition leaves open the *choice* function, that is taken as a further parameter of \mathcal{TR}^{ev} , and specifies how to choose the next unanswered event to respond to. For its instantiation one needs to decide: 1) in which order should events be responded and 2) how should an event be responded. The former defines the handling order of events in case of conflict, e.g. based on when events have occurred (temporal order), on a priority list, or any other criteria. The latter defines the response policy of an ECA-language, i.e. when is an event considered to be responded. E.g., if an event occurs more than once before the system can respond to it, this specifies if such response should be issued only once or equally to the amount of occurrences. Choosing the appropriate operational semantics depends on the application in mind. In the following definition we exemplify how this *choice* function can be instantiated, for a case when events are responded in the (temporal) order in which they occurred, and events for which there was already a response are not responded again.

Definition 9 (Temporal choice function)

Let M be an interpretation and π be a path. The temporal choice function is $\text{choice}(M, \pi) = \text{firstUnans}(M, \pi, \text{order}(M, \pi))$ where:

- $\text{order}(M, \pi) = \langle e_1, \dots, e_n \rangle$ iff $\forall e_i \ 1 \leq i \leq n, \exists \pi_i$ subpath of π where $M, \pi \models_{\mathcal{TR}} \mathbf{o}(e_i)$ and $\forall e_j$ s.t. $i < j$ then e_j occurs after e_i
- e_2 occurs after e_1 w.r.t. π and M iff there exists π_1, π_2 subpaths of π such that $\pi_1 = \langle D_i \xrightarrow{O_i} \dots \xrightarrow{O_{j-1}} D_j \rangle$, $\pi_2 = \langle D_n \xrightarrow{O_n} \dots \xrightarrow{O_{m-1}} D_m \rangle$, $M, \pi_1 \models \mathbf{o}(e_1)$, $M, \pi_2 \models \mathbf{o}(e_2)$ and $D_j \leq D_m$ w.r.t. the ordering in π .
- $\text{firstUnans}(M, \pi, \langle e_1, \dots, e_n \rangle) = e_i$ iff e_i is the first event in $\langle e_1, \dots, e_n \rangle$ where given π' subpath of π and $M, \pi' \models_{\mathcal{TR}} \mathbf{o}(e)$ then $\neg \exists \pi''$ s.t. π'' is also a subpath of π , π'' is after π' and $M, \pi'' \models \mathbf{r}(e)$.

We continue by exemplifying the semantics in examples.

Example 2

$$\begin{array}{lll} p \leftarrow a.\text{ins} & & p \leftarrow a.\text{ins} \\ \mathbf{r}(e_1) \leftarrow c.\text{ins} & (P_3) & \mathbf{r}(e_1) \leftarrow c.\text{ins} \\ & & \mathbf{o}(e_1) \leftarrow \mathbf{o}(a.\text{ins}) \end{array} \quad (P_4)$$

Consider the programs² P_3 and P_4 . In P_3 , p holds in the path $\langle \{\} \xrightarrow{\mathbf{o}(a.\text{ins})} \{a\} \rangle$. This is true since all interpretations must comply with the oracles and thus $\forall M: a.\text{ins} \in M(\langle \{\} \xrightarrow{\mathbf{o}(a.\text{ins})} \{a\} \rangle)$

² For brevity, in this and the following examples we assume the rule $\mathbf{r}(p) \leftarrow \text{true}$ to appear in every program for every primitive action p defined in the signature of the oracles, unless when stated otherwise. I.e., we assume the responses of events inferred from primitive actions to hold trivially whenever their rules do not appear explicitly in the program.

implying $M, \langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \rangle \models a.ins$. Assuming that M is a model of P_3 , then it satisfies the rule $p \leftarrow a.ins$, which means that $p \in M(\langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \rangle)$ and $M, \langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \rangle \models p$.

However, since $\mathbf{o}(e_1) \leftarrow \mathbf{o}(a.ins) \in P_4$ and $\forall M. \mathbf{o}(a.ins) \in M(\langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \rangle)$, for M to be a model of P_4 , then $\mathbf{o}(e_1) \in M(\langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \rangle)$. Since e_1 has a response defined, then in path $\langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \rangle$ the occurrence e_1 is unanswered and both the transactions p and $a.ins$ cannot succeed in that path. Namely, $\mathbf{o}(e_1)$ constrains the execution of *every* transaction in the path $\langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \rangle$ and, for transaction formulas to succeed, such path needs to be *expanded* with e_1 's response. Since, $\exp_M(\langle \{ \}^a.ins \rightarrow \{a\} \rangle) = \langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \mathbf{o}(c.ins) \rightarrow \{a, c\} \rangle$ then, both transactions p and $a.ins$ succeed in the *longer* path $\langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \mathbf{o}(c.ins) \rightarrow \{a, c\} \rangle$, i.e. for an M model of P_4 : $M, \langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \mathbf{o}(c.ins) \rightarrow \{a, c\} \rangle \models p$ and $M, \langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \mathbf{o}(c.ins) \rightarrow \{a, c\} \rangle \models a.ins$. Notice the non-monotonicity of \mathcal{TR}^{ev} , viz. that adding a new event rule to P_3 falsifies the transaction formulas p and $a.ins$ in paths where they were previously true.

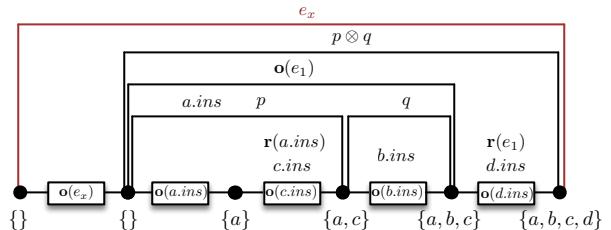
As in \mathcal{TR} , in \mathcal{TR}^{ev} every formula that is meant to be executed, is meant to be executed as a transaction. As such, the primitive $a.ins$ in example P_4 cannot succeed in the path $\langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \rangle$ since there are unanswered events in that path. However, note that $a.ins$ belongs to every interpretation M of that path (due to the restrictions in Def. 1). Thus the primitive $a.ins$ is true in $\langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \rangle$ although the transaction $a.ins$ is not.

Example 3

```

 $p \leftarrow a.ins$ 
 $q \leftarrow b.ins$ 
 $\mathbf{r}(e_x) \leftarrow p \otimes q$ 
 $\mathbf{r}(e_1) \leftarrow d.ins$ 
 $\mathbf{r}(a.ins) \leftarrow c.ins$ 
 $\mathbf{o}(e_1) \leftarrow \mathbf{o}(a.ins) \otimes \mathbf{o}(b.ins)$ 

```



The right-hand side figure illustrates a satisfaction of the external event e_x . The occurrence of e_x forces the satisfaction of the transaction $p \otimes q$, which is true if both its “subformulas” (p and q) are satisfied over smaller paths. Note that, by definition of the relation \models , all occurrences detected over the independent paths that satisfy p and q are already responded in those paths. Thus, we need only to cater for the events triggered due to the serial conjunction. Here, for a model M of the program, $M, \langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \mathbf{o}(c.ins) \rightarrow \{a, c\} \rangle \models p$ and $M, \langle \{a, c\}^{\mathbf{o}(b.ins)} \rightarrow \{a, b, c\} \rangle \models q$. Further, the rule $\mathbf{o}(e_1) \leftarrow \mathbf{o}(a.ins) \otimes \mathbf{o}(b.ins)$ defines one pattern for the occurrence of e_1 which constrains the execution of transaction $p \otimes q$ and forces the expansion of the path to satisfy $\mathbf{r}(e_1)$. Consequently, $M, \langle \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \mathbf{o}(c.ins) \rightarrow \{a, c\} \mathbf{o}(b.ins) \rightarrow \{a, b, c\} \mathbf{o}(d.ins) \rightarrow \{a, b, c, d\} \rangle \models p \otimes q$, and $M, \langle \{ \}^{\mathbf{o}(e_x)} \rightarrow \{ \}^{\mathbf{o}(a.ins)} \rightarrow \{a\} \mathbf{o}(c.ins) \rightarrow \{a, c\} \mathbf{o}(b.ins) \rightarrow \{a, b, c\} \mathbf{o}(d.ins) \rightarrow \{a, b, c, d\} \rangle \models e_x$

3.2 Entailment and Properties

The logical entailment defined in Def. 4 can be used to reason about properties of transaction and event formulas that hold for *every* possible path of execution. In \mathcal{TR}^{ev} , similarly to \mathcal{TR} , we further define executional entailment, to talk about properties of a *particular* execution path. But, to reason about the execution of transactions over a specific path, care must be taken since, as described above, the satisfaction of a new occurrence in a path may invalidate transaction formulas that were previously true.

To deal with a similar behavior, non-monotonic logics rely on the concept of minimal or preferred models: instead of considering all possible models, non-monotonic theories restrict to the most skeptical ones. Likewise, \mathcal{TR}^{ev} uses the minimal models of a program to define entailment, whenever talking about a particular execution of a formula. As usual, minimality is defined by set inclusion on the amount of predicates that an interpretation satisfies, and a minimal model is a model that minimizes the set of formulas that an interpretation satisfies in a path.

Definition 10 (Minimal Model)

Let M_1 and M_2 be interpretations. Then $M_1 \leq M_2$ if $\forall \pi: M_2(\pi) = \top \vee M_1(\pi) \subseteq M_2(\pi)$

Let ϕ be a \mathcal{TR}^{ev} formula, and P a program. M is a *minimal model* of ϕ (resp. P) if M is a model of ϕ (resp. P) and $M \leq M'$ for every model M' of ϕ (resp. P).

Thus, to know if a formula succeeds in a particular path, we need only to consider the event occurrences *supported* by that path, either because they appear as occurrences in the transition of states, or because they are a necessary consequence of the program's rules given that path. Because of this, executional entailment in \mathcal{TR}^{ev} is defined w.r.t. minimal models (cf. Def. 5).

Definition 11 (\mathcal{TR}^{ev} Executional Entailment)

Let P be a program, ϕ a transaction formula and $D_1 \xrightarrow{O_0} \dots \xrightarrow{O_n} D_n$ a path. Then $P, (D_1 \xrightarrow{O_0} \dots \xrightarrow{O_n} D_n) \models \phi$ (\star) iff for every minimal model M of P , $M, \langle D_1 \xrightarrow{O_0} \dots \xrightarrow{O_n} D_n \rangle \models \phi$. $P, D_1 \vdash \phi$ is said to be true, if there is a path $D_1 \xrightarrow{O_0} \dots \xrightarrow{O_n} D_n$ that makes (\star) true.

Interestingly, as in logic programs, formulas satisfied by this entailment have some support.

Lemma 1 (Support)

Let P be a program, π a path, ϕ a transaction atom. Then, if $P, \pi \models \phi$ one of the following holds:

1. ϕ is an elementary action and either $\phi \in \mathcal{O}^d(\pi)$ or $\phi \in \mathcal{O}^t(\pi)$;
2. ϕ is the head of a transaction rule in P ($\phi \leftarrow body$) and $P, \pi \models body$;

As expected, \mathcal{TR}^{ev} extends \mathcal{TR} . Precisely, if a program P has no complex event rules, and for every elementary action a defined by the oracles the only rule for $r(a)$ in P is $r(a) \leftarrow \text{true}$, then executional entailment in \mathcal{TR}^{ev} can be recast in \mathcal{TR} if, \mathcal{TR} executional entailment is also restricted to minimal models. It is worth noting that, for a large class of \mathcal{TR} theories, and namely for the so-called serial-Horn theories, executional entailment in general coincides with that only using minimal models (cf. (Bonner and Kifer 1993)). As an immediate corollary, it follows that if P is *event-free* and serial-Horn, then executional entailment in \mathcal{TR}^{ev} and in \mathcal{TR} coincide.

4 Discussion and Related Work

Several solutions exist to reason about complex events. Complex event processing (CEP) systems as (Adaikkalavan and Chakravarthy 2004; Wu et al. 2006) can reason efficiently with large streams of data and detect (complex) events. These support a rich specification of events based on event pattern rules combining atomic events with some temporal constructs. As shown in Theorem 1, \mathcal{TR} and \mathcal{TR}^{ev} can express most event patterns of SNOOP and, ETALIS (Anicic et al. 2012) CEP system even uses \mathcal{TR} 's syntax and connectives, although abandoning \mathcal{TR} 's model theory and providing a different satisfaction definition. However, in contrast to \mathcal{TR}^{ev} , CEP systems do not deal with the execution of actions in reaction to the events detected.

Extensions of Situation Calculus, Event Calculus, Action Languages, etc. exist with the ability to react to events, and have some transactional properties (Baral et al. 1997; Bertossi et al. 1998).

However, as in database triggers, these events are restricted to detect simple actions like “on insert/delete” and thus have a very limited expressivity that fails to encode complex events, as defined in CEP systems and in \mathcal{TR}^{ev} . To simultaneously reason about actions and complex events, ECA (following the syntax “on *event* if *condition* do *action*”) languages (Alferes et al. 2011; Bry et al. 2006; Chomicki et al. 2003) and logic programming based languages (Kowalski and Sadri 2012; Costantini and Gasperis 2012) exist. These languages normally do not allow the action component of the language to be defined as a transaction, and when they do, they lack from a declarative semantics as (Papamarkos et al. 2006); or they are based on active databases and can only detect atomic events defined as insertions/deletes (Zaniolo 1995; Lausen et al. 1998).

In contrast, \mathcal{TR}^{ev} can deal with arbitrary atomic and complex events, and make these events trigger transactions. This is done by a logic-programming like declarative language. We have also defined a procedure to execute these reactive transactions, which is built upon the complex event detection algorithm of ETALIS and the execution algorithm of \mathcal{TR} , but is omitted for lack of space.

References

- ADAIKKALAVAN, R. AND CHAKRAVARTHY, S. 2004. Formalization and detection of events over a sliding window in active databases using interval-based semantics. In *ADBIS*. 241–256.
- ADAIKKALAVAN, R. AND CHAKRAVARTHY, S. 2006. Snoopib: Interval-based event specification and detection for active databases. *Data Knowl. Eng.* 59, 1, 139–165.
- ALFERES, J. J., BANTI, F., AND BROGI, A. 2011. Evolving reactive logic programs. *Intelligenza Artificiale* 5, 1, 77–81.
- ANICIC, D., RUDOLPH, S., FODOR, P., AND STOJANOVIC, N. 2012. Stream reasoning and complex event processing in etalis. *Semantic Web* 3, 4, 397–407.
- BAILEY, J., DONG, G., AND RAMAMOHANARAO, K. 2004. On the decidability of the termination problem of active database systems. *Theor. Comput. Sci.* 311, 1-3, 389–437.
- BARAL, C., LOBO, J., AND TRAJCEVSKI, G. 1997. Formal characterizations of active databases: Part ii. In *DOOD*. LNCS, vol. 1341. Springer, 247–264.
- BERTOSSI, L. E., PINTO, J., AND VALDIVIA, R. 1998. Specifying active databases in the situation calculus. In *SCCC*. IEEE Computer Society, 32–39.
- BONNER, A. J. AND KIFER, M. 1993. Transaction logic programming. In *ICLP*. 257–279.
- BONNER, A. J. AND KIFER, M. 1998. Results on reasoning about updates in transaction logic. In *Transactions and Change in Logic Databases*. 166–196.
- BONNER, A. J., KIFER, M., AND CONSENS, M. P. 1993. Database programming in transaction logic. In *DBPL*. 309–337.
- BRY, F., ECKERT, M., AND PATRANJAN, P.-L. 2006. Reactivity on the web: Paradigms and applications of the language xchange. *J. Web Eng.* 5, 1, 3–24.
- CHOMICKI, J., LOBO, J., AND NAQVI, S. A. 2003. Conflict resolution using logic programming. *IEEE Trans. Knowl. Data Eng.* 15, 1, 244–249.
- COSTANTINI, S. AND GASPERIS, G. D. 2012. Complex reactivity with preferences in rule-based agents. In *RuleML*. 167–181.
- KOWALSKI, R. A. AND SADRI, F. 2012. A logic-based framework for reactive systems. In *RuleML*. 1–15.
- LAUSEN, G., LUDÄSCHER, B., AND MAY, W. 1998. On active deductive databases: The statelog approach. In *Transactions and Change in Logic Databases*. 69–106.
- PAPAMARKOS, G., POULOVASSILIS, A., AND WOOD, P. T. 2006. Event-condition-action rules on rdf metadata in p2p environments. *Comp. Networks* 50, 10, 1513–1532.
- WU, E., DIAO, Y., AND RIZVI, S. 2006. High-performance complex event processing over streams. In *SIGMOD Conference*. ACM, 407–418.

- ZANIOLO, C. 1995. Active database rules with transaction-conscious stable-model semantics. In *DOOD*. 55–72.