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 (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 TR is a natural choice to deal with transactions, it remains the question whether 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 TR can express complex events, and in particular, how SNOOP event expressions can be translated in the logic. Afterwards, we show why 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 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/BID/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, TRev, an extension of 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 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 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 TRev, its language and model theory (Section 3.1), as well as
its executional semantics (Section 3.2).

2 Using TR to express complex events

In this section we briefly recall 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 TR to deal with reactive features and complex events.

Atoms in TR have the form p(t₁, . . . , tn) where p is a predicate symbol and tᵢ’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, TR uses
the classical connectives ∧, ∨, ¬, ← and the connectives ⊗, ♦ denoting serial conjunction and
hypothetical execution. Informally, φ ⊗ ψ is an action composed of an execution of φ followed
by an execution of ψ; and ♦φ tests if φ can be executed without materializing the changes. In
general, formulas are viewed as (the execution of) transactions, where, φ ∧ ψ is the simultaneous
execution of φ and ψ; φ ∨ ψ the non-deterministic choice of executing φ or ψ. φ ← ψ is a rule
saying that one way to execute of φ is by executing ψ. As in classical logic, ∧ and ← can
be written using ∨ and ¬ (e.g. φ ∧ ψ ≡ ¬(¬φ ∨ ¬ψ)). Finally, we also use the connective ; as it is useful to express common complex events. φ; ψ says that ψ is true after φ but possibly interleaved with other occurrences, and it can be written in TR syntax as: φ ⊗ path ⊗ ψ where path ≡ (φ ∨ ¬φ) 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, TR considers a pair of oracles – Od (data oracle) and Ot (transition oracle) – as a parameter of the theory. These oracles are mappings that assume a set of state identifiers. Od is a mapping from state identifiers to a set of formulas that hold in that state, and Ot 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
Transaction Logic with (Complex) Events

Definition 1

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

1. \( \varphi \in M((D_1)) \) if \( \varphi \in \mathcal{O}^d(D) \)
2. \( \{ \varphi, o(\varphi) \} \subseteq M((D_1 \circ o(\varphi) \rightarrow D_2)) \) if \( \varphi \in \mathcal{O}^i(D_1, D_2) \)

In point 2 we additionally (i.e., when compared to the original definition) force \( o(\varphi) \) to belong to the same path where the primitive action \( \varphi \) 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 \( \pi \) be a \( k \)-path, i.e. a path of length \( k \) of the form \( \langle D_0 \circ O_1 \rightarrow D_1 \circ \ldots O_{k-1} \rightarrow D_k \rangle \). A split of \( \pi \) is any pair of subpaths, \( \pi_1 \) and \( \pi_2 \), such that \( \pi_1 = \langle D_0 \circ O_1 \rightarrow D_i \rangle \) and \( \pi_2 = \langle D_i \circ O_{i+1} \rightarrow D_k \rangle \) for some \( i \) (1 \( \leq \) \( i \) \( \leq \) \( k \)). In this case, we write \( \pi = \pi_1 \circ \pi_2 \).

A subpath \( \pi' \) of \( \pi \) is any subset of states and annotations of \( \pi \) where both the order of the states and their annotations is preserved. A prefix \( \pi_1 \) of \( \pi \) is any subpath of \( \pi \) sharing the initial state.

\footnote{For not having to consider partial mappings, besides formulas, interpretations can also return the special symbol \( \top \). The interested reader is referred to \cite{Bonner and Kifer 1993} for details.}
Definition 3 (TR Satisfaction of Complex Formulas)
Let $M$ be an interpretation, $\pi$ a path and $\phi$ a formula. If $M(\pi) = \top$ then $M, \pi \models TR \phi$; else:

1. **Base Case:** $M, \pi \models TR \phi$ iff $\phi \in M(\pi)$ for every event occurrence $\phi$

2. **Negation:** $M, \pi \models TR \neg \phi$ iff it is not the case that $M, \pi \models TR \phi$

3. **Disjunction:** $M, \pi \models TR \phi \lor \psi$ iff $M, \pi \models TR \phi$ or $M, \pi \models TR \psi$

4. **Serial Conjunction:** $M, \pi \models TR \phi \otimes \psi$ iff there exists a split $\pi_1 \circ \pi_2$ of $\pi$ s.t. $M, \pi_1 \models TR \phi$ and $M, \pi_2 \models TR \psi$

5. **Executional Possibility:** $M, \pi \models TR \Diamond \phi$ iff $\phi$ is a 1-path of the form $\langle D \rangle$ for some state $D$ and $M, \pi' \models TR \phi$ for some path $\pi'$ 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 $\phi$ iff for every path $\pi$, $M, \pi \models TR \phi$. $M$ is a model
of a set of rules $P$ (denoted $M \models TR P$) iff it is a model of every rule in $P$.

$\phi$ is said to logically entail another formula $\psi$ iff every model of $\phi$ is also a model of $\psi$.

Logical entailment is useful to define general equivalence and implication of formulas that
express properties like “transaction $\phi$ is equivalent to transaction $\psi$” or “whenever transaction $\psi$
is executed, $\psi'$ 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 TR as:

$$o(alarm) \leftarrow (o(e_2) \land o(e_3)) ; o(e_1) \tag{1}$$

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

Other complex event definitions are possible, and in fact we can encode most of SNOOP
operators in 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 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 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
true event in the path $\langle s_i, s_{i+1} \rangle$.

Theorem 1 (SNOOP Algebra and 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 $e_j[t_1]$ that have occurred over the time interval
$t_1, t_{max}$, and $\langle s_1, \ldots, s_{max+1} \rangle$ be a path with size $t_{max} - t_1 + 1$. Let $\tau$ be the following function:

**Primitive:** $\tau(E) = o(E)$ where $E$ is a primitive event

**Sequence:** $\tau(E_1 \bullet E_2) = \tau(E_1) \otimes \text{path} \otimes \tau(E_2)$

**Or:** $\tau(E_1 \lor E_2) = \tau(E_1) \lor \tau(E_2)$

**AND:** $\tau(E_1 \land E_2) = \tau(E_1 \land \text{path}) \lor (\text{path} \otimes \tau(E_2))$ \lor (\text{path} \otimes \tau(E_1))$

**NOT:** $\tau(\neg E_i(E_1 \bullet E_2)) = \tau(E_1) \otimes \neg \tau(E_i) \otimes \tau(E_2)$
Define $E[H]$ as 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 $e_j[t_i] \in H$: $M, \langle s_{t_i}, s_{t_{i+1}} \rangle \models E O(e_j)$.

Besides the logical entailment, $TR$ 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, $\phi$ a formula, and $D_0 O_1 \ldots O_n \rightarrow D_n$ a path.

$P, (D_0 O_1 \ldots O_n \rightarrow D_n) \models (\star)$ iff for every model $M$ of $P, M, (D_0 O_1 \ldots O_n \rightarrow D_n) \models \phi$.

Additionally, $P, D_0 \models \phi$ holds, if there is a path $D_0 O_1 \ldots O_n \rightarrow D_{move}$ that makes $(\star)$ true.

This entailment has a corresponding proof theory (Bonner and Kifer 1993) which, for a subset of $TR$, is capable of constructing such a path given a program, a $TR$ 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 $TR^{ev}$: 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, $TR$ 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 $TR$ as:

$$p \leftarrow body \otimes ev$$

$$ev \leftarrow r(ev)$$

(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, $r(ev)$, is executed. Note that, both $r(ev)$ 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/and delete"). Triggering external events in $TR$ 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 $\pi$ 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:

$$\text{o(alarm)} \leftarrow (\text{o}(e_2) \land \text{o}(e_3)); \text{o}(e_1)$$

$$\text{r(alarm)} \leftarrow a_1 \otimes a_2$$

But this does not drive the execution of r(alarm) when o(alarm) holds; one has further to force that whenever o(alarm) holds, r(alarm) must be made true subsequently. Of course, adding a rule r(alarm) $\leftarrow$ 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 o(alarm) to make 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 $\mathcal{T}R_{\text{ev}}$. This extension of $\mathcal{T}R$ evaluates event formulas and transaction formulas differently, using two distinct relations (respectively $|=_{\mathcal{T}R}$ and $|=\), and occurrences and responses are syntactic represented w.r.t. a given event name $e$, as $o(e)$ and $r(e)$, respectively. In this context, $|=\) requires transactions to be satisfied in expanded paths, where every occurring event (made true by $|=_{\mathcal{T}R}$) is properly responded.

### 3.1 $\mathcal{T}R_{\text{ev}}$ Syntax and Model Theory

To make possible a different evaluation of events and transactions, predicates in $\mathcal{T}R_{\text{ev}}$ are partitioned into transaction names ($P_t$), event names ($P_e$), and oracle primitives ($P_\text{o}$) and, as with $\mathcal{T}R$, we work with the Herbrand instantiation of the language.

Formulas in $\mathcal{T}R_{\text{ev}}$ 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 $\neg \phi$, $\phi \land \psi$, $\phi \lor \psi$, $\phi \otimes \psi$, or $\phi; \psi$ where $\phi$ and $\psi$ are event formulas. An event occurrence is of the form $o(\varphi)$ s.t. $\varphi \in P_e$ or $\varphi \in P_\text{o}$. 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 $\neg \phi$, $\diamond \phi$, $\phi \land \psi$, $\phi \lor \psi$, or $\phi \otimes \psi$. A transaction atom is either a transaction name (in $P_t$), an oracle defined primitive (in $P_\text{o}$), the response to an event (r($\varphi$) where $\varphi \in P_\text{o} \cup P_e$), or an event name (in $P_e$). 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 $\varphi$ is a transaction atom and $\psi$ a transaction formula; in an event rule $\varphi$ is an event occurrence and $\psi$ is a event formula. A program is a set of transaction and event rules.

Importantly, besides the data and transition oracles, $\mathcal{TR}^{cu}$ 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, $\mathcal{TR}^{cu}$ does not commit to any particular definition, encapsulating it in function choice.

As a reactive system, $\mathcal{TR}^{cu}$ receives a series of external events which may cause the execution of transactions in response. This is defined as $P, D_0 \models e_1 \otimes \ldots \otimes e_k$, where $D_0$ is the initial KB state and $e_1 \otimes \ldots \otimes e_k$ is the sequence of external events that arrive to the system. Here, we want to find the path $D_0 \mathcal{O}_1 \rightarrow \ldots \mathcal{O}_n \rightarrow D_n$ encoding a KB evolution that responds to $e_1 \otimes \ldots \otimes e_k$.

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 ($\mathcal{TR}^{cu}$ interpretations)**

A $\mathcal{TR}^{cu}$ interpretation is a $\mathcal{TR}$ interpretation that additionally satisfies the restriction: 3) $\mathcal{O}(e) \in M((D^{\mathcal{O}(e)} \rightarrow D))$ 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, $\pi$ a path, $\phi$ transaction formula. If $M(\pi) = \top$ then $M, \pi \models \phi$; else:

1. **Base Case:** $M, \pi \models p$ iff $\exists \pi'$ prefix of $\pi$ s.t. $p \in M(\pi')$ and $\pi = \exp_M(\pi')$, for every transaction atom $p$ where $p \not\in \mathcal{P}_e$.
2. **Event Case:** $M, \pi \models e$ iff $e \in \mathcal{P}_e$, $\exists \pi'$ prefix of $\pi$ s.t. $M, \pi' \models_{\mathcal{TR}} \mathcal{O}(e)$ 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:** $M, \pi \models \phi \lor \psi$ iff $M, \pi \models \phi$ or $M, \pi \models \psi$.
5. **Serial Conjunction:** $M, \pi \models \phi \otimes \psi$ iff $\exists \pi'$ prefix of $\pi$ and some split $\pi_1 \circ \pi_2$ of $\pi'$ 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 $\pi$ is a 1-path of the form $\langle D \rangle$ for some state $D$ and $M, \pi' \models \phi$ for some path $\pi'$ that begins at $D$.

An interpretation $M$ is a model of a transaction formula (resp. event formula) $\phi$ if for every path $\pi$, $M, \pi \models \phi$ (resp. $M, \pi \models_{\mathcal{TR}} \phi$). $M$ is a model of a program $P$ (denoted $M \models P$) if 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 perforce 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 \( \pi_1 \) and an interpretation \( M \), the response operator \( R_M(\pi_1) \) is defined as follows:

\[
R_M(\pi_1) = \begin{cases} 
\pi_1 \circ \pi_2 & \text{if } \text{choice}(M, \pi_1) = e \text{ and } M, \pi_2 \models r(e) \\
\pi_1 & \text{if } \text{choice}(M, \pi_1) = \epsilon 
\end{cases}
\]

The expansion of a path \( \pi \) is \( \exp_M(\pi) = \uparrow R_M(\pi) \).

In general it may not be possible to address all events in a finite path, and thus, \( R_M \) 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 \( R_M \) is an approximation of the expansion operator \( \exp_M \).

This definition leaves open the choice function, that is taken as a further parameter of \( TR^w \), 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 on 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 \( \pi \) be a path. The temporal choice function is \( \text{choice}(M, \pi) = \text{firstUnans}(M, \pi, \text{order}(M, \pi)) \) where:

1. \( \text{order}(M, \pi) = (e_1, \ldots, e_n) \) iff \( \forall e_i, 1 \leq i \leq n, \exists \pi_i \) subpath of \( \pi \) where \( M, \pi \models_{TR} o(e_i) \) and \( \forall e_j \) s.t. \( i < j \) then \( e_j \) occurs after \( e_i \)
2. \( e_2 \) occurs after \( e_1 \) w.r.t. \( \pi \) and \( M \) iff there exists \( \pi_1, \pi_2 \) subpaths of \( \pi \) such that \( \pi_1 = (D_1 \circ O_1 \circ \ldots \circ O_{m-1} \circ D_m), \pi_2 = (D_n \circ O_n \circ \ldots \circ O_{m-1} \circ D_m), M, \pi_1 \models o(e_1), M, \pi_2 \models o(e_2) \) and \( D_j \leq D_m \) w.r.t. the ordering in \( \pi \).
3. \( \text{firstUnans}(M, \pi, (e_1, \ldots, e_n)) = e_i \) iff \( e_i \) is the first event in \( (e_1, \ldots, e_n) \) where given \( \pi' \) subpath of \( \pi \) and \( M, \pi' \models_{TR} o(e) \) then \( \exists \pi'' \) s.t. \( \pi'' \) is also a subpath of \( \pi, \pi'' \) is after \( \pi' \) and \( M, \pi'' \models r(e) \).

We continue by exemplifying the semantics in examples.

Example 2

\[
\begin{align*}
p &\leftarrow a.\text{ins} & (P_2) & \quad p &\leftarrow a.\text{ins} \\
r(e_1) &\leftarrow c.\text{ins} & (P_3) & \quad r(e_1) &\leftarrow c.\text{ins} \\
o(e_1) &\leftarrow o(\text{a.ins}) & (P_4) & \quad o(e_1) &\leftarrow o(\text{a.ins})
\end{align*}
\]

Consider the programs \( P_2 \) and \( P_3 \). In \( P_2, \) \( p \) holds in the path \( \{\{\} o_{\text{a.ins}} \rightarrow \{a\}\} \). This is true since all interpretations must comply with the oracles and thus \( \forall M: a.\text{ins} \in M(\{\{\} o_{\text{a.ins}} \rightarrow \{a\}\}) \)

\( ^2 \) For brevity, in this and the following examples we assume the rule \( r(p) \leftarrow \text{true} \) 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 \( M, \langle \{ \} \circ(a.\text{ins}) \rightarrow \{ a \} \rangle \models a.\text{ins} \). Assuming that \( M \) is a model of \( \mathcal{P} \), then it satisfies the rule \( p \leftarrow a.\text{ins} \), which means that \( p \in M(\langle \{ \} \circ(a.\text{ins}) \rightarrow \{ a \} \rangle) \) and \( M, \langle \{ \} \circ(a.\text{ins}) \rightarrow \{ a \} \rangle \models p \).

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

As in \( \mathcal{T} \mathcal{R} \), in \( \mathcal{T} \mathcal{R}^\text{ev} \) every formula that is meant to be executed, is meant to be executed as a transaction. As such, the primitive \( a.\text{ins} \) in example \( \mathcal{P} \) cannot succeed in the path \( \langle \{ \} \circ(a.\text{ins}) \rightarrow \{ a \} \rangle \) since there are unanswerable events in that path. However, note that \( a.\text{ins} \) belongs to every interpretation \( M \) of that path (due to the restrictions in Def. \( \mathbb{I} \)). Thus the primitive \( a.\text{ins} \) is true in \( \{ \} \circ(a.\text{ins}) \rightarrow \{ a \} \) although the transaction \( a.\text{ins} \) is not.

**Example 3**

\[
\begin{aligned}
p &\leftarrow a.\text{ins} \\
q &\leftarrow b.\text{ins} \\
r(e_x) &\leftarrow p \otimes q \\
r(e_1) &\leftarrow d.\text{ins} \\
o(e_1) &\leftarrow o(a.\text{ins}) \otimes o(b.\text{ins})
\end{aligned}
\]

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, \( M, \langle \{ \} \circ(a.\text{ins}) \rightarrow \{ a \} \circ(c.\text{ins}) \rightarrow \{ a, c \} \rangle \models p \) and \( M, \langle \{ \} \circ(a.\text{ins}) \rightarrow \{ a, c \} \circ(b.\text{ins}) \rightarrow \{ a, b, c \} \rangle \models q \). Further, the rule \( o(e_1) \leftarrow o(a.\text{ins}) \otimes o(b.\text{ins}) \) 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 \( r(e_1) \). Consequently, \( M, \langle \{ \} \circ(a.\text{ins}) \rightarrow \{ a \} \circ(c.\text{ins}) \rightarrow \{ a, c \} \circ(d.\text{ins}) \rightarrow \{ a, b, c, d \} \rangle \models p \otimes q \), and \( M, \langle \{ \} \circ(e_x) \rightarrow \{ \} \circ(a.\text{ins}) \rightarrow \{ a \} \circ(c.\text{ins}) \rightarrow \{ a, c \} \circ(b.\text{ins}) \rightarrow \{ a, b, c \} \circ(d.\text{ins}) \rightarrow \{ a, b, c, d \} \rangle \models e_x \).

### 3.2 Entailment and Properties

The logical entailment defined in Def. \( \mathbb{I} \) can be used to reason about properties of transaction and event formulas that hold for every possible path of execution. In \( \mathcal{T} \mathcal{R}^\text{ev} \), similarly to \( \mathcal{T} \mathcal{R} \), 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, \( \mathcal{T}R^{ev} \) 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 \( \forall \pi: M_2(\pi) = \top \lor M_1(\pi) \subseteq M_2(\pi) \)

Let \( \phi \) be a \( \mathcal{T}R^{ev} \) formula, and \( P \) a program. \( M \) is a minimal model of \( \phi \) (resp. \( P \)) if \( M \) is a model of \( \phi \) (resp. \( P \)) and \( M \leq M' \) for every model \( M' \) of \( \phi \) (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 \( \mathcal{T}R^{ev} \) is defined w.r.t. minimal models (cf. Def. 5).

**Definition 11 (\( \mathcal{T}R^{ev} \) Executional Entailment)**

Let \( P \) be a program, \( \phi \) a transaction formula and \( D_1 \overset{O_1}{\rightarrow} \ldots \overset{O_n}{\rightarrow} D_n \) a path. Then \( P, (D_1 \overset{O_1}{\rightarrow} \ldots \overset{O_n}{\rightarrow} D_n) \models \phi \) (\( \ast \)) iff for every minimal model \( M \) of \( P \), \( M, (D_1 \overset{O_1}{\rightarrow} \ldots \overset{O_n}{\rightarrow} D_n) \models \phi \).

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

**Lemma 1 (Support)**

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

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

As expected, \( \mathcal{T}R^{ev} \) extends \( \mathcal{T}R \). 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 \( \mathcal{T}R^{ev} \) can be recast in \( \mathcal{T}R \) if, \( \mathcal{T}R \) executional entailment is also restricted to minimal models. It is worth noting that, for a large class of \( \mathcal{T}R \) 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 \( \mathcal{T}R^{ev} \) and in \( \mathcal{T}R \) 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, \( \mathcal{T}R \) and \( \mathcal{T}R^{ev} \) can express most event patterns of SNOOP and, ETALIS (Anicic et al. 2012) CEP system even uses \( \mathcal{T}R \)’s syntax and connectives, although abandoning \( \mathcal{T}R \)’s model theory and providing a different satisfaction definition. However, in contrast to \( \mathcal{T}R^{ev} \), 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 †TRev. 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, †TRev 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 †TR, but is omitted for lack of space.

References

Adakkalavan, 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.

Adakkalavan, R., and Chakravarthy, S. 2006. Snoopy: 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.

Bailley, 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 Patranchal, 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., Ludescher, B., and May, W. 1998. On active deductive databases: The statelog approach. In Transactions and Change in Logic Databases. 69–106.

Papamarkos, G., Poulavassilis, 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.