Extending Transaction Logic with External Actions

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

We provide a complete formalization of External Transaction Logic (or $\mathcal{ETR}$), an extension of Transaction Logic to reason and execute external actions, i.e. actions performed on an external domain of which one has limited or no control; while interacting with an internal knowledge base compliant with ACID transactions. In such complex environment, if a failure occurs, it is no longer possible to simply rollback to the initial state prior to the execution of the transaction. Since rollbacking of external actions is unfeasible, as in databases, a relaxed transaction model can still be achieved by defining compensating operations that revert each of the external actions performed. To reason about external actions, we augment Transaction Logic's theory with the ability to consult an external oracle, thereby allowing the abstraction from the theory and semantics of the external domain.

In this paper we define the semantics of $\mathcal{ETR}$, prove the equivalence to Transaction Logic when no external actions are defined, and construct a sound and complete SLD-style proof theory for a Horn-like subset of the logic. The solution obtained extends the semantics and proof theory of traditional Logic Programming both with the ability to perform ACID updates and to interact with an external domain.

1 Introduction

Transaction Logic ($\mathcal{TR}$) is an extension of predicate logic proposed in (Bonner and Kifer 1995) able to integrate both *reasoning* and *execution* of transaction programs in one unified framework. While reasoning is achieved through a model-theoretic semantics that enables $\mathcal{TR}$ to study properties as equivalence and implication of transactions; execution is provided by a proof theory that, being sound and complete with the semantics, can answer practical questions like "can this action be executed in this state" or "how does my database evolve if this action is executed".

$\mathcal{TR}$ is flexible in the sense that it does not commit to any particular semantics of state change. This is obtained through a parameterization of the logic by a pair of oracles defining the elementary operations that update and query the knowledge base (KB). Operations like "insert(p)" or "delete(p)" are encapsulated by these oracles, enabling $\mathcal{TR}$ to reason and execute transactions while accommodating a wide variety of state change semantics (as relational databases, well-founded semantics, first-order logic or other non-standard semantics). Due to these characteristics, $\mathcal{TR}$ has proven to be a powerful tool for reasoning about actions (Rezk and Kifer 2012), argumentation theories (Fodor and Kifer 2011), AI planning (Bonner and Kifer 1995), workflow management and semantic web services (Roman and Kifer 2007), databases (Bonner et al. 1993), and general KB representation (Bonner and Kifer 1994).

∗ The authors thank Michael Kifer, Paul Fodor and the Stony Brook group for their valuable comments in a preliminary version of this work. The first author was supported by FCT grant SFRH/BD/64038/2009. The work was partially supported by project ERRO (PTDC/EIA-CCO/121823/2010).

Unlike many other logic systems, $\mathcal{TR}$ imposes that the KB evolves only into consistent states respecting ACID properties (Atomicity, Consistency, Isolation and Durability), as it is desired in databases. As an inherent consequence, although $\mathcal{TR}$ can reason about actions and their effects in arbitrary logic theories, $\mathcal{TR}$ requires a complete control and specification of the KB on which these actions are executed. Since every action is a strict ACID transaction, $\mathcal{TR}$ fails to encode richer situations that involve interaction and execution of actions in external domains, besides an internal database. The problem about external domains is that it is no longer possible to ensure the same transaction ACID model as in the internal KB. In fact, since one does not control the external environment on which these actions were executed, rollback of external actions is impossible. However, in case of a transaction failure, if some external action was previously performed, something should be done to preserve some kind of consistency in the external KB.

Example 1
Consider the following $\mathcal{TR}$ program P (where $\otimes$ denotes serial conjunction):

$$t \leftarrow p.ins \otimes external\_a \otimes external\_b$$
$$t \leftarrow q.ins \otimes external\_c$$

Just like in logic programs, transaction t has two different ways to succeed: if the insertion of predicate p followed by the actions $external\_a$ and $external\_b$ succeeds; or if the insertion of predicate q followed by $external\_c$ succeeds. Let's consider $external\_a$, $external\_b$ and $external\_c$ as actions performed in an external domain and that $external\_b$ fails after the execution of $external\_a$. In $\mathcal{TR}$, t is only satisfied when $external\_c$ is performed after $q.ins$ (2nd rule). But if the 1st rule was "tried" previously, then the actions $p.ins$ and $external\_a$ are meant to be rolled back, and this execution is considered to have never happened. However, as it was performed in an external domain of which $\mathcal{TR}$ does not control, it may not be possible to roll back $external\_a$. E.g. if we are in the scenario of a travel agency and $external\_a$ is e.g. the web-service request of booking an hotel room, then we cannot simply rollback our request. Yet, something must be done to restore consistency or the agency will be charged for an unused room.

In database literature this kind of problems is solved using long-running transactions (Garcia-Molina and Salem 1987). To deal with the impediment of rollbacking, the usual approach is to define compensating operations for each external action to be executed. If each compensation reverts the original action specified, by executing the compensations of the external actions performed in backward order, we obtain an external state considered equivalent to the initial one, achieving a relaxed model of consistency and atomicity externally. E.g., in the example 1, one could define the compensation of the booking of the room ($external\_a$), as an action of canceling the room ($external\_a^{-1}$). Here, the transaction t would also succeed when $external\_a$ and then $external\_a^{-1}$ are performed, followed by the insertion q and the execution of $external\_c$.

To reason and execute external actions, $\mathcal{ETR}$ extends $\mathcal{TR}$ and includes an additional external oracle to define the elementary operations of the external domain. $\mathcal{ETR}$ is thus partitioned into an internal and external KB. While actions performed in the internal KB follow the strict ACID model, actions executed externally follow a relaxed model based on compensations. As in $\mathcal{TR}$, assuming this external oracle allows $\mathcal{ETR}$ to reason and execute transactions that require interaction with external sources without committing to any semantics for the external KB. Logics that talk about the dynamics of external worlds like Action Languages (Gelfond and Lifschitz 1998), Event Calculus (Kowalski and Sergot 1986) or Situation Calculus (McCarthy 1963) are obvious candidates to instantiate the external oracle. However, other non-standard logics are possible.

$\mathcal{ETR}$ is useful for reasoning and executing transactions in systems that comprise two different domains: an internal domain of which one as an absolute control and where updates compliant with the ACID model can be perfomed; and an external domain of which one does not fully control and thus only a weaker model of transaction properties can be achieved. This is e.g. the case of an intelligent agent perfoming actions in an outside world while also dealing with an internal KB (e.g. containing preferences, rules of behavior and its own state) that evolves according to the ACID model of databases.

In this paper we define the language (Section 2), model theory (Section 3) and execution model (Sections 4 and 5) of $\mathcal{ETR}$, presenting some fundamental properties.

2 Syntax

$\mathcal{ETR}$ operates over a KB which includes both an internal and an external component. For that, formally $\mathcal{ETR}$ works over two disjoint propositional languages: $\mathcal{L}_P$ (program language), and $\mathcal{L}_\mathcal{O}$ (oracles primitives language). Propositions in $\mathcal{L}_P$ denote action and fluent names that can be defined in the program. As usual, fluents are propositions that can be evaluated without changing the state and actions are propositions that cause evolution of states. Propositions in $\mathcal{L}_\mathcal{O}$ define the primitive actions and queries to deal with the internal and external KB. $\mathcal{L}_\mathcal{O}$ can still be partitioned into $\mathcal{L}_i$ and $\mathcal{L}_a$, where $\mathcal{L}_i$ denote primitives that query and change the internal KB, while $\mathcal{L}_a$ denote the external actions primitives that can be executed externally. For convenience, we assume that $\mathcal{L}_a$ always contains two distinct actions `failop` and `nop`, the former that always fails in any external domains, and the latter that always succeeds without changing anything. Moreover, we also define $\mathcal{L}_a^*$ as the result of augmenting $\mathcal{L}_a$ with expressions $\mathbf{ext}(a, b)$, where $a, b \in \mathcal{L}_a$. Such an expression defines an external action a that has b as compensating action.

To build complex logical formulas, $\mathcal{ETR}$ uses the standard logic connectives $\wedge, \vee, \neg, \leftarrow$ plus the connective $\otimes$ from $\mathcal{TR}\otimes$ where $\phi \otimes \psi$ represents the action composed by an execution of ϕ followed by an execution of ψ.

Definition 1 (*$\mathcal{ETR}$ atoms, formulas and programs*)
An *$\mathcal{ETR}$ atom* is either a proposition in $\mathcal{L}_P$, $\mathcal{L}_i$ or $\mathcal{L}_a^*$ and an $\mathcal{ETR}$ literal is either ϕ or $\neg\phi$ where ϕ is an $\mathcal{ETR}$ atom. An *$\mathcal{ETR}$ formula* is either a literal, or an expression, defined inductively, of the form $\phi \wedge \psi$, $\phi \vee \psi$ or $\phi \otimes \psi$, where ϕ and ψ are $\mathcal{ETR}$ formulas.
An *$\mathcal{ETR}$ program is a set of rules of the form $\phi \leftarrow \psi$ where ϕ is a proposition in $\mathcal{L}_P$ and ψ is an $\mathcal{ETR}$ formula*.

We also use logical variables, that are to be understood as universally quantified. This is an abuse of the language, that formally stands for the Herbrand instantiations of the formulas. This use of Herbrand instantiations is done without loss of generality, in a way similar to what is done in $\mathcal{TR}$ (cf. (Bonner and Kifer 1998b)). As usual, the Herbrand base $\mathcal{B}$ is a set of all ground atomic formulas in the language; and a classical Herbrand structure is any subset of $\mathcal{B}$.

2.1 States, Operations and Oracles

As in $\mathcal{TR}$, both the language and the semantics of $\mathcal{ETR}$ are parameterized by a set of oracles to reason about basic actions and queries. This allows for the separation of elementary operations from the logic of combining them, giving it a powerful flexibility. So, the language itself is not fixed and $\mathcal{ETR}$ can incorporate a wide variety of KB semantics (Bonner and Kifer 1995).

In $\mathcal{TR}$ there is a notion of *state identifiers*, that uniquely identify the state of the internal KB on which a $\mathcal{TR}$ theory operates. Here, two oracles are considered: the *state data oracle* $\mathcal{O}^d$, and *state transition oracle* $\mathcal{O}^t$. The former maps state identifiers (or simply "states", for short) into formulas in $\mathcal{L}_i$ and, $\mathcal{O}^d(D) \models \varphi$ denotes that φ is true in the state D. $\mathcal{O}^t$ maps pairs of states into sets of formulas, where $\mathcal{O}^t(D_1, D_2) \models \varphi$ means that φ is true in the transition D_1 into D_2.

A simple example of these states and oracles from (Bonner and Kifer 1998a), is the formalization of relation databases. In it, a state D is a set of ground atoms where $\mathcal{O}^d(D) = D$. Moreover, for every predicate p in D, there are new predicates $p.ins$ and $p.del$ respectively denoting insertion and deletion of atom p, where $p.ins \in \mathcal{O}^t(D_1, D_2)$ iff $D_2 = D_1 \cup \{p\}$ and $p.del \in \mathcal{O}^t(D_1, D_2)$ iff $D_2 = D_1 - \{p\}$. Other oracles examples, such as based in first-order logic, well-founded semantics or scientific oracles can be found in (Bonner and Kifer 1998a).

Now, since $\mathcal{ETR}$ is meant to operate on both an internal KB and external domain, two disjoint sets of state identifiers are needed: one for internal states, and another for uniquely identifying states of the external domain (external states). The two oracles of $\mathcal{TR}$ operate, obviously, with internal states. In $\mathcal{ETR}$ an additional oracle is needed to evaluate elementary external operations: the *external oracle* $\mathcal{O}^e$. This oracle is a mapping from a pair of external states into formulas in $\mathcal{L}_a^*$ and, similarly to $\mathcal{O}^t$, we write $\mathcal{O}^e(E_1, E_2) \models \varphi$.

Since we stipulated that the actions `failop` and `nop` always belong to $\mathcal{L}_a^*$ with a precise meaning, we force that for every external oracle and every pair of external states $\mathcal{O}^e(E_1, E_2) \not\models$ `failop` and $\mathcal{O}^e(E_1, E_1) \models$ `nop` (i.e. `failop` always fails, and `nop` always succeed leaving the state unchanged, as desired). External actions with compensations, $\mathbf{ext}(a, b)$, are evaluated by the external oracle solely according to what is known about a, i.e. $\mathcal{O}^e(E_1, E_2) \models \mathbf{ext}(a, b)$ iff $\mathcal{O}^e(E_1, E_2) \models a$ for any $a, b \in \mathcal{L}_a$. Thus, as expected, it is not the task of the oracle (but rather of the $\mathcal{ETR}$ semantics, as we shall see) to deal with compensations. Note that, contrarily to the internal KB, we do not distinguish between external queries and actions as both are evaluated by $\mathcal{O}^e$ and both force a state transition. Thus, external queries can be seen as external actions where the new state transition stores that the given query was done.

The external oracle abstracts the theory and semantics of the external domain, encapsulating the elementary operations that can be performed externally. In the sequel we see this oracle as a black box, that encodes the behavior of the external domain in a completely independent way. However, if one wants to reason about both the internal and the external KB, one can fix $\mathcal{O}^e$ by formalizing it with some logic for describing external worlds, such as, e.g., Action Languages (Gelfond and Lifschitz 1998), Event Calculus (Kowalski and Sergot 1986) or Situation Calculus (McCarthy 1963). For example, one can define an external state as a Situation Calculus theory T plus a specific situation S in the theory, and define $\mathcal{O}^e$ as:

- $\mathcal{O}^e((T, S), (T, S)) \models f$ iff $T \models_{SitCal} Holds(f, S)$
- $\mathcal{O}^e((T, S), (T, do(S))) \models a$ iff $T \models_{SitCal} Poss(a, S)$

3 Model Theory

Satisfaction of formulas in $\mathcal{ETR}$ means *execution*: a formula is said to be true if it can be executed successfully. As a result, contrarily to most logics of state change, formulas are not evaluated on states but on *paths*, i.e. sequence of states (for convenience, we also record the operations performed between states). A formula is satisfied in a path if that path is a valid execution trace for that formula.

Definition 2 (*States and Paths*)
An $\mathcal{ETR}$ *state* S is a pair (D, E) where D and E are, respectively, internal and external states. A *path* of length k, or a *k-path*, is a finite sequence of states, $S_1, ^{A_1} \ldots, ^{A_{k-1}} S_k$ where A_is $(1 \leq i < k)$ are atoms from $\mathcal{L}_a^*$ or $\mathcal{L}_i$.

An interpretation determines what atoms are true on what paths by defining mappings from paths to a Herbrand structures. If $\phi \in M(\pi)$ then, in the interpretation M, path π is a valid execution for the formula ϕ. Interpretations need to be compliant with the specified oracles. The oracles define elementary primitives for the internal and external KB which all interpretations must model. By limiting the set of possible interpretations, these restrictions force the satisfaction of primitive formulas only in the paths that the oracles define it so.

Definition 3 (*Interpretations*)
An interpretation is a mapping M assigning a classical Herbrand structure (or $\top$[1]) to every path. This mapping is subject to the following restrictions, for all states D_i,E_j and every formula φ:

1. $\varphi \in M(\langle (D, E) \rangle)$ iff $\mathcal{O}^d(D) \models \varphi$ for any external state E
2. $\varphi \in M(\langle (D_1, E), ^{\varphi} (D_2, E) \rangle)$ iff $\mathcal{O}^t(D_1, D_2) \models \varphi$ for any external state E
3. $\varphi \in M(\langle (D, E_1), ^{\varphi} (D, E_2) \rangle)$ iff $\mathcal{O}^e(E_1, E_2) \models \varphi$ for any internal state D

General satisfaction of $\mathcal{ETR}$ formulas over paths, requires the prior definition of operations on paths. For example, in $\mathcal{TR}$ the formula $\phi \otimes \psi$ is true (i.e. successfully executes) in a path that executes ϕ up to some point in the middle, and executes ψ from then onwards. To deal with this:

Definition 4 (*Path Splits*)
A *split* of a k-path π is any pair of subpaths, π_1 and π_2, such that $\pi_1 = \langle S_1, ^{A_1} \ldots, ^{A_{i-1}} S_i \rangle$ and $\pi_2 = \langle S_i, ^{A_i} \ldots, ^{A_{k-1}} S_k \rangle$ for some i $(1 \leq i \leq k)$. In this case, we write $\pi = \pi_1 \circ \pi_2$.

The definition of satisfaction of the standard $\mathcal{TR}$ can now easily be generalized to the notion of path in $\mathcal{ETR}$, where states are pairs with the internal and external state:

Definition 5 (*Classical Satisfaction*)
Let M be an interpretation, π a path and ϕ a formula. If $M(\pi) = \top$ then $M, \pi \models_c \phi$; otherwise:

1. **Base Case:** $M, \pi \models_c \phi$ iff $\phi \in M(\pi)$ for any atom ϕ
2. **Negation:** $M, \pi \models_c \neg\phi$ iff it is not the case that $M, \pi \models_c \phi$
3. **"Classical" Disjunction:** $M, \pi \models_c \phi \vee \psi$ iff $M, \pi \models_c \phi$ or $M, \pi \models_c \psi$.
4. **"Classical" Conjunction:** $M, \pi \models_c \phi \wedge \psi$ iff $M, \pi \models_c \phi$ and $M, \pi \models_c \psi$.
5. **Serial Conjunction:** $M, \pi \models_c \phi \otimes \psi$ iff $M, \pi_1 \models_c \phi$ and $M, \pi_2 \models_c \psi$ for some split $\pi_1 \circ \pi_2$ of path π.

This satisfaction, coming from $\mathcal{TR}$, does not consider the possibility of failure. Since $\mathcal{ETR}$ allows external actions as transaction formulas, it must take into the account the possibility of a transaction to fail. Viz. if a failure occurs after the execution of some external actions, then we need to execute some compensating operations to invert the external actions already performed and recover a consistent state in the external KB.

The partial satisfaction relation below is the first ingredient to deal with such failures. The idea is to, given a path, determine the formulas that either "completely" succeed, or at least succeed up to some point and then fail to execute some action or query.

[1] Similar to $\mathcal{TR}$, for not having to consider partial mappings, besides formulas, interpretation can also return the special symbol $\top$. The interested reader is referred to (Bonner and Kifer 1998b) for details.

Example 2 (*Running Example*)
Recall example 1, but where external expressions like $external\_a$ are replaced by actions with compensation like $\mathbf{ext}(a, a^{-1})$. Moreover, the internal KB is a relational database formalized as explained in Section 2.1, and the external oracle includes: $\mathcal{O}^e(E_1, E_2) \models a$, (i.e. the external execution of a in state E_1 succeeds, and makes the external world evolve into E_2), $\mathcal{O}^e(E_1, E_4) \models c$, and that for every state E, $\mathcal{O}^e(E_2, E) \not\models b$ (i.e. the execution of b in state E_2 fails).

Then, formulas $p.ins \otimes \mathbf{ext}(a, a^{-1})$ and $q.ins \otimes \mathbf{ext}(c, c^{-1})$ are classically satisfied in paths $\langle(\{\}, E_1),^{p.ins}(\{p\}, E_1),^{\mathbf{ext}(a,a^{-1})}(\{p\}, E_2)\rangle$ and $\langle(\{\}, E_1),^{q.ins}(\{q\}, E_1),^{\mathbf{ext}(c,c^{-1})}(\{q\}, E_4)\rangle$ respectively. Moreover, it is easy to check that $\mathbf{ext}(b, b^{-1})$ cannot succeed in any path starting in state E_2 (given the external oracle definition). The idea of partial satisfaction is to identify the path $\langle(\{\}, E_1),^{p.ins}(\{p\}, E_1),^{\mathbf{ext}(a,a^{-1})}(\{p\}, E_2)\rangle$ as one that satisfies the formula $p.ins \otimes \mathbf{ext}(a, a^{-1}) \otimes \mathbf{ext}(b, b^{-1})$ up to some point, though it eventually fails.

So, in partial satisfaction, we allow primitives to fail at some point, and for $\phi \otimes \psi$ it suffices to satisfy part of the conjunction. Specifically:

Definition 6 (*Partial Satisfaction*)
Let M be an interpretation, π a path and ϕ a formula. If $M(\pi) = \top$ then $M, \pi \models_p \phi$; otherwise:

1. **Base Case:** $M, \pi \models_p \phi$ iff ϕ is an atom and one of the following holds:
   (a) $M, \pi \models_c \phi$
   (b) $M, \pi \not\models_c \phi$, $\phi \in \mathcal{L}_i$, $\pi = \langle(D, E)\rangle$ and $\neg\exists D_i$ s.t. $M, \langle(D, E),^{\phi}(D_i, E)\rangle \models_c \phi$
   (c) $M, \pi \not\models_c \phi$, $\phi \in \mathcal{L}_a^*$, $\pi = \langle(D, E)\rangle$ and $\neg\exists E_i$ s.t. $M, \langle(D, E),^{\phi}(D, E_i)\rangle \models_c \phi$
2. **Negation:** $M, \pi \models_p \neg\phi$ iff it is not the case that $M, \pi \models_p \phi$
3. **"Classical" Disjunction:** $M, \pi \models_p \phi \vee \psi$ iff $M, \pi \models_p \phi$ or $M, \pi \models_p \psi$
4. **"Classical" Conjunction:** $M, \pi \models_p \phi \wedge \psi$ iff $M, \pi \models_p \phi$ and $M, \pi \models_p \psi$
5. **Serial Conjunction:** $M, \pi \models_p \phi \otimes \psi$ iff one of the following holds:
   (a) $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$
   (b) $\exists$ split $\pi_1 \circ \pi_2$ of path π s.t. $M, \pi_1 \models_c \phi$ and $M, \pi_2 \models_p \psi$

For the usage below of this definition, it is crucial to know the exact failure point of the transaction, so that external actions priorly performed can be compensated. For that, the path in which a formula is partially but not classically satisfied must always end exactly in the state prior to the failure (cf. Proposition 1 below), and it is why failures are constraint to 1-paths in points 1b and 1c. These 1-paths represent the state where the transaction failed. Moreover, as desired, in any path π where a ϕ can be partially but not classically executed, then $M, \pi \models_p \phi \otimes \psi$ for every formula ψ. Also, for formulas without negation the partial satisfaction is a relaxed version of the classical satisfaction, and the two satisfaction relations coincide whenever they are evaluating atoms that are not specified by the oracles.

Proposition 1
Let M be an interpretation, π a path, $\pi_{\mathtt{end}}$ the 1-path containing the last state of π, ϕ and ψ any $\mathcal{ETR}$ formulas, ϕ' a formula without negation, ϕ_P an atom from $\mathcal{L}_P$ and a an atom such that $a \in \mathcal{L}_i$ or $a \in \mathcal{L}_a^*$.

1. If $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$ then $\exists a$ s.t. $M, \pi_{\mathtt{end}} \models_p a$ and $M, \pi_{\mathtt{end}} \not\models_c a$
2. If $M, \pi \models_p \phi$ and $M, \pi \not\models_c \phi$ then $M, \pi \models_p \phi \otimes \psi$
3. If $M, \pi \models_c \phi'$ then $M, \pi \models_p \phi'$
4. $M, \pi \models_c \phi_P$ iff $M, \pi \models_p \phi_P$

Consider again example1. In it, somehow (to be better defined below) a transaction (t) is the disjunction of the two bodies of the rules, and we want to consider an additional way of satisfying that disjunction. Namely, the disjunction is satisfied if the first body is "tried", compensated, and then the second disjunct is successfully executed. With the definitions above, we made precise what is meant by the "tried", viz. it is partially but not classically satisfied. The next definitions specify what is left, i.e. how to compensate a formula that is partially but not classically satisfied.

For this, some operations on paths are required. To start, one has to collect all actions that have been executed in a path and need to be compensated; and to rollback the internal state:

Definition 7 (*Rollback Path, and Sequence of External Actions*)
Let π be a k-path of the form $\langle (D_1, E_1), ^{A_1} (D_2, E_2), ^{A_2} \ldots, ^{A_{k-1}} (D_k, E_k)\rangle$. The *rollback* path of π is the path obtained from π by: (1) Replacing all D_is by the initial state D_1; (2) Keeping just the transitions where $A_i \in \mathcal{L}_a^*$.
The sequence of external actions of π, denoted $\texttt{Seq}(\pi)$, is the sequence of actions of the form $\mathbf{ext}(a, a^{-1})$ that appear in the transitions of the rollback path of π.

$\texttt{Seq}(\pi)$ only collects the external actions that have the form $\mathbf{ext}(a, a^{-1})$. Since this operation aims to compensate the executed actions, then actions without compensations are skipped. To define compensations that always fail, one should use the the primitive `failop` in a^{-1}.

Building on this, we define a recovery path as the path obtained from executing each compensation operation defined in $\texttt{Seq}(\pi)$ in the inverse order.

Definition 8 (*Inversion, and Recovery Path*)
Let $S = \langle \mathbf{ext}(A_1, A_1^{-1}), \ldots \mathbf{ext}(A_n, A_n^{-1})\rangle$ be a sequence of actions from $\mathcal{L}_a^*$. Then, the inversion of S is the transaction formula $\texttt{Inv}(S) = A_n^{-1} \otimes \ldots \otimes A_1^{-1}$.
π_r is a *recovery path* of $\texttt{Seq}(\pi)$ w.r.t. M iff $M, \pi_r \models_c \texttt{Inv}(\texttt{Seq}(\pi))$.

Example 3
Recall example 2. The rollback path of $\pi = \langle (\{\}, E_1), ^{p.ins} (\{p\}, E_1), ^{\mathbf{ext}(a,a^{-1})} (\{p\}, E_2)\rangle$ is $\langle (\{\}, E_1), ^{\mathbf{ext}(a,a^{-1})} (\{\}, E_2)\rangle$ and $\texttt{Seq}(\pi) = \langle \mathbf{ext}(a, a^{-1})\rangle$. Assuming that $\mathcal{O}^e(E_2, E_3) \models a^{-1}$ then $\langle (\{\}, E_2), ^{a^{-1}} (\{\}, E_3)\rangle$ is a recovery path of $\texttt{Seq}(\pi)$ w.r.t. any interpretation M.

Equipped with these auxiliary definitions, we can finally make precise what we mean by compensating a formula that is partially but not classically satisfied. I.e., given an interpretation M, what are the paths π in which all actions executed in a formula ϕ are compensated, even though ϕ is not satisfied in π (written as $M, \pi \rightsquigarrow \phi$).

Definition 9 (*Compensating Path for a Transaction*)
Let M be an interpretation, π a path and ϕ a formula. $M, \pi \rightsquigarrow \phi$ iff *all* the following hold:

1. $\exists \pi_1$ such that $M, \pi_1 \models_p \phi$ and $M, \pi_1 \not\models_c \phi$
2. $\exists \pi_0$ such that π_0 is the rollback path of π_1
3. $\texttt{Seq}(\pi_1) \neq \emptyset$ and $\exists \pi_r$ such that π_r is a recovery path of $\texttt{Seq}(\pi_1)$ w.r.t. M
4. π_0 and π_r are a split of π, i.e. $\pi = \pi_0 \circ \pi_r$

In the scenario of examples 2 and 3, $M, \langle (\{\}, E_1), ^{\mathbf{ext}(a,a^{-1})} (\{\}, E_2), ^{a^{-1}} (\{\}, E_3)\rangle \rightsquigarrow p.ins \otimes \mathbf{ext}(a, a^{-1}) \otimes \mathbf{ext}(b, b^{-1})$ for any interpretation M (that satisfies the oracles). Note that this path does not satisfy the formula (since $\mathbf{ext}(b, b^{-1})$ fails) but, at least, it leaves the internal and external KBs in a state somehow equivalent to the initial state: the operations done in the internal KB are rolled back, and the externally executed actions are compensated.

We are now able to formalize what (complex) formulas are true on what paths.

Definition 10 (*General Satisfaction*)
Let M be an interpretation, π a path and ϕ a formula. If $M(\pi) = \top$ then $M, \pi \models \phi$; otherwise:

1. **Base Case:** $M, \pi \models \phi$ if $\phi \in M(\pi)$ for any atom ϕ
2. **Negation:** $M, \pi \models \neg\phi$ if it is not the case that $M, \pi \models \phi$
3. **"Classical" Disjunction:** $M, \pi \models \phi \vee \psi$ if $M, \pi \models \phi$ or $M, \pi \models \psi$.
4. **"Classical" Conjunction:** $M, \pi \models \phi \wedge \psi$ if $M, \pi \models \phi$ and $M, \pi \models \psi$.
5. **Serial Conjunction:** $M, \pi \models \phi \otimes \psi$ if $M, \pi_1 \models \phi$ and $M, \pi_2 \models \psi$ for some split $\pi_1 \circ \pi_2$ of π.
6. **Compensating Case:** $M, \pi \models \phi$ if $M, \pi_1 \rightsquigarrow \phi$ and $M, \pi_2 \models \phi$ for some split $\pi_1 \circ \pi_2$ of π
7. For no other M, π and ϕ, $M, \pi \models \phi$.

Intuitively, with this notion of satisfaction, a formula ϕ succeeds if it succeeds classically, or if although a primitive action failed to be executed, the system can recover from the failure and ϕ can still succeed in an alternative path (point 6). Obviously, recovery only makes sense when external actions were performed before the failure. Otherwise we can just rollback for the initial state and try to satisfy the formula in an alternative branching.

Recall examples 2 and 3, and assume $\mathcal{O}^e(E_3, E_4) \models \mathbf{ext}(c, c^{-1})$. Then the complex formula $(p.ins \otimes \mathbf{ext}(a, a^{-1}) \otimes \mathbf{ext}(b, b^{-1})) \vee (q.ins \otimes \mathbf{ext}(c, c^{-1}))$ is satisfied both in the path $\langle(\{\}, E_1),^{q.ins}(\{q\}, E_1),^{\mathbf{ext}(c,c^{-1})}(\{q\}, E_4)\rangle$ – without using compensations (point6) – and in $\langle(\{\}, E_1),^{\mathbf{ext}(a,a^{-1})}(\{\}, E_2),^{a^{-1}}(\{\}, E_3),^{q.ins}(\{q\}, E_3),^{\mathbf{ext}(c,c^{-1})}(\{q\}, E_4)\rangle$ – using point6.

Besides the compensating case, the definition of general satisfaction exactly coincides with classical satisfaction (Definition 5). Next, we formalize this correspondence.

Theorem 1
Let M be an interpretation, ϕ an $\mathcal{ETR}$ formula and π, π' paths such that π' is a path where no external actions appear in the transitions. Then:

$$\text{If } M, \pi \models_c \phi \text{ then } M, \pi \models \phi \tag{1}$$

$$M, \pi' \models_c \phi \text{ iff } M, \pi' \models \phi \tag{2}$$

When compared with the classical satisfaction, the general satisfaction relation weakens the conditions for the satisfaction of formulas in a path and thus, whenever ϕ is classically satisfied in π, then it is also generally satisfied. This is made precise in statement (1) of Theorem 1. Moreover, (2) shows the correspondence whenever ϕ does not depend on external actions.

The satisfaction relation induces a definition of model and of entailment in the usual way.

Definition 11 (*Models and Logical Entailment*)
Let ϕ and ψ be two $\mathcal{ETR}$ formulas and M be an interpretation. M is a model of ϕ (denoted $M \models \phi$) iff $M, \pi \models \phi$ for every path π. Let $Bodies(A)$ be the disjunction of the bodies of all rules with head A. An interpretation M is a model of a program P if, for every atom $A \in \mathcal{L}_P$ and every path π, whenever $M, \pi \models Bodies(A)$ then $M, \pi \models A$.
We say that ϕ logically entails ψ ($\phi \models \psi$) if every model of ϕ is also a model of ψ.

4 Executional Entailment

The previously defined logical entailment specifies a general entailment that takes into account all the possible execution paths of a transaction formula. Hence, logical entailment is useful to define general equivalence and implication of formulas, as one can express properties like "transaction ϕ is equivalent to transaction ψ" or "whenever transaction ϕ is executed, ψ is also executed".

This kind of general entailment is very powerful but sometimes one needs a simpler kind of reasoning that is concerned only with a particular execution of a formula. As such, similarly to $\mathcal{TR}$, in addition to logical entailment $\mathcal{ETR}$ supports another entailment called *executional entailment*. Whilst logical entailment allows one to *reason* about $\mathcal{ETR}$ theories, executional entailment provides a logical account to *execute* $\mathcal{ETR}$ programs.

Definition 12 (*Executional Entailment*)
Let P be a program, ϕ be a formula and $S_1,^{A_1} \ldots,^{A_{n-1}} S_n$ be a path:

$$P, (S_1,^{A_1} \ldots,^{A_{n-1}} S_n) \models \phi \tag{3}$$

is true if $M, \langle S_1,^{A_1} \ldots,^{A_{n-1}} S_n\rangle \models \phi$ for every model M of P. We write $P, S_1{-} \models \phi$ when there exists a path $S_1,^{A_1} \ldots,^{A_{n-1}} S_n$ that makes (3) true.

The expression $P, (S_1,^{A_1} \ldots,^{A_{n-1}} S_n) \models \phi$ denotes that given a transaction program P, the path $(S_1,^{A_1} \ldots,^{A_{n-1}} S_n)$ is a valid execution for transaction ϕ. In the case of our running example we can say that: $P, \langle(\{\}, E_1),^{q.ins} (\{q\}, E_1),^{\mathbf{ext}(c,c^{-1})} (\{q\}, E_4)\rangle \models t$, but also:
$P, \langle(\{\}, E_1),^{\mathbf{ext}(a,a^{-1})} (\{\}, E_2),^{a^{-1}} (\{\}, E_3),^{q.ins} (\{q\}, E_3),^{\mathbf{ext}(c,c^{-1})} (\{q\}, E_4)\rangle \models t$.

$P, S_1{-} \models \phi$ accounts for situations where all one wants to know is whether ϕ can succeed from S_1 under P, e.g. $P, (\{\}, E_1){-} \models t$ (meaning that t succeeds if executed in that initial state).

We can now precise the relation between $\mathcal{TR}$ and $\mathcal{ETR}$. Namely we can conclude that if ϕ and P are valid in both logics (e.g. do not contain external actions), then the two logics coincide, i.e. they satisfy the same formulas in the same paths. Obviously, since paths in $\mathcal{ETR}$ have an additional external component than in $\mathcal{TR}$, the paths only coincide in their shared internal path.

Theorem 2 (*Relation to $\mathcal{TR}$*)
Let P be a transaction program and ϕ a transaction formula such that P and ϕ are valid both in $\mathcal{TR}$'s and in $\mathcal{ETR}$'s syntax. Then:

$$P, \pi' \models_{TR} \phi \text{ iff } P, \pi \models_{ETR} \phi$$

where π' is obtained from π by removing the external component and the annotated transitions in every transition of states.

5 A Procedure for serial-$\mathcal{ETR}$

The previously defined executional entailment determines what is the meaning of executing a transaction defined in a program, starting from an initial state. Our next step is to define a procedure for executing transactions in that way. In this section we extend the proof theory for ground[2] the serial-Horn $\mathcal{TR}$ fragment as described in (Bonner and Kifer 1995). The advantage of this fragment is that it can be formulated as a least-fixpoint in a logic programming style.

A serial-Horn program P is a finite set of *serial goals*. A serial goal is a transaction formula of the form $a_1 \otimes a_2 \otimes \ldots \otimes a_n$, where each a_i is an atom and $n \geq 0$. When $n = 0$, we write $()$, which denotes the empty goal. A *serial-Horn rule* has the form $b \leftarrow a_1 \otimes \ldots \otimes a_n$, where the body $a_1 \otimes \ldots \otimes a_n$ is a serial goal and the head b is an atom.

Here, we present a procedure to verify that $P, S_0{-} \models \phi$, i.e, that a transaction ϕ can succeed

[2] The restriction to ground formulas is not essential and can be easily lifted. We only require it in order to simplify the presentation.

starting from the state $S_0 = (D_0, E_0)$ and, in case of success, to obtain a path starting in S_0 that satisfies ϕ. In a nutshell, given a program and an initial state, the procedure non-deterministically applies a series of rules, until eventually reaches the empty goal and succeeds, or no more rules are applicable and fails. In case of success, it also return a path in which it succeeds. To cater for this last requirement, resolvents are of the form $\pi, S_i \Vdash_P \phi$, meaning that, we want to execute transaction ϕ in P from the initial state S_i; where π is used to record the previous path.

A proof, or successful derivation, for $P, S_0{}^- \models \phi$ starts with $\langle S_0 \rangle, S_0 \Vdash_P \phi$ and applies the rules defined below, until eventually it reaches a resolvent $\pi, S_f \Vdash_P ()$. If such a proof is found, then we further conclude that $P, \pi \models \phi$ (where π starts with S_0 and ends with S_f). i.e. not only we have prove that ϕ can succeed starting from S_0, but we also found a path where ϕ succeeds.

Most of the derivation rules are pretty easy to follow: when proving an atom which is at the head of some rule, replace the atom in the resolvent by the body of the rule; when proving an atom that is true in an oracle, just remove the atom from the resolvent, and update the path and state appropriately. This forms the basis of the so called $SLD_{\mathcal{ETR}}$ classical derivation. $SLD_{\mathcal{ETR}}$ derivations can use an additional rule, that deals with compensations. For that, we need, besides the usual notion of successful derivation, also to deal with derivations that do not succeed, but end in the execution of an action that fails in the oracle – we call these *action-failed* derivation. In the latter case, we need to roll back, compensate the executed actions, and proceed.

Definition 13 (*$SLD_{\mathcal{ETR}}$ Derivation and Classical Derivation*)
An $SLD_{\mathcal{ETR}}$-derivation (resp. classical derivation) for a serial goal ϕ in a program P and state S_0 is a sequence of resolvents starting with $\langle S_0 \rangle, S_0 \Vdash_P \phi$, and obtained by non-deterministically applying the following r_1–r_5 (resp. r_1–r_4) rules. Let $\pi, (D_1, E_1) \Vdash_P L_1 \otimes \ldots \otimes L_n$ be a resolvent. Then the next resolvent in the derivation can be:

$\mathbf{r_1}$. $\pi, (D_1, E_1) \Vdash_P B_1 \otimes \ldots \otimes B_j \otimes L_2 \otimes \ldots \otimes L_n$ if $L_1 \leftarrow B_1 \otimes \ldots \otimes B_j \in P$
$\mathbf{r_2}$. $\pi, (D_1, E_1) \Vdash_P L_2 \otimes \ldots \otimes L_n$ if $\mathcal{O}^d(D_1) \models L_1$
$\mathbf{r_3}$. $\pi \circ \langle (D_1, E_1), {}^{L_1}(D_2, E_1) \rangle, (D_2, E_1) \Vdash_P L_2 \otimes \ldots \otimes L_n$ if $\mathcal{O}^t(D_1, D_2) \models L_1$
$\mathbf{r_4}$. $\pi \circ \langle (D_1, E_1), {}^{L_1}(D_1, E_2) \rangle, (D_1, E_2) \Vdash_P L_2 \otimes \ldots \otimes L_n$ if $\mathcal{O}^e(E_1, E_2) \models L_1$
$\mathbf{r_5}$. $\pi \circ \langle S_1, {}^{A_1} \ldots, {}^{A_{p-1}} S_p, {}^{A_k^{-1}} \ldots, {}^{A_1^{-1}} S_q \rangle, S_q \Vdash_P L_1 \otimes \ldots \otimes L_n$ if all conditions hold:

- There is an action-failed classical derivation starting in $\langle S_1 \rangle, S_1 \Vdash_P L_1 \otimes \ldots \otimes L_n$ (where $S_1 = (D_1, E_1)$) ending in $\langle S_1, {}^{A_1} \ldots, {}^{A_{j-1}} S_j \rangle, S_j \Vdash_P \phi$, for some transaction ϕ
- $S_1, {}^{A_1} \ldots, {}^{A_{p-1}} S_p$ is the rollback path of $S_1, {}^{A_1} \ldots, {}^{A_{j-1}} S_j$ (cf. Definition 7)
- $\texttt{Inv}(\texttt{Seq}(\langle S_1, {}^{A_1} \ldots, {}^{A_{p-1}} S_p \rangle)) = A_k^{-1} \otimes \ldots \otimes A_1^{-1}$ (cf. Definition 8)
- There is successful classical derivation for $\langle S_p \rangle, S_p \Vdash_P A_k^{-1} \otimes \ldots \otimes A_1^{-1}$ ending in $\langle S_p, {}^{A_k^{-1}} \ldots, {}^{A_1^{-1}} S_q \rangle, S_q \Vdash_P ()$

Definition 14 (*Successful and Action-failed Derivations*)
Let P be a program, ϕ a serial goal and S_0 an initial state. An $SLD_{\mathcal{ETR}}$-derivation (resp. classical derivation) for ϕ in P starting in S_0 is *successful* if it ends in a resolvent of the form $\pi, S_f \Vdash_P ()$. In this case we write $P, \pi \vdash \phi$ (resp. $P, \pi \vdash_c \phi$).
The derivation is *action-failed* if it ends in a resolvent of the form $\pi, S_f \Vdash_P L_1 \otimes \ldots \otimes L_n$ s.t.:

(i). $L_1 \in \mathcal{L}_i$, $\mathcal{O}^d(D_f) \not\models L_1$ and $\neg\exists D_i$ s.t. $\mathcal{O}^t(D_f, D_i) \models L_1$, or
(ii). $L_1 \in \mathcal{L}_a^*$ and $\neg\exists E_i$ s.t. $\mathcal{O}^e(E_f, E_i) \models L_1$

Taken together, definitions 13 and 14 determine a sound and complete procedure to find the paths that satisfy a transaction ϕ given a program P and an initial state S_0. This procedure

resembles an SLD-style procedure and can be seen as an extension of the inference system for serial-$\mathcal{TR}$ as presented in (Bonner and Kifer 1993). The main differences when compared to the inference system defined in (Bonner and Kifer 1993) are the evaluation of external actions w.r.t to an external oracle $\mathcal{O}^e$ and the non-deterministic possibility of executing compensations.

Theorem 3 (*Soundness and Completeness of* $\vdash$)
Let P be a serial-Horn program, ϕ a serial-Horn goal, and π be a path starting in state S_0 and ending in S_f. Then, $P, \pi \models \phi$ iff $P, \pi \vdash \phi$.

6 Discussion and Related Work

In this paper we propose an extension of $\mathcal{TR}$ to accommodate interaction with an external world that requires the relaxation of the standard ACID transaction model. This continues the work started in (Gomes and Alferes 2011), providing cleaner and intuitive definitions that allow us to prove a seamlessly relation with $\mathcal{TR}$ and to construct an SLD-style proof theory for the Horn subset of the logic. By providing a model-theoretic semantics along with the sound and complete top-down procedure, $\mathcal{ETR}$ becomes useful for executing *and* reasoning about hybrid systems that have both an internal and an external component and require properties on the outcome of the updates. This allows $\mathcal{ETR}$ to study properties of updates (as e.g. saying that in every possibility of execution transaction ϕ implies the execution of transaction ψ, or what constraints are always true in every execution of ϕ) while at the same time providing the possibility to materialize these updates according to the model-theory defined. Examples of these hybrid systems range from an intelligent agent that has an internal knowledge base where he executes reasoning, but is integrated in an evolving external world where he can execute actions; to web-based systems with an internal knowledge base that follows the ACID model, but also needs to interact with other systems, e.g. to request a web-service.

$\mathcal{ETR}$ as $\mathcal{TR}$ can be compared to many logics that reason about state change or about the related phenomena of time and action. These include Action Languages (Gelfond and Lifschitz 1998), the Situation Calculus (McCarthy 1963), the Event Calculus (Kowalski and Sergot 1986), Process Logic (Harel et al. 1982), etc. An important formalism in this area that relates to $\mathcal{ETR}$ is the work of (Eiter et al. 2007). Here it is presented a solution based on Action Languages to represent actions and reason about their correct reverse actions (or compensations).

The aforementioned logics aim to provide complete descriptions of the domain in order to reason about what actions are possible and what are their (possibly indirect) effects in the domain. As such, these solutions need to be as expressible as possible and to provide solutions to the inherent frame problem. In this sense, $\mathcal{ETR}$ (as $\mathcal{TR}$) was design with a completely different intent. $\mathcal{ETR}$ semantics concerns with the combination of several atomic actions into transaction programs by abstracting the meaning of the atomic actions and their effects in the knowledge base. Thus, rather than an alternative to $\mathcal{ETR}$, these logics can be used to instantiate its oracles and reason about the dynamics of the internal and external knowledge base as e.g. shown by the Situation Calculus Oracle formalization provided in Section 2.1. Similarly, one could use the solution of (Eiter et al. 2007) to define the correct compensations for the external actions predicates $\mathbf{ext}(a, a^{-1})$. In this sense, whenever the external action $\mathbf{ext}(a, a^{-1})$ is posed as a call to the external oracle defined by (Eiter et al. 2007), the oracle would instantiate a^{-1} with the action that reverses the effects of a. For a detail comparison between $\mathcal{TR}$ and some of these logics the reader is referred to (Bonner and Kifer 1995).

One can also compare $\mathcal{ETR}$ to formalisms based on process algebras that involve the notion of long-running transactions like (Butler et al. 2004; Vaz and Ferreira 2012; Bocchi et al. 2003). These are generally based on algebraic systems for modeling concurrent communicating processes, as Milner's CCS (Milner 1983) or Hoare's CSP (Hoare 1985), among others. One clear difference between $\mathcal{ETR}$ and these solutions is that $\mathcal{ETR}$ does not support concurrency and synchronization. However, providing these features to $\mathcal{ETR}$ is an obvious future work milestone and is in line with what has been done in Concurrent Transaction Logic (Bonner and Kifer 1996). Notwithstanding the major difference between $\mathcal{ETR}$ and proposals based on process algebras is mainly conceptual. These systems focus mostly on the correct execution and synchronization of processes and thus possess a powerful operational semantics that worries about properties like correctness and termination of execution. They enclose powerful operators that in some cases even allow the system to construct the correct compensation for each action "on-the-fly" as in (Vaz and Ferreira 2012). However, these solutions based on process algebras are mostly operational and are not suitable to be used as knowledge representation formalisms. Their lack of model theory and knowledge of state makes it impossible to model what is true at each step of the execution or to specify constraints on their execution based on this knowledge. $\mathcal{ETR}$ stands in between the two worlds. It provides a clean model-theoretic semantics that allows us to talk about properties of transactions like equivalence and implication that hold independently of what execution path is chosen. But also, by providing a proof-theory that is sound and complete with the semantics, $\mathcal{ETR}$ is able to talk about a particular execution of a transaction and what are the possible evolution paths for a given formula.

Another interesting work is the rule-based language ULTRA (Fent et al. 1999) that is based on minimal model semantics to define concurrent transactions in arbitrary domains. While in its semantics, ULTRA and $\mathcal{TR}$ are very similar (and in their sequential version, the logics are proven to have the same modeling power), ULTRA's implementation identifies the need for defining compensating subtransactions for every transaction committed. However, this notion of compensation is not defined in ULTRA's model theory which does not account for any means to soften the ACID model of transaction. Thus there is no formal correspondence between the procedure of ULTRA and its model theory as in $\mathcal{ETR}$.

Finally, in (Rezk and Kifer 2012) the authors propose an extension of Transaction Logic with Partially Defined Actions ($\mathcal{TR}^{PAD}$) that allows one to define axioms stating the indirect effects of actions and to directly encode partial descriptions of states. $\mathcal{TR}^{PAD}$ can be use to model external environments with incomplete information, reason about the actions and their effects in the domain. Moreover, since it provides a proof-theory sound and complete with the model theory defined, $\mathcal{TR}^{PAD}$ also provides a way to execute these actions. However, $\mathcal{TR}^{PAD}$ goes against the original philosophy of $\mathcal{TR}$. Since the information about the domain and actions is defined directly in the logic, the notion of oracles is abandoned and states can only be defined by ground propositional formulas. This swift of paradigm precludes the possibility of changing the semantics of states or combine several semantics to reason with more than one demain, making $\mathcal{TR}^{PAD}$ less flexible. Moreover, it is unclear how the transaction properties of $\mathcal{TR}$ can benefit the $\mathcal{TR}^{PAD}$ or how can they be ensured in an external domain like this. It is also impossible to relax some properties of transactions as in $\mathcal{ETR}$.

References

BOCCHI, L., LANEVE, C., AND ZAVATTARO, G. 2003. A calculus for long-running transactions. In

FMOODS, E. Najm, U. Nestmann, and P. Stevens, Eds. Lecture Notes in Computer Science, vol. 2884. Springer, 124–138.

BONNER, A. J. AND KIFER, M. 1993. Transaction logic programming. In *Proceedings of the tenth international conference on logic programming on Logic programming*. ICLP'93. MIT Press, Cambridge, MA, USA, 257–279.

BONNER, A. J. AND KIFER, M. 1994. Applications of transaction logic to knowledge representation. In *ICTL*, D. M. Gabbay and H. J. Ohlbach, Eds. Lecture Notes in Computer Science, vol. 827. Springer, 67–81.

BONNER, A. J. AND KIFER, M. 1995. Transaction logic programming (or a logic of declarative and procedural knowledge). Tech. Rep. CSRI-323, Univ. of Toronto.

BONNER, A. J. AND KIFER, M. 1996. Concurrency and communication in transaction logic. In *JICSLP*. MIT Press, 142–156.

BONNER, A. J. AND KIFER, M. 1998a. A logic for programming database transactions. In *Logics for Databases and Information Systems*, J. Chomicki and G. Saake, Eds. Kluwer, 117–166.

BONNER, A. J. AND KIFER, M. 1998b. Results on reasoning about updates in transaction logic. In *Transactions and Change in Logic Databases*, B. Freitag, H. Decker, M. Kifer, and A. Voronkov, Eds. Lecture Notes in Computer Science, vol. 1472. Springer, 166–196.

BONNER, A. J., KIFER, M., AND CONSENS, M. P. 1993. Database programming in transaction logic. In *DBPL*, C. Beeri, A. Ohori, and D. Shasha, Eds. Workshops in Computing. Springer, 309–337.

BUTLER, M. J., HOARE, C. A. R., AND FERREIRA, C. 2004. A trace semantics for long-running transactions. In *25 Years Communicating Sequential Processes*, A. E. Abdallah, C. B. Jones, and J. W. Sanders, Eds. Lecture Notes in Computer Science, vol. 3525. Springer, 133–150.

EITER, T., ERDEM, E., AND FABER, W. 2007. On reversing actions: Algorithms and complexity. In *IJCAI*, M. M. Veloso, Ed. 336–341.

FENT, A., WICHERT, C.-A., AND FREITAG, B. 1999. Logical update queries as open nested transactions. In *FMLDO - Selected Papers*, G. Saake, K. Schwarz, and C. Türker, Eds. Lecture Notes in Computer Science, vol. 1773. Springer, 45–66.

FODOR, P. AND KIFER, M. 2011. Transaction logic with defaults and argumentation theories. In *ICLP (Technical Communications)*, J. P. Gallagher and M. Gelfond, Eds. LIPIcs, vol. 11. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 162–174.

GARCIA-MOLINA, H. AND SALEM, K. 1987. Sagas. *SIGMOD Rec. 16*, 249–259.

GELFOND, M. AND LIFSCHITZ, V. 1998. Action languages. *Electron. Trans. Artif. Intell. 2*, 193–210.

GOMES, A. S. AND ALFERES, J. J. 2011. Transaction logic with external actions. In *LPNMR*. 272–277.

HAREL, D., KOZEN, D., AND PARIKH, R. 1982. Process logic: Expressiveness, decidability, completeness. *J. Comput. Syst. Sci. 25,* 2, 144–170.

HOARE, C. A. R. 1985. *Communicating Sequential Processes*. Prentice-Hall.

KOWALSKI, R. A. AND SERGOT, M. J. 1986. A logic-based calculus of events. *New Generation Comp. 4,* 1, 67–95.

MCCARTHY, J. 1963. Situations, actions, and causal laws. Tech. rep., Stanford University. Reprinted in MIT Press, Cambridge, Mass., 1968 pages 410-417.

MILNER, R. 1983. Calculi for synchrony and asynchrony. *Theor. Comput. Sci. 25*, 267–310.

REZK, M. AND KIFER, M. 2012. Transaction logic with partially defined actions. *J. Data Semantics 1,* 2, 99–131.

ROMAN, D. AND KIFER, M. 2007. Reasoning about the behavior of semantic web services with concurrent transaction logic. In *VLDB*, C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C.-C. Kanne, W. Klas, and E. J. Neuhold, Eds. ACM, 627–638.

VAZ, C. AND FERREIRA, C. 2012. On the analysis of compensation correctness. *J. Log. Algebr. Program. 81,* 5, 585–605.