**Journal of Intelligent Information Systems manuscript No.**
(will be inserted by the editor)

# Synthesizing Advanced Transaction Models Using the Situation Calculus*

**Iluju Kiringa**[1], **Alfredo Gabaldon**[2,3]

[1] School of Information Technology and Engineering
Faculty of Engineering, University of Ottawa, Ottawa, Canada
`kiringa@site.uottawa.ca`
[2] National ICT Australia
[3] School of Computer Science and Engineering
University of New South Wales, Sydney, Australia
alfredo@cse.unsw.edu.au

Received: / Revised version:

**Abstract** The situation calculus is a versatile logic for reasoning about actions and formalizing dynamic domains. Using the non-Markovian action theories formulated in the situation calculus, one can specify and reason about the effects of database actions under the constraints of the classical, flat database transactions, which constitute the state of the art in database systems. Classical transactions are characterized by the so-called ACID properties. With non-Markovian action theories, one can also specify, reason about, and even synthesize various extensions of the flat transaction model, generally called *advanced transaction models* (ATMs).

In this paper, we show how to use non-Markovian theories of the situation calculus to specify and reason about the properties of ATMs. In these theories, one may refer to past states other than the previous one. ATMs are expressed as such non-Markovian theories using the situation calculus. We illustrate our method by specifying (and sometimes reasoning about the properties of) several classical models and known ATMs.

## 1 Introduction

### *1.1 Motivation*

Transaction systems that constitute the state of the art in database systems have a flat structure defined in terms of the so-called ACID (Atomicity-Consistency-Isolation-Durability) properties. From the system point of view, a database transaction is a sequence of operations on the database state, which exhibit the ACID

* Early short versions of this paper appeared in [18,19]

properties and are bracketed by $Begin$ and $Commit$ or $Begin$ and $Rollback$ [14]. A transaction makes the results of its operations durable when nothing goes wrong before its normal end by executing a $Commit$ operation, upon which the database cannot be rolled back. Should anything go wrong before the commitment, the transaction rolls the database back to the state before beginning.

A transaction is *atomic* when it either brings the database from the initial state to a final state, or it appears as if it had never changed the database content. *Consistency* of a transaction means that, given an initial database state that satisfies all the integrity constraints of the database, the final state also satisfies them. *Isolation* is the requirement that a transaction running concurrently with other transactions must exactly behave in the same way as if it were running in a single-transaction mode; it is traditionally ensured through the *serializability* property. A *concurrent* execution of a set of transactions is serializable iff it has the same outcome as some sequential execution of the involved transactions. Finally, *durability* means that from the commitment point onwards, the results of a transaction are permanent.

Over the last two decades, various other transaction models have been proposed to extend the classical, flat transaction model by relaxing some of the ACID properties [10].Many of these extensions introduced various models of nested transactions [28], which are sets of transactions (called subtransactions) forming a tree structure. Such extensions, generally called *advanced transaction models* (ATMs), are proposed for dealing with new applications involving long-lived, endless, and cooperative activities. In the classic transaction model and its extensions into transactions with savepoints and chained transactions [14], the components of a transaction are sequenced. On the contrary, most ATMSs permit a hierarchical organization structure of their components. Moreover, ATMs generally relax the ACID properties of the flat transactions by dropping or modifying some of these properties. The ATMs aim at improving the functionality and the performance of the new applications.

The ATMs, however, have been proposed in an *ad hoc* fashion, thus lacking in generality in a way that it is not obvious to compare the different ATMs, to say exactly how they extend the traditional flat model, and to formulate their properties in a way that one sees clearly which new functionality has been added, or which one has been subtracted. To address these questions, there is a need for a general and common framework within which to specify ATMs, specify their properties, and reason about these properties. Thus far, ACTA [9], $\mathcal{TR}$ [6,7], and Statelog [21] seem to be the only frameworks in the literature that address the specification of ATMs at a general level. In ACTA, a logic-like language is used, whereas in $\mathcal{TR}$ and Statelog, pure logic is used.

## *1.2 Contributions*

We extend the specification of database updates given in [32] to a logical framework to specify and reason about the properties of ATMs. This framework constitutes a general theory of database transactions with (relaxed) ACID properties.

To do so, we appeal to the non-Markovian action theories formulated in the situation calculus, which is a classical second order logic for specifying dynamic domains [34, 12] where one may refer to past states other than the previous one. Non-Markovian theories are needed, for example, to concisely (i.e., with few axioms) capture the semantics of transaction actions which very often refer to the history of a database. Next, we introduce the building blocks of our specification framework for representing relational database transactions in the situation calculus. After that, we model relational flat database transactions as second order theories called *basic relational theories*, and define a legal log as one in which all database actions that are possible have indeed been executed. Here, we focus on using basic relational theories to model flat transactions with ACID properties, which turn out to be properties of logs that are legal. We then generalize these relational theories to deal with variations of classical flat transactions, and various ATMs, among which closed and open nested transactions are good examples. We show how to build relational theories corresponding to these various ATMs, to formulate properties of these ATMs in the situation calculus (using the case of closed nested transactions), and to prove these properties as logical consequences of the relational theories. The main and crucial advantage of our framework over ACTA, $\mathcal{TR}$, and Statelog lies in the use of a logic specifically designed for dealing with actions as first class objects of the language. Our framework is intended as a formal tool for the precise analysis of existing ATMs and the systematic development of new ATMs. Specifications thus obtained can be used as background theories for simulating the corresponding ATMs as shown in [18].

### *1.3 Outline*

The remainder of this paper is organized as follows. Section 2 introduces the situation calculus for representing relational database transactional domains as non-Markovian action theories. This section also spells out the details of our specification framework. In Section 3, we specify several variants of the classical flat transaction model, we show how to reason about some basic properties of these variants in the situation calculus, and formalize closed and open nested transactions, as well as cooperative transaction hierarchies. Them, in Section 4, we consider several ATMs and show their formalization. In Section 5, we consider an example to illustrate the formalization of a domain using a relational theory for closed nested transactions. In Section 6, we review related work. Finally, Section 7 concludes the paper and discusses some avenues for future work.

## 2 The Specification Framework

### *2.1 The Situation Calculus*

The situation calculus ($\mathcal{L}_{sitcalc}$) is a many-sorted second order language with sorts for $actions$, $situations$, and $objects$ [27,33]. $Actions$ are first order terms consisting of an action function symbol and its arguments, $situations$ are first order terms denoting finite sequences of actions, and $objects$ represents domain specific individuals other than actions and situations. **In formalizing databases, actions correspond to the elementary database operations of inserting, deleting and updating relational tuples, and situations represent the database *log*.** Relations and functions whose truth values vary from situation to situation are called *fluents*, and are denoted by predicate symbols and function symbols with last argument a situation term.

The language has an alphabet with variables and a finite number of constants for each sort, a finite number of function symbols called *action functions* (e.g., $a\_del(accid, branchid, accbal, tellerid, t)$), a finite number of function symbols called *functional fluents*, a finite number of function symbols called *situation independent functions*, a finite number of predicate symbols called *relational fluents* (e.g., $accounts(accid, branchid, accbal, tellerid, s)$), and a finite number of predicate symbols called *situation independent predicates*. Situations are represented using a binary function symbol $do$: $do(\alpha, s)$ denotes the sequence resulting from adding the action $\alpha$ to the sequence $s$. There is a distinguished constant $S_0$ denoting the initial situation; $S_0$ stands for the empty action sequence. The language also includes special predicates $Poss$, and $\sqsubset$; $Poss(a, s)$ means that the action $a$ is possible in the situation $s$, and $s \sqsubset s'$ states that the situation $s'$ is reachable from $s$ by performing some sequence of actions. In database terms, $s \sqsubset s'$ means that $s$ is a proper sublog of the log $s'$.

A situation calculus axiomatization of a domain includes the following set of axioms:

1. For each action function $A(\mathbf{x})$, an *action precondition axiom* of the form: $Poss(A(x_1, \ldots, x_n), s) \equiv \Pi_A(x_1, \ldots, x_n, s)$ where $s$ is the only term of sort situation in $\Pi_A(x_1, \ldots, x_n, s)$.
2. For each fluent $F(\mathbf{x}, s)$, a *successor state axiom* of the form: $F(x_1, \ldots, x_n, do(a, s)) \equiv \Phi_F(x_1, \ldots, x_n, a, s)$ where $s$ is the only term of sort situation in $\Phi_F(x_1, \ldots, x_n, a, s)$.
3. Unique names axioms for actions. For instance: $account\_insert(aid, abal) \neq address\_insert(aid, addr)$.
4. Axioms describing the initial situation, e.g. the initial database: a finite set of sentences whose only situation term is the constant $S_0$.

A set of these axioms, together with a set of domain independent foundational axioms (more on these in Section 3.1), is called a (Markovian) *basic action theory* [20,30].

Finally, as a notational convention in this paper, a free variable will always be implicitly bound by a prenex universal quantifier.

## 2.2 Non-Markovian Control in the Situation Calculus

Including dynamic aspects into database formalization has emerged as an active area of research. In particular, a variety of proposals have been made concerning the formalization of the evolution of databases with respect to update operations.

These approaches can be classified into procedural(essentially due to the work of Abiteboul and Vianu in [35,1–3] and logical [11,39,15,13,17,6,32,5]. Procedural approaches deal with updates at the physical level. Logical approaches generally view updates as a removal and addition of sentences into the logical theory capturing the evolution of the database.

Proposals in [32], and [5] use the language of the situation calculus [26,34]. These proposals use basic action theories for reasoning about actions that rely on two important assumptions: their axioms describe *deterministic* primitive actions, and their execution preconditions and effects depend solely on the current situation. The latter assumption is what control theoreticians call the *Markov property*. Thus both indeterminate and non-Markovian actions are precluded in the theories introduced in [32,5]. However, in formalizing database transactions, one quickly encounters settings where using non-Markovian actions and fluents is unavoidable. For example, a transaction may explicitly execute a $Rollback$ action to go back to the last database state $s'$ in the past in which a $Begin$ action was executed; and an $End$ action is executable only if it closes a bracket opened by a $Begin$ action in the past and no other transaction specific action occurred meanwhile. Thus more than one situation is involved in considering the semantics of actions such as $Begin$ and $Rollback$. Thus one clearly needs to address the issue of non-Markovian actions and fluents explicitly when formalizing database transactions, and researchers using the situation calculus or similar languages to account for updates and transactions are not addressing this need. Moreover, one also needs to accommodate for indeterminate actions that arise in realistic database settings in the form of null values. This issue, however, remains out of the scope of our paper.

One striking feature of the basic action theories of [32] and [5] is their inability to characterize both the truth value of fluents and the actions preconditions in the current situation in terms of more than one past situations. This inability is formally caused by the fact that the predicate $\sqsubset$ is not allowed on the right hand side of successor state and action precondition axioms.

Thus, a first step towards formalizing database transactions is to extend the action precondition axioms of the form $Poss(A(\mathbf{x}), s) \equiv \mathit{\Pi}_A(\mathbf{x}, s)$ by allowing $\mathit{\Pi}_A(\mathbf{x}, s)$ to be a formula with free variables among $\mathbf{x}, s$ that may mention the predicate $\sqsubset$. Similarly, we must also extend the successor state axioms of the form $F(\mathbf{x}, do(a, s)) \equiv \mathit{\Phi}_F(\mathbf{x}, a, s)$ by allowing $\mathit{\Phi}_F(\mathbf{x}, a, s)$ to be a formula with free variables among $\mathbf{x}, a, s$ that may mention the predicate $\sqsubset$.

Following [12], we call such an extension of action precondition and successor state axioms together with the foundational axioms, unique name axioms, and axioms describing the initial situation a *non-Markovian basic action theory*. Non-Markovian basic action theories are suited for expressing non-Markovian control in the situation calculus.

In order to represent relational database transactions, we need some appropriate restrictions on $\mathcal{L}_{sitcalc}$.

**Definition 1** *A* basic relational language *is a subset of* $\mathcal{L}_{sitcalc}$ *that has the following restrictions on the alphabet* $\mathfrak{A}$*:*

*1.* $\mathfrak{A}$ *has a finite number of constants, but at least one.*

2. *$\mathfrak{A}$ has a finite number of action functions.*
3. *$\mathfrak{A}$ has a finite number of relational fluents.*

Fluents now contain a further argument specifying which transaction contributed to its truth value in a given log. So a *basic relational language* is a finite fragment of the situation calculus that is suitable for modeling relational database transactions. In the sequel of this section we shall introduce an extension of non-Markovian basic action theories called *basic relational theories* which is tailored to relational languages and shall devote the subsequent sections to extending them.

For simplicity, we consider basic relational languages whose only primitive update operations correspond to insertion or deletion of tuples into relations. For each such relation $F(\mathbf{x}, t, s)$, where $\mathbf{x}$ is a tuple of objects, $t$ is a transaction argument, and $s$ is a situation argument, a *primitive internal action* is a parameterized primitive action of the situation calculus of the form $F\_ins(\mathbf{x}, t)$ or $F\_del(\mathbf{x}, t)$. Intuitively, $F\_ins(\mathbf{x}, t)$ and $F\_del(\mathbf{x}, t)$ denote the actions of inserting the tuple $\mathbf{x}$ into and deleting it from the relation $F$ by the transaction $t$, respectively; for convenience, we will abbreviate long symbols when necessary (e.g., $account\_ins(\mathbf{x}, t)$ will be abbreviated as $a\_ins(\mathbf{x}, t)$). Below, we will use the following

**Abbreviation 1**

$$writes(a, F, t) =_{df} (\exists \mathbf{x}).a = F\_ins(\mathbf{x}, t) \vee a = F\_del(\mathbf{x}, t),$$

one for each fluent. We distinguish the primitive internal actions from *primitive external actions* which are $Begin(t)$, $Commit(t)$, $End(t)$, and $Rollback(t)$, whose meaning will be clear in the sequel of this section; these are external as they do not specifically affect the content of the database.[1] The argument $t$ is a unique transaction identifier. Finally, the set of fluents of a relational language is partitioned into two disjoint sets, namely a set of *database fluents* and a set of *system fluents*. Intuitively, the database fluents represent the relations of the database domain, while the system fluents are used to formalize the processing of the domain. Usually, any functional fluent in a relational language will always be a system fluent.

## *2.3 Building Blocks for Specifying Transaction Models*

In [9], five building blocks for transaction models are identified: *history*, intertransaction *dependencies*, *visibility* of operations on database objects, *conflict* between operations, and *delegation* of responsibility for objects visible to a transaction. We now show how these building blocks are represented in the situation calculus.

In the situation calculus, the history of [9] corresponds to the log. We extend the basic action theories of [34] to include a specification of relational database transactions, by giving action precondition axioms for external actions such as $Begin(t)$, $End(t)$, $Commit(t)$, $Rollback(t)$, $Spawn(t, t')$, etc. $Commit(t)$ and $Rollback(t)$ are coercive actions that must occur whenever they are possible. We

[1] The terminology internal versus *external* action is also used in [24], though with a different meaning.

also give successor state axioms that state how change occurs in databases in the presence of both internal and external actions. All these axioms provide the *first dimension* of the situation calculus framework for axiomatizing transactions, namely the axiomatization of the effects of transactions on fluents; they also comprise axioms indicating which transactions are conflicting with each other, and what sublogs of the current log are visible; which visible sublogs are delegated to the transactions is expressed implicitly in successor state axioms.

A useful concept that underlies most of the transaction models is that of responsibility over changes operated on data items. For example, in a nested transaction, a parent transaction will take responsibility of changes done by any of its committed children. The only way we can keep track of those responsibilities is to look at the transaction arguments of the actions present in the log. To that end, we introduce a system fluent $responsible(t, a, s)$, which intuitively means that transaction $t$ is responsible for the action $a$ in the log s, which we characterize with an appropriate successor state axiom of the form $responsible(t, a', do(a, s)) \equiv \Phi_{tm}(t, a, a', s)$, where $\Phi_{tm}(t, a, a', s)$ is a transaction model-dependent first order formula whose only free variables are among $t, a, a'$, and $s$. For example, in the flat transactions, we will have the following, simple axiom:

$$responsible(t, a, s) \equiv \bigvee_{A \in \mathcal{A}} (\exists \mathbf{x}) a = A(\mathbf{x}, t)$$

i.e., each transaction is considered responsible for any action whose last argument bears its name; here, $\mathcal{A}$ is the set of actions of the relational language.

To express conflicts between transactions, we need the predicate $termAct(a, t)$ and the system fluents $updConflict(a, a', s)$ and $transConflict(t, t', s)$, whose intuitive meaning is that the action $a$ is a terminal action of $t$, the action $a$ is conflicting with the action $a'$ in $s$, and the transaction $t$ is conflicting with the transaction $t'$ in $s$; their characterization is as follows:

**Abbreviation 2**

$$termAct(a, t) =_{df} a = Commit(t) \lor a = Rollback(t);$$

**Abbreviation 3**

$$updConflict(a, a', s) =_{df}$$
$$\bigvee_{F \in \mathcal{F}} (\exists \mathbf{x}, t, t') \neg [F(\mathbf{x}, t, do(a, do(a', s))) \equiv F(\mathbf{x}, t', do(a', do(a, s)))];$$

*here, $\mathcal{F}$ is the set of database fluents of the relational language; this definition says that two internal actions $a$ and $a'$ conflict in the log $s$ iff the value of the fluents depends on the order in which $a$ and $a'$ appear in $s$.*

**Abbreviation 4**

$$transConflict(t, t', do(a, s)) =_{df} t \neq t' \land responsible(t', a, s) \land$$
$$(\exists a', s')[responsible(t, a', s) \land do(a', s') \sqsubset s \land updConflict(a', a, s)] \lor$$
$$transConflict(t, t', s) \land \neg termAct(a, t);$$

*i.e., transaction $t$ conflicts with transaction $t'$ in the log $s$ iff $t'$ executes an internal action $a$ after $t$ has executed an internal action $a'$ that conflicts with $a$ in the log $s$.*

Notice that we define $updConflict(a, a', s)$ in terms of performing action $a$ and action $a'$ one immediately after the other and vice-versa; in the definition of the predicate $transConflict(t, t', s)$, however, we allow action $a'$ to be executed long before action $a$. This does not mean that actions that are performed between $a'$ and $a$ are irrelevant with respect to update conflicts. Rather, Abbreviation (3) just means that actions $a$ and $a'$ conflict whenever executing one immediately after the other *would* result in a discrepancy in the truth value of at least one of the relational fluents; and Abbreviation (4) allows for the possibility of other update conflicts arising between $a'$ and other actions before the execution of $a$.

A further useful system fluent that we provide in the general framework is $readsFrom(t, t', s)$. This is used in most transaction models as a source of dependencies among transactions, and intuitively means that the transaction $t$ reads a value written by the transaction $t'$ in the log $s$. The axiomatizer must provide a successor state axiom for this fluent depending on the application.

The visibility of portions of the log is characterized by a transaction model-specific system fluent $visible(t, s)$, which intuitively means that the transaction $t$ sees the log $s$. In general, it has the form $visible(t, s) \equiv \mathcal{H}(t, s)$, where $\mathcal{H}(t, s)$ is a condition on the log $s$ depending on the transaction $t$. In the transaction models in which we have $visible(t, s) \equiv true$, no mention will be made of visibility.

The *second dimension* of the situation calculus framework is made of dependencies between transactions. All the dependencies expressed in ACTA ([9]) can also be expressed in the situation calculus. As an example, we have:

**Commit Dependency** of $t$ on $t'$

$$\begin{array}{l} do(Commit(t), s) \sqsubset s^* \supset \\ \quad [do(Commit(t'), s') \sqsubseteq s^* \supset do(Commit(t'), s') \sqsubset do(Commit(t), s)]; \end{array}$$

i.e., If $t$ commits in a log $s^*$, then, whenever $t'$ also commits in $s^*$, $t'$ commits before $t$.

**Strong Commit Dependency** of $t$ on $t'$

$$(\exists s')do(Commit(t'), s') \sqsubset s^* \supset (\exists s)do(Commit(t), s) \sqsubseteq s^*;$$

i.e., If $t'$ commits in a log $s^*$, then $t$ must also commit in $s^*$.

**Rollback Dependency** of $t$ on $t'$

$$(\exists s')do(Rollback(t'), s') \sqsubset s^* \supset (\exists s)do(Rollback(t), s) \sqsubseteq s^*;$$

i.e., If $t'$ rolls back in a log $s^*$, then $t$ must also roll back in that log.

**Weak Rollback Dependency** of $t$ on $t'$

$$\begin{array}{l} do(Rollback(t'), s') \sqsubset s^* \supset \\ \quad \{(\forall s)[s \sqsubset s^* \wedge do(Commit(t), s) \not\sqsubseteq do(Rollback(t'), s')] \supset \\ \qquad\qquad\qquad\qquad (\exists s'')do(Rollback(t), s'') \sqsubseteq s^*\}; \end{array}$$

i.e., If $t'$ rolls back in a log $s^*$, then, whenever $t$ does not commit before $t'$, $t$ must also roll back in $s^*$.

**Begin on Commit Dependency** of $t$ on $t'$

$$do(Begin(t) \sqsubset s^* \supset (\forall s)do(Commit(t'), s) \sqsubset do(Begin(t'), s^*);$$

i.e., If $t$ begins in a log $s^*$, then, $t'$ must commit before the beginning of $t$ in $s^*$.

As we shall see below, all these dependencies are properties of legal database logs of various transaction models.

To control dependencies that may develop among running transactions, we use a set of predicates denoting these dependencies. For example, we use $c\_dep(t, t', s)$, $sc\_dep(t, t', s)$, $r\_dep(t, t', s)$, $wr\_dep(t, t', s)$, and and $bc\_dep(t, t', s)$ to denote the commit, strong commit, rollback, weak rollback, and begin on commit dependencies, respectively. These are system fluents whose truth value is changed by the relevant transaction models by taking into account dependencies generated by the execution of its external actions (*external dependencies*) and those generated by the execution of its internal actions (*internal dependencies*). As an example, in the nested transaction model, we have the following successor state axiom for $wr\_dep(t, t', s)$:

$$\begin{aligned} wr\_dep(t, t', do(a, s)) \equiv\ & a = Spawn(t, t') \vee \\ & wr\_dep(t, t', s) \wedge \neg termAct(a, t) \wedge \neg termAct(a, t'). \end{aligned}$$

This says that a weak rollback dependency of $t$ on $t'$ arises in $do(a, s)$ when either $a$ is the action of $t$ spawning $t'$, or that dependency existed already in $s$ and neither $t$ nor $t'$ terminated with the action $a$.

## 3 Modeling Flat Transactions Models

### *3.1 Axiomatization*

Flat transactions exhibit ACID properties. This section introduces a characterization of flat transactions in terms of theories of the situation calculus. These theories give axioms of flat transaction models that constrain database logs in such a way that these logs satisfy important correctness properties of database transaction, including the ACID properties.

**Definition 2 (Flat Transaction)** *A sequence of database actions is a* flat transaction *iff it is one of the following:*

1. *Atomic transaction:* $[a_1, \ldots, a_n]$*, where the* $a_1$ *must be* $Begin(t)$*, and* $a_n$ *must be either* $Commit(t)$*, or* $Rollback(t)$*;* $a_i, i = 2, \cdots, n-1$*, may be any of the primitive actions, except* $Begin(t)$*,* $Rollback(t)$*, and* $Commit(t)$*; here, the argument t is a unique identifier for the atomic transaction.*
2. *Transaction:* $at_1 \bullet \ldots \bullet at_m$*, where the* $at_i$*,* $1 \leq i \leq m$*, are atomic transactions.*[2]

Flat transactions can be sequenced or run in parallel. Notice that we do not introduce a term of a new sort for transactions, as is the case in [5]; we treat transactions

[2] Given two atomic transactions $A = [A_1, \cdots, A_n]$ and $B = [B_1, \cdots, B_m]$, $A \bullet B$ is an abbreviation for $[A_1, \cdots, A_n, B_1, \cdots, B_m]$.

as run-time activities, whose compile-time counterparts will be GOLOG programs as shown in [18]. We refer to transactions by their names that are of sort *object*.

The axiomatization of a dynamic relational database with flat transaction properties comprises the following classes of axioms:

**Foundational Axioms**. These are constraints imposed on the structure of database logs ([30]):

$$do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2, \tag{1}$$
$$(\forall P).P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \supset (\forall s)P(s), \tag{2}$$
$$\neg(s \sqsubset S_0), \tag{3}$$
$$s \sqsubset do(a, s') \equiv s \sqsubseteq s'. \tag{4}$$

These characterize database logs as finite sequences of updates. Notice that the second axiom is a second-order induction axiom; the third and fourth axioms characterize the subsequence relation $\sqsubset$.

**Integrity Constraints**. These are constraints imposed on the data in the database at a given situation $s$; their set is denoted by $\mathcal{IC}_e$ for constraints that must be enforced at each update execution, and by $\mathcal{IC}_v$ for those that must be verified at the end of the flat transaction.

**Update Precondition Axioms**. There is one for each internal action $A(\mathbf{x}, t)$, with syntactic form

$$Poss(A(\mathbf{x}, t), s) \equiv (\exists t')\Pi_A(\mathbf{x}, t', s) \wedge IC^e(do(A(\mathbf{x}, t), s)) \wedge running(t, s). \tag{5}$$

Here, $\Pi_A(\mathbf{x}, t, s)$ is a first order formula with free variables among $\mathbf{x}, t$, and $s$. These axioms characterize the preconditions of the update $A$; $IC^e(s)$ and $running(t, s)$ are defined as follows:

**Abbreviation 5** $IC^e(s) =_{df} \bigwedge_{IC \in \mathcal{IC}_e} IC(s)$.

**Abbreviation 6**

$$running(t, s) =_{df} (\exists s').do(Begin(t), s') \sqsubseteq s \wedge$$
$$(\forall a, s'')[do(Begin(t), s') \sqsubset do(a, s'') \sqsubset s \supset a \neq Rollback(t) \wedge a \neq End(t)].$$

In a banking Credit/Debit example formalized below (in Section 5), the following states that it is possible to insert a tuple into the $teller$ relation relative to the database log $s$ iff, as a result of performing the actions in the log, that tuple would not already be present in the $teller$ relation, the integrity constraints are satisfied, and transaction $t$ is running.

$$\begin{aligned} Poss(t\_delete(tid, tbal, t), s) \equiv &(\exists t')teller(tid, tbal, t', s) \wedge \\ &IC^e(do(t\_delete(tid, tbal, t), s)) \wedge running(t, s). \end{aligned} \tag{6}$$

**Successor State Axioms**. These have the syntactic form

$$F(\mathbf{x}, t, do(a, s)) \equiv (\exists \mathbf{t'}) \Phi_F(\mathbf{x}, a, \mathbf{t'}, s) \wedge \neg(\exists t'')a = Rollback(t'') \vee \\ (\exists t'')a = Rollback(t'') \wedge restoreBeginPoint(F, \mathbf{x}, t'', s), \tag{7}$$

where $\Phi_F(\mathbf{x}, a, \mathbf{t}, s)$ is a formula with free variables among $\mathbf{x}, a, \mathbf{t}, s$. The formula on the right hand side of (7) is uniform in s, and $\Phi_F(\mathbf{x}, a, \mathbf{t}, s)$ is a formula with free variables among $\mathbf{x}, a, \mathbf{t}, s$; $\Phi_F(\mathbf{x}, a, \mathbf{t}, s)$ stands for the right hand side of the successor state axioms of Section 2.1 and has the following canonical form [34]:

$$\gamma_F^{+}(\mathbf{x}, a, \mathbf{t}, s) \vee F(\mathbf{x}, s) \wedge \neg\gamma_F^{-}(\mathbf{x}, a, \mathbf{t}, s), \tag{8}$$

where $\gamma_F^+(\mathbf{x}, a, \mathbf{t}, s)$ ($\gamma_F^-(\mathbf{x}, a, \mathbf{t}, s)$) denotes a first order formula specifying the conditions that make a fluent $F$ true (false) in the situation following the execution of an update $a$.

There is one successor state axiom for each database relational fluent $F$, and $restoreBeginPoint(F, \mathbf{x}, t, s)$ is defined as follows:

**Abbreviation 7**

$$\begin{aligned} & restoreBeginPoint(F, \mathbf{x}, t, s) =_{df} \\ & \quad [(\exists a^*, s', s^*, t').do(Begin(t), s') \sqsubset do(a^*, s^*) \sqsubseteq s \wedge \\ & \quad\quad writes(a^*, F, t) \wedge F(\mathbf{x}, t', s')] \vee \\ & \quad [(\forall a^*, s^*, s').do(Begin(t), s') \sqsubset do(a^*, s^*) \sqsubseteq s \supset \neg writes(a^*, F, t)] \wedge \\ & \qquad\qquad\qquad (\exists t')F(\mathbf{x}, t', s). \end{aligned}$$

Notice that system fluents have successor state axioms that have to be specified on a case by case basis and do not necessarily have the form (7). Intuitively, $restoreBeginPoint(F, \mathbf{x}, t, s)$ means that the system restores the value that the database fluent $F$ with arguments $\mathbf{x}$ had before the execution of the $Begin$ action of the transaction $t$ in the log $s$ if the transaction $t$ has updated $F$; it keeps the value it had in $s$ otherwise. Given the actual situation $s$, the successor state axioms characterize the truth values of the fluent $F$ in the next situation $do(a, s)$ in terms of all the past situations. In the banking example, the following successor state axiom (9) states that the tuple $(tid, tbal)$ will be in the $teller$ relation relative to the log $do(a, s)$ iff the last database operation $a$ in the log inserted it there, or it was already in the $teller$ relation relative to the log $s$, and $a$ didn't delete it; all this, provided that the operation $a$ is not rolling the database back. In the case the operation $a$ is rolling the database back, the $tellers$ relation will get a value according to the logic of (7).

$$\begin{aligned} & tellers(tid, tbal, t, do(a, s)) \equiv \\ & \quad ((\exists t_1)a = t\_insert(tid, tbal, t_1) \vee (\exists t_2)tellers(tid, tbal, t_2, s) \wedge \\ & \quad\quad \neg(\exists t_3)a = t\_delete(tid, tbal, t_3)) \wedge \neg(\exists t')a = Rollback(t') \vee \\ & \quad (\exists t').a = Rollback(t') \wedge restoreBeginPoint(tellers, (tid, tbal), t', s). \end{aligned} \tag{9}$$

**Precondition Axioms for External Actions**. This is a set of action precondition axioms for the transaction specific actions $Begin(t)$, $End(t)$, $Commit(t)$, and $Rollback(t)$. The external actions of flat transactions have the following precondition axioms:

$$Poss(Begin(t), s) \equiv \neg(\exists s')do(Begin(t), s') \sqsubseteq s, \tag{10}$$

$$Poss(End(t), s) \equiv running(t, s), \tag{11}$$

$$\begin{aligned} &Poss(Commit(t), s) \equiv (\exists s').s = do(End(t), s')\wedge \\ &\bigwedge_{IC \in \mathcal{IC}_v} IC(s) \wedge (\forall t')[sc\_dep(t, t', s) \supset (\exists s'')do(Commit(t'), s'') \sqsubseteq s], \end{aligned} \tag{12}$$

$$\begin{aligned} &Poss(Rollback(t), s) \equiv (\exists s')[s = do(End(t), s')\wedge \\ &\quad \neg \bigwedge_{IC \in \mathcal{IC}_v} IC(s)] \vee (\exists t', s'')[r\_dep(t, t', s) \wedge do(Rollback(t'), s'') \sqsubseteq s]. \end{aligned} \tag{13}$$

**Dependency axioms**. These are transaction model-dependent axioms of the form

$$dep(t, t', do(a, s)) \equiv \mathcal{C}(t, t', a, s), \tag{14}$$

where $\mathcal{C}(t, t', a, s)$ is a condition involving the conflict relation between internal actions of any two transactions $t$ and $t'$, and $dep(t, t', s)$ is one of the dependency predicates $c\_dep(t, t', s)$, $sc\_dep(t, t', s)$, etc. These axioms are used to capture the notion of *recoverability*, *avoiding cascading rollbacks*, etc, of the classical concurrency control theory [4]. For example, to achieve recoverability and avoid cascading rollbacks, the following axioms are used, respectively:

$$r\_dep(t, t', s) \equiv transConflict(t, t', s), \tag{15}$$

$$sc\_dep(t, t', s) \equiv readsFrom(t, t', s). \tag{16}$$

The first axiom says that a transaction conflicting with another transaction generates a rollback dependency, and the second says that a transaction reading from another transaction generates a strong commit dependency. Axioms (15) and (16) generate internal dependencies.

**Unique Names Axioms**. These state that the primitive updates and the objects of the domain are pairwise unequal.

**Initial Database**. This is a set of first order sentences specifying the initial database state. They are completion axioms of the form

$$(\forall \mathbf{x}, t).F(\mathbf{x}, t, S_0) \equiv \mathbf{x} = \mathbf{C}^{(1)} \vee \ldots \vee \mathbf{x} = \mathbf{C}^{(r)}, \tag{17}$$

one for each (database or system) fluent $F$. Here, the $\mathbf{C}^i$ are tuples of constants. Also, $\mathcal{D}_{S_0}$ includes unique name axioms for constants of the database. Axioms of the form (17) say that our theories accommodate a complete initial database state, which is commonly the case in relational databases as unveiled in [31]. This requirement is made to keep the theory simple and to reflect the standard practice in databases. It has the theoretical advantage of simplifying the establishment of logical entailments in the initial database; moreover, it has the practical advantage of facilitating rapid prototyping of the ATMs using Prolog which embodies negation by failure, a notion close to the completion axioms used here.

One striking feature of our axioms is the use of the predicate $\sqsubset$ on the right hand side of action precondition axioms and successor state axioms. That is, they are capturing the notion of a situation being located in the past relative to the current situation which we express with the predicate $\sqsubset$ in the situation calculus. Thus they are capturing non-Markovian control. We call these axioms a *basic relational theory*, and introduce the following:

**Definition 3** *(***Basic Relational Theory***) Suppose* $\mathfrak{R}$ *is a basic relational language with alphabet* $\mathfrak{A}$*. Then a theory* $\mathcal{D} \subseteq \mathfrak{W}$ *is a* non-Markovian basic relational theory *iff it is of the form*

$$\mathcal{D} = \mathcal{D}_f \cup \mathcal{D}_{IC} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{FT} \cup \mathcal{D}_{dep} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$$

*where*

1. $\mathfrak{A}$ *comprises, in addition to the internal actions, the external actions* $Begin(t)$*,* $End(t)$*,* $Commit(t)$*, and* $Rollback(t)$*.*
2. $\mathcal{D}_f$ *is the set of foundational axioms.*
3. $\mathcal{D}_{IC}$ *is a set of integrity constraints* $IC(s)$*. More specifically, we have built-in ICs (*$\mathcal{D}_{IC^e}$*) and generic ICs (*$\mathcal{D}_{IC^v}$*). Built-in ICs are: not null attributes, primary keys, and uniqueness ICs.*
4. $\mathcal{D}_{ap}$ *is a set of non-Markovian action precondition axioms of the form (5), one for each primitive internal action of* $\mathfrak{R}$*.*
5. $\mathcal{D}_{ss}$ *is a set of non-Markovian successor state axioms of the form (7), one for each database fluent of* $\mathfrak{R}$*. Also,* $\mathcal{D}_{ss}$ *includes successor state axioms for all the system fluents of the flat transaction model.*
6. $\mathcal{D}_{FT}$ *is a set of action precondition axioms for the primitive external actions of* $\mathfrak{R}$*.*
7. $\mathcal{D}_{dep}$ *is a set of dependency axioms.*
8. $\mathcal{D}_{una}$ *consists of unique names axioms for objects and for actions.*
9. $\mathcal{D}_{S_0}$ *is an initial relational theory, i.e. a set of completion axioms of the form*

   $$(\forall \mathbf{x}).F(\mathbf{x}, S_0) \equiv \mathbf{x} = \mathbf{C}^{(1)} \vee \ldots \vee \mathbf{x} = \mathbf{C}^{(r)},$$

   *one for each fluent* $F$ *whose interpretation contains* $r$ *n-tuples, together with completion axioms of the form* $(\forall \mathbf{x})\neg F(\mathbf{x}, S_0)$*, one for each fluent* $F$ *whose interpretation is empty. Also,* $\mathcal{D}_{S_0}$ *includes unique name axioms for constants of the database.*

**Definition 4** *(***Relational Database***) A relational database is a pair* $(\mathfrak{R}, \mathcal{D})$*, where* $\mathfrak{R}$ *is a relational language and* $\mathcal{D}$ *is a basic relational theory.*

### *3.2 Legal Flat Transactions*

A fundamental property of $Rollback(t)$ and $Commit(t)$ actions is that, the database system *must* execute them in any database state in which they are possible. In this sense, they are coercive actions, and we call them *system actions*:[3]

**Abbreviation 8**

$$systemAct(a,t) =_{df} a = Commit(t) \vee a = Rollback(t).$$

We constrain legal logs to include these mandatory system actions, as well as the requirement that all actions in the log be possible:

**Abbreviation 9**

$$\begin{aligned} legal(s) =_{df} & (\forall a, s^*)[do(a, s^*) \sqsubseteq s \supset Poss(a, s^*)] \wedge \\ & (\forall a', a'', s', t)[systemAct(a', t) \wedge responsible(t, a', s') \wedge \\ & \quad responsible(t, a'', s') \wedge Poss(a', s') \wedge do(a'', s') \sqsubset s \supset a' = a'']. \end{aligned}$$

### *3.3 Properties*

Simple properties such as well-formedness of atomic transactions [23] can be formulated in the situation calculus and proved as logical consequences of basic relational theories. We first introduce the following abbreviation:

**Abbreviation 10**

$$\begin{aligned} & externalAct(a,t) =_{df} \\ & \quad a = Begin(t) \vee a = End(t) \vee a = Commit(t) \vee a = Rollback(t). \end{aligned}$$

**Theorem 1 (Well-Formedness of Flat Transactions)** *Suppose $\mathcal{D}$ is a basic relational theory. Then*

1. *No external action may occur twice in a legal log; i.e.,*

$$\begin{aligned} & \mathcal{D} \models legal(s) \supset \\ & \quad \{do(a, s') \sqsubset s \wedge do(a, s'') \sqsubset s \wedge externalAct(a, t) \supset s' = s''\}. \end{aligned}$$

2. *There are no dangling $Commit$ or $Rollback$ actions; i.e.,*

$$\begin{aligned} & \mathcal{D} \models legal(s) \supset \\ & \quad \{[do(Commit(t), s') \sqsubset s \supset (\exists s'') do(Begin(t), s'') \sqsubset do(Commit(t), s')] \wedge \\ & \quad\; [do(Rollback(t), s') \sqsubset s \supset (\exists s'') do(Begin(t), s'') \sqsubset do(Rollback(t), s')]\}. \end{aligned}$$

[3] It must be noted that, in reality, a part of rolling back and committing lies with the user and another part lies with the system. So, we could in fact have something like $Rollback_{sys}(t)$ and $Commit_{sys}(t)$ on the one hand, and $Rollback_{usr}(t)$ and $Commit_{usr}(t)$ on the other hand. However, the discussion is simplified by considering only the system's role in executing these actions.

*3. No transaction may commit and then roll back, and conversely; i.e.,*

$$\begin{array}{l}\mathcal{D} \models legal(s) \supset \\ \qquad \{[do(Commit(t), s') \sqsubset s \supset \neg(\exists s'')do(Rollback(t), s'') \sqsubset s] \wedge \\ \qquad\ [do(Rollback(t), s') \sqsubset s \supset \neg(\exists s'')do(Commit(t), s'') \sqsubset s]\}.\end{array}$$

These properties are similar to the fundamental axioms, applicable to all transactions, of [9]. They are well-formedness properties since they rule out all the ill-formed transactions such as

$[Begin(t), a\_ins(A_1, B_1, 1000, T_1), Begin(t), a\_del(A_1, B_1, 1000, T_1), Commit(t)]$,

$[Begin(t), a\_ins(A_1, B_1, 1000, T_1), Commit(t), a\_del(A_1, B_1, 1000, T_1), Commit(t)]$,

$[Begin(t), a\_ins(A_1, B_1, -1000, T_1), Commit(t), a\_del(A_1, B_1, -1000, T_1), Rollback(t)]$, etc.

**Theorem 2** *Suppose $\mathcal{D}$ is a basic relational theory. Then any legal log satisfies the strong commit and rollback dependency properties; i.e.,*

$$\begin{array}{l}\mathcal{D} \models legal(s) \supset \\ \quad (\forall t, t').\{sc\_dep(t, t', s) \supset \\ \qquad [(\exists s')do(Commit(t'), s') \sqsubset s \supset (\exists s^*)do(Commit(t), s^*) \sqsubseteq s]\} \wedge \\ \quad\ \{c\_dep(t, t', s) \supset \\ \qquad [(\exists s')do(Rollback(t'), s') \sqsubset s \supset (\exists s^*)do(Rollback(t), s^*) \sqsubset s]\}.\end{array}$$

Now we turn to the ACID properties, which are the most important properties of flat transactions.

**Theorem 3** (**Atomicity**) *Suppose $\mathcal{D}$ is a relational theory. Then for every database fluent $F$*

$$\begin{array}{l}\mathcal{D} \models legal(s) \supset \\ \quad (\forall t, s_1, s_2)\{do(Begin(t), s_1) \sqsubset do(a, s_2) \sqsubset s \wedge \\ \quad (\exists a^*, s^*)[do(Begin(t), s_1) \sqsubset do(a^*, s^*) \sqsubset do(a, s_2) \wedge writes(a^*, F, t)] \supset \\ \qquad [(a = Rollback(t) \supset ((\exists t_1)F(\mathbf{x}, t_1, do(a, s_2)) \equiv (\exists t_2)F(\mathbf{x}, t_2, s_1))) \wedge \\ \qquad (a = Commit(t) \supset ((\exists t_1)F(\mathbf{x}, t_1, do(a, s_2)) \equiv (\exists t_2)F(\mathbf{x}, t_2, s_2)))]\}.\end{array}$$

This says that rolling back restores any modified database fluent to the value it had just before the last $Begin(t)$ action, and committing endorses the value it had in the situation just before the $Commit(t)$ action.

**Theorem 4 (Consistency)** *Suppose $\mathcal{D}$ is a relational theory. Then All integrity constraints are satisfied at committed logs; i.e.,*

$$\mathcal{D} \models legal(s) \supset \{do(Commit(t), s') \sqsubseteq s \supset \textstyle\bigwedge_{IC \in \mathcal{IC}_v \cup \mathcal{IC}_e} IC(do(Commit(t), s'))\}.$$

**Theorem 5** *$\mathcal{D}$ is satisfiable iff $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \cup \mathcal{D}_{IC}[S_0]$ is.*[4] *In other words, provided the constraints are consistent with the initial database state and unique names for actions, then the entire relational theory is satisfiable, and conversely.*

[4] Here, $\mathcal{D}_{IC}[S_0]$ is the set $\mathcal{D}_{IC}$ relativized to the situation $S_0$.

Some properties of transactions need the notions of committed and rolled back updates. With the predicates $committed(a, s)$ and $rolledBack(a, s)$, we express these notions in the situation calculus using the following definitions:

$$\begin{aligned} &committed(a, s) =_{df} \\ &\qquad (\exists t, s').responsible(t, a, s) \wedge do(Commit(t), s') \sqsubseteq s, \end{aligned} \tag{18}$$

$$\begin{aligned} &rolledBack(a, s) =_{df} \\ &\qquad (\exists t, s').responsible(t, a, s) \wedge do(Rollback(t), s') \sqsubseteq s. \end{aligned} \tag{19}$$

**Theorem 6 (Durability)** *Suppose $\mathcal{D}$ is a relational theory. Then whenever an update is committed or rolled back by a transaction, another transaction can not change this fact:*

$$\begin{aligned} \mathcal{D} \models legal(s) \supset \{do(Rollback(t), s') \sqsubseteq s \wedge \neg responsible(t, a, s) \supset \\ [Committed(a, s') \equiv Committed(a, do(Rollback(t), s'))] \wedge \\ [rolledBack(a, s') \equiv rolledBack(a, do(Rollback(t), s'))]. \end{aligned}$$

**Definition 5 (Serializability)**

$$\begin{aligned} &transConflict^*(t, t', s) =_{df} (\forall C)[(\forall t)C(t, t, s) \wedge \\ &\qquad (\forall s, t, t', t'')[C(t, t'', s) \wedge transConflict(t'', t', s) \supset C(t, t', s)] \supset C(t, t', s)], \\ &serializable(s) =_{df} (\forall t).do(Commit(t), s') \sqsubset s \supset \neg transConflict^*(t, t, s). \end{aligned}$$

**Theorem 7 (Isolation)** *Suppose $\mathcal{D}$ is a relational theory. Then*

$$\mathcal{D} \models legal(s) \supset serializable(s).$$

### *3.4 Flat Transactions with Savepoints*

Flat transactions with savepoints are a variation of flat transactions which provides the user with a new external action $Save(t)$ to establish savepoints in the database log ([14]). The user program can roll back to those savepoints from later database logs. A flat transaction with savepoints is a sequence $[a_1, \ldots, a_n]$ of primitive actions, where $a_1$ must be $Begin(t)$, and $a_n$ must be either $Commit(t)$, or $Rollback(t)$; $a_i, i = 2, \cdots, n-1$, may be any of the primitive actions including $Save(t)$ and $Rollback(t, n)$, except $Begin(t)$, $Commit(t)$, and $Rollback(t)$; $a_{n-2}$ may be $End(t)$.

The external action $Rollback(t, n)$, where $t$ is a transaction, and $n$ is a monotonically increasing number – the savepoint –, brings the database back to the database state corresponding to that savepoint. With this action, we now can roll back with respect to savepoints; thus the precondition axiom for $Rollback(t, n)$, which now has a savepoint as argument, must be specified accordingly. If a $Rollback(t, n)$ action is executed in situation $s$, its effect is that we ignore any situation between some $s'$ and $s$, where $s'$ is the database log corresponding to the savepoint $n$. One way of doing this is to maintain a predicate $Ignore(t, s', s)$ in order to know which parts of the log to skip over. The following action precondition axioms and definition reflect these changes to the corresponding axioms for flat transactions of Section 3.1:

$$Poss(Save(t), s) \equiv running(t, s), \tag{20}$$

$$\begin{aligned} &Poss(Rollback(t), s) \equiv \\ &\quad (\exists s')[s = do(End(t), s') \wedge \neg \bigwedge_{IC \in \mathcal{IC}_v} IC(s)] \vee \\ &\quad (\exists t', s'')[r\_dep(t, t', s) \wedge do(Rollback(t'), s'') \sqsubseteq s] \vee \\ &\quad (\exists t', n, s', s^*, s^{**})[s' \sqsubseteq s^* \sqsubseteq s^{**} \wedge r\_dep(t, t', s^*) \wedge \\ &\qquad s' = sitAtSavePoint(t', n) \wedge do(Rollback(t', n), s^{**}) \sqsubseteq s], \end{aligned} \tag{21}$$

$$\begin{aligned} &Poss(Rollback(t, n), s) \equiv \\ &\quad running(t, s) \wedge (\exists s').s' = sitAtSavePoint(t, n) \wedge \\ &\quad s' \sqsubset s \wedge numOfSavePoints(t, s) \geq n \wedge \neg(\exists s^*, s^{**}).s^* \sqsubseteq s' \sqsubseteq s^{**} \wedge \\ &\quad Ignore(t, s^*, s^{**}), \end{aligned} \tag{22}$$

$$\begin{aligned} &Ignore(t, s', do(a, s)) \equiv s' \sqsubseteq do(a, s) \wedge \\ &\qquad (\exists n).sitAtSavePoint(t, n) = s' \wedge a = Rollback(t, n), \end{aligned} \tag{23}$$

$$\begin{aligned} &numOfSavePoints(t, do(a, s)) = n \equiv a = Begin(t) \wedge n = 1 \vee \\ &\quad a = Save(t) \wedge n = numOfSavePoints(t, s) + 1 \vee \\ &\quad numOfSavePoints(t, s) = n \wedge a \neq Begin(t) \wedge a \neq Save(t), \end{aligned} \tag{24}$$

$$\begin{aligned} &sitAtSavePoint(t, n) = s =_{df} (\exists a, s').numOfSavePoints(t, s) = n \wedge \\ &\qquad s = do(a, s') \wedge (a = Begin(t) \vee a = Save(t)). \end{aligned} \tag{25}$$

In flat transactions with save points, successor state axioms for relations have the following form that reflects changes introduced by the external $Rollback(t, n)$ action.

$$\begin{aligned} &F(\mathbf{x}, t, do(a, s)) \equiv \\ &\quad (\exists \mathbf{t'}).\Phi_F(\mathbf{x}, a, \mathbf{t'}, s) \wedge \neg(\exists t'')a = Rollback(t'') \wedge \\ &\qquad\qquad \neg(\exists t'', n)a = Rollback(t'', n) \vee \\ &\quad (\exists t'')a = Rollback(t'') \wedge restoreBeginPoint(F, \mathbf{x}, t'', s) \vee \\ &\quad (\exists n, t'').a = Rollback(t'', n) \wedge restoreSavePoint(F, \mathbf{x}, n, t'', s), \end{aligned} \tag{26}$$

one for each relation $F$, where $\Phi_F(\mathbf{x}, a, \mathbf{t'}, s)$ is a formula with free variables among $a, s, \mathbf{x}, \mathbf{t'}$; Abbreviation (7) defines $restoreBeginPoint(F, \mathbf{x}, t'', s)$, and $restoreSavePoint(F, \mathbf{x}, n, t'', s)$ is defined as follows:

**Abbreviation 11**

$$\begin{aligned} &restoreSavePoint(F, \mathbf{x}, n, t, s) =_{df} \\ &\qquad (\exists s').s' \sqsubset s \wedge sitAtSavePoint(t, n) = s' \wedge (\exists t')F(\mathbf{x}, t', s'), \end{aligned} \tag{27}$$

where $sitAtSavePoint(t, n)$ is a function returning the log relative to the transaction $t$ at the savepoint $n$, defined by (25); $restoreSavePoint(F, \mathbf{x}, n, t, s)$ means that the value of the fluent $F$ with arguments $\mathbf{x}$ is set back to the value it had at the sublog of $s$ corresponding to the savepoint $n$ established by the transaction $t$.

The dependency axioms have to be adapted to this new setting where dependencies that held previously may no longer hold as a consequence of the partial rollback mechanism; these axioms are now of the form

$$\begin{aligned}&dep(t, t', do(a, s)) \equiv \mathcal{C}(t, t', s) \wedge a \neq Rollback(t) \wedge a \neq Rollback(t') \wedge \\ &\quad \neg(\exists n, s')[(a = Rollback(t, n) \vee a = Rollback(t', n)) \wedge \\ &\quad sitAtSavePoint(t', n) = s' \wedge (\forall s'').s' \sqsubseteq s'' \sqsubseteq s \supset \neg dep(t, t', s'')],\end{aligned} \tag{28}$$

where $\mathcal{C}(t, t')$ and $dep(t, t', s)$ are defined as in (14); we have one such axiom for each dependency predicate.

A basic relational theory for flat transactions with savepoints is as in Definition 3, but where the relational language includes $Save(t)$ and $Rollback(t, n)$ as further actions, the axioms (20) – (25) are added to $\mathcal{D}_{FT}$, the set $\mathcal{D}_{ss}$ is a set of successor state axioms of the form (26), and the set $\mathcal{D}_{dep}$ is a set of dependency axioms of the form (28). All the other axioms of Definition 3 remain unchanged.

*3.4.1 Properties* Now we turn to the ACID properties of flat transactions with savepoints. The introduction of the $Rollback(t, n)$ action modifies some of the previous theorems.

**Lemma 1** *Suppose $\mathcal{D}$ is a relational theory. Then for every relational fluent $F$*

$$\begin{aligned}&\mathcal{D} \models legal(s) \supset \\ &\quad \{do(Begin(t), s_1) \sqsubset do(a, s_2) \sqsubset s \wedge \\ &\qquad (\exists a^*, s^*)[do(Begin(t), s_1) \sqsubset do(a^*, s^*) \sqsubset do(a, s_2) \wedge writes(a^*, F, t)] \supset \\ &\qquad\quad [(\exists n)(a = Rollback(t, n) \wedge sitAtSavePoint(t, n) = s') \supset \\ &\qquad\qquad (((\exists t_1)F(\mathbf{x}, t_1, do(a, s_2)) \equiv (\exists t_2)F(\mathbf{x}, t_2, s')))]\}.\end{aligned}$$

This tells us that $Rollback(t, n)$ does not fall under the all-or-nothing logic that characterizes flat transactions since the situation at a given savepoint of a transaction is not necessarily the same as the situation at the beginning of that transaction.

Note that Theorem 3 continues to hold for flat transactions with savepoints. Hence, from Theorem 3 and Lemma 1, we have the following

**Corollary 1** *(***Atomicity of Transactions with Savepoints***) Suppose $\mathcal{D}$ is a relational theory. Then for every database fluent $F$*

$$\begin{aligned}&\mathcal{D} \models legal(s) \supset \\ &\quad \{do(Begin(t), s_1) \sqsubset do(a, s_2) \sqsubset s \wedge \\ &\qquad (\exists a^*, s^*)[do(Begin(t), s_1) \sqsubset do(a^*, s^*) \sqsubset do(a, s_2) \wedge writes(a^*, F, t)] \supset \\ &\qquad\quad [[a = Rollback(t) \supset ((\exists t_1)F(\mathbf{x}, t_1, do(a, s_2)) \equiv (\exists t_2)F(\mathbf{x}, t_2, s_1))] \wedge \\ &\qquad\quad [(\exists n)(a = Rollback(t, n) \wedge sitAtSavePoint(t, n) = s') \supset \\ &\qquad\qquad (((\exists t_1)F(\mathbf{x}, t_1, do(a, s_2)) \equiv (\exists t_2)F(\mathbf{x}, t_2, s')))] \\ &\qquad\quad [a = Commit(t) \supset ((\exists t_1)F(\mathbf{x}, t_1, do(a, s_2)) \equiv (\exists t_2)F(\mathbf{x}, t_2, s_2))]]\}.\end{aligned}$$

**Lemma 2** *Suppose $\mathcal{D}$ is a relational theory. Then for every database fluent $F$*

$$\begin{aligned}
&\mathcal{D} \models legal(s) \supset \\
&\quad \{do(Rollback(t,n),s') \sqsubset s \wedge sitAtSavePoint(t,n) \sqsubset s' \supset \\
&\quad ((\exists t_1)F(\mathbf{x},t_1,do(Rollback(t,n),s_2)) \equiv (\exists t_2)F(\mathbf{x},t_2,sitAtSavePoint(t,n)))\wedge \\
&\qquad\qquad (\forall a,s^*)[sitAtSavePoint(t,n) \sqsubset do(a,s^*) \sqsubseteq s_2 \supset a \neq Commit(t)]]\}.
\end{aligned}$$

Theorem 6, which also holds for flat transactions with savepoints, and Lemma 2 characterize the durability of flat transactions with Savepoints.

The consistency Theorem 4 also holds for flat transactions with savepoints, as do Theorems 5 and 7.

The following theorem establishes a fundamental property of transactions with savepoints: if a transaction rolls back to a given savepoint, say, $n$, all the updates on the way back to the situation corresponding to $n$ are aborted, and no future rollback to the situations generated by these updates are possible.

**Theorem 8** *Suppose $\mathcal{D}$ is a relational theory. Then*

$$\begin{aligned}
&\mathcal{D} \models legal(s) \supset \\
&\quad \{do(Rollback(t,n),s') \sqsubset s \quad \supset \\
&\qquad [\neg(\exists n^*,s^*).do(Rollback(t,n),s') \sqsubset do(Rollback(n^*),s^*) \sqsubset s\wedge \\
&\qquad\; sitAtSavePoint(n) \sqsubseteq sitAtSavePoint(n^*) \sqsubset do(Rollback(t,n),s')]\}.
\end{aligned}$$

## *3.5 Chained Flat Transactions*

A chained flat transaction is a sequence $[a_1,\ldots,a_n]$ of primitive actions, where $a_1$ must be $Begin(t)$, and $a_n$ must be either $Commit(t)$, or $Rollback(t)$; $a_i, i = 2,\cdots,n-1$, may be any of the primitive actions including $Chain(t)$, except $Begin(t)$ and $Commit(t)$.

Chained flat transactions are equivalent to the special case of flat transactions with save points, where only the most recent savepoint is restored. The intuition behind chained transactions is to allow committing work done so far, thus waiving any further right to execute a rollback over the committed logs ([14]). The new external action $Chain(t)$ is used by the programmer to commit work done so far and continue with work yet to be done. For any $s$, we call the situation $do(Chain(t),s)$ a chaining situation of transaction $t$.

The following action precondition axioms capture the essence of chained flat transactions:

$$\begin{aligned}
&Poss(Rollback(t), s) \equiv (\exists s')[s = do(End(t), s') \wedge \neg \bigwedge_{IC \in \mathcal{IC}_v} IC(s)] \vee \\
&\quad (\exists t', s^*, s^{**}).do(Chain(t'), s^*) \sqsubset do(Rollback(t'), s^{**}) \sqsubseteq s \wedge \\
&\qquad [(\forall a, s'').do(Chain(t'), s^*) \sqsubset do(a, s'') \sqsubset s^{**} \supset a \neq Chain(t)] \wedge \\
&\qquad [(\exists s'').do(Chain(t'), s^*) \sqsubset s'' \sqsubset s^{**} \wedge r\_dep(t, t', s'')],
\end{aligned} \tag{29}$$

$$\begin{aligned}
&Poss(Chain(t), s) \equiv \bigwedge_{IC \in \mathcal{IC}_v} IC(s) \wedge running(t, s) \wedge \\
&\quad (\forall t').c\_dep(t, t', s) \supset \\
&\qquad (\exists s'')\{do(Commit(t'), s'') \sqsubseteq s \vee [do(Chain(t'), s'') \sqsubseteq s \wedge \\
&\qquad\quad (\forall a^*, s^*)(do(Chain(t'), s'') \sqsubset do(a^*, s^*) \sqsubset s \supset \\
&\qquad\qquad a^* \neq Chain(t) \wedge \neg c\_dep(t, t', s^*))]\}.
\end{aligned} \tag{30}$$

The later axiom is particularly critical: it prevents the user from chaining a transaction $t$ that is commit-dependent on another transaction $t'$ that has not committed before the last chaining situation of $t$. Axioms for $Begin(t)$, $End(t)$ and $Commit(t)$ remain unchanged.

Successor state axioms for fluents of chained flat transactions have the form (7), but with a different definition for $restoreBeginPoint$:

**Abbreviation 12**

$$\begin{aligned}
&restoreBeginPoint(F, \mathbf{x}, t, s) =_{df} \\
&\quad (\exists a, s').(a = Begin(t) \vee a = Chain(t)) \wedge \\
&\quad (\forall a^*, s^*)[s' \sqsubset do(a^*, s^*) \sqsubset s \supset a^* \neq Chain(t) \wedge a^* \neq Begin(t)] \wedge \\
&\quad \{[(\exists a^*, s^*, t').do(a, s') \sqsubseteq s \wedge [do(a, s') \sqsubset do(a^*, s^*) \sqsubseteq s \wedge writes(a^*, F, t)] \wedge \\
&\qquad\qquad F(\mathbf{x}, t', s')] \vee \\
&\quad [(\forall a^*, s^*).do(a, s') \sqsubset do(a^*, s^*) \sqsubseteq s \supset \neg writes(a^*, F, t)] \wedge (existst')F(\mathbf{x}, t', s)\}.
\end{aligned}$$

The dependency axioms must now be adapted to this setting where dependencies that held previously may no longer hold as a consequence of the system rolling back to the last chaining situation; these axioms are now of the form

$$\begin{aligned}
&dep(t, t', do(a, s)) \equiv \\
&\quad \mathcal{C}(t, t', s) \wedge a \neq Rollback(t) \wedge a \neq Rollback(t') \wedge a \neq Chain(t'),
\end{aligned} \tag{31}$$

one for each dependency predicate; $\mathcal{C}(t, t')$ and $dep(t, t', s)$ are defined as in (14).

A basic relational theory for chained flat transactions is as in Definition 3, where $\mathfrak{A}$ comprises $Chain(t)$ as a further action, the set $\mathcal{D}_{FT}$ is modified accordingly to accommodate the new axioms (29)–(30), the set $\mathcal{D}_{ss}$ is now a set of successor state axioms that reflects the changes brought by Abbreviation 12, and the set $\mathcal{D}_{dep}$ is a set of dependency axioms of the form (31). All the other axioms of Definition 3 remain unchanged.

The following is a property specific to chained transactions. It captures the intuition behind chained transactions which is that whenever chained, a database transaction can never roll back over the last chaining situation.

**Theorem 9** *(***Durability of Chaining Situations***) Suppose $\mathcal{D}$ is a relational theory for chained flat transactions. Then, for all database fluents $F$*

$$\begin{aligned}
\mathcal{D} \models & legal(s) \supset \\
& \{do(Chain(t), s') \sqsubset do(Rollback(t), s'') \sqsubseteq s \wedge \\
& \neg(\exists s^*) do(Chain(t), s') \sqsubset do(Chain(t), s^*) \sqsubset do(Rollback(t), s'') \supset \\
& \quad ((\exists t') F(\mathbf{x}, t', do(Rollback(t), s'')) \equiv (\exists t'') F(\mathbf{x}, t'', do(Chain(t), s')))]\}.
\end{aligned}$$

## 4 Modeling Advanced Transaction Models

### *4.1 Closed Nested Transactions*

The main idea conveyed by the notion of nested transactions is that of levels of abstractions: each nesting in the hierarchy of nested transactions corresponds to a level of abstraction from the details of the action execution.

Nested transactions ([28]) are the best known example of ATMs. A nested transaction is a set of transactions (called subtransactions) forming a tree structure, meaning that any given transaction, the parent, may spawn a subtransaction, the child, nested in it. A child commits only if its parent has committed. If a parent transaction rolls back, all its children are rolled back. However, if a child rolls back, the parent may execute a recovery procedure of its own. Each subtransaction, except the root, fulfills the A, C, and I among the ACID properties. The root (level 1) of the tree structure is the only transaction to satisfy all of the ACID properties. This version of nested transactions is called closed because of this inability of subtransactions to durably commit independently of the outcome of the root transaction ([38]). This section deals with closed nested transactions (CNTs), open nested transactions will be the topic of the next section.

A root transaction $t$ is a sequence $[a_1, \ldots, a_n]$ of primitive actions, where $a_1$ must be $Begin(t)$, and $a_n$ must be either $Commit(t)$, or $Rollback(t)$; $a_i, i = 2, \cdots, n-1$, may be any of the primitive actions, except $Begin(t)$, $Commit(t)$, and $Rollback(t)$, but including $Spawn(t, t')$, $Rollback(t')$, and $Commit(t')$, with $t \neq t'$. A child transaction $t$ is a sequence $[a_1, \ldots, a_n]$ of primitive actions, where $a_1$ must be $Spawn(t', t)$, and $a_n$ must be either $Commit(t)$, or $Rollback(t)$; $a_i, i = 2, \cdots, n-1$, may be any of the primitive actions, except $Spawn(t, t')$, $Commit(t)$, and $Rollback(t)$, but including $Spawn(t^*, t^{**})$, $Rollback(t^{**})$, and $Commit(t^{**})$, with $t \neq t^{**}$. We capture the typical relationships that hold between transactions in the hierarchy of a nested transaction with the system fluents $transOf(t, a, s)$, $parent(t, t', s)$ and $ancestor(t, t', s)$, which are introduced in the following successor state axiom and abbreviation, respectively:

$$transOf(t,a,s) \equiv (\exists a').a = a'(\mathbf{x},t), \tag{32}$$

$$\begin{aligned} &parent(t,t',do(a,s)) \equiv a = Spawn(t,t')\vee \\ &\qquad parent(t,t',s) \wedge \neg termAct(a,t) \wedge \neg termAct(a,t'), \end{aligned} \tag{33}$$

$$\begin{aligned} &ancestor(t,t',s) =_{df} (\forall A)[(\forall t)A(t,t,s)\wedge \\ &\quad (\forall s,t,t',t'')[A(t,t'',s) \wedge parent(t'',t',s) \supset A(t,t',s)] \supset A(t,t',s)]. \end{aligned} \tag{34}$$

Responsibility over actions that are executed and conflicts between transactions are specified with the following axioms:

$$\begin{aligned} &responsible(t,a',do(a,s)) \equiv \\ &\quad transOf(t,a',s) \wedge \neg(\exists t^*)parent(t,t^*,s)\vee \\ &\quad (\exists t^*)[parent(t,t^*,s) \wedge a = Commit(t^*) \wedge responsible(t^*,a')]\vee \\ &\quad responsible(t,a',s) \wedge \neg termAct(a,t), \end{aligned} \tag{35}$$

$$\begin{aligned} &transConflictNT(t,t',do(a,s)) \equiv \\ &\quad t \neq t' \wedge responsible(t',a,s)\wedge \\ &\quad (\exists a',s')[responsible(t,a',s) \wedge updConflict(a',a,s) \wedge do(a',s') \sqsubset s]\wedge \\ &\quad \neg responsible(t,a,s) \wedge running(t',s)\wedge \\ &\qquad\qquad ((\exists t'')parent(t,t'',s) \supset \neg ancestor(t,t',s))\vee \\ &\quad transConflictNT(t,t',s) \wedge \neg termAct(a,t). \end{aligned} \tag{36}$$

Intuitively, (36) means that transaction $t$ conflicts with transaction $t'$ in the log $s$ iff $t$ and $t'$ are not equal, internal actions they are responsible for are conflicting in $s$, $t$ is not responsible for the action of $t'$ it is conflicting with, $t'$ is running; moreover, a transaction cannot conflict with actions his ancestors are responsible for. Due to the presence of the new external action $Spawn$, we need to redefine $running(t,s)$ as follows:

**Abbreviation 13**

$$\begin{aligned} &running(t,s) =_{df} (\exists s').\{do(Begin(t),s') \sqsubseteq s\wedge \\ &\quad (\forall a,s'')[do(Begin(t),s') \sqsubset do(a,s'') \sqsubset s \supset a \neq Rollback(t) \wedge a \neq End(t)]\vee \\ &\quad (\exists t').do(Spawn(t',t),s') \sqsubseteq s\wedge \\ &\quad (\forall a,s'')[do(Spawn(t',t),s') \sqsubset do(a,s'') \sqsubset s \supset a \neq Rollback(t) \wedge a \neq End(t)]\}. \end{aligned}$$

Now the external actions of closed nested transactions have the following precondition axioms:

$$\begin{aligned}Poss(Begin(t), s) \equiv{}& \neg(\exists t')parent(t', t, s)\wedge \\ & [s = S_0 \vee (\exists s', t').t \neq t' \wedge do(Begin(t'), s') \sqsubset s],\end{aligned} \tag{37}$$

$$\begin{aligned}Poss(Spawn(t, t'), s) \equiv{}& t \neq t'\wedge \\ & (\exists s', t'')[do(Begin(t), s') \sqsubset s \vee do(Spawn(t'', t), s') \sqsubset s],\end{aligned} \tag{38}$$

$$Poss(End(t), s) \equiv running(t, s), \tag{39}$$

$$\begin{aligned}Poss(Commit(t), s) \equiv{}& (\exists s').s = do(End(t), s') \wedge \bigwedge_{IC \in \mathcal{IC}_v} IC(s)\wedge \\ & (\forall t')[sc\_dep(t, t', s) \supset (\exists s'')do(Commit(t'), s'') \sqsubseteq s]\wedge \\ & (\forall t')[c\_dep(t, t', s) \wedge \neg(\exists s^*)do(Rollback(t'), s^*) \sqsubseteq s \supset \\ & (\exists s')do(Commit(t'), s') \sqsubset s)],\end{aligned} \tag{40}$$

$$\begin{aligned}Poss(Rollback(t), s) \equiv{}& (\exists s').s = do(End(t), s') \wedge \bigwedge_{IC \in \mathcal{IC}_v} IC(s)\vee \\ & (\exists t', s'').r\_dep(t, t', s) \wedge do(Rollback(t'), s'') \sqsubset s'\vee \\ & (\exists t', s^*).wr\_dep(t, t', s) \wedge do(Rollback(t'), s^*) \sqsubset s\wedge \\ & \neg(\exists s^{**})do(Commit(t), s^{**}) \sqsubset do(Rollback(t'), s^*).\end{aligned} \tag{41}$$

Dependency axioms characterizing the system fluents $r\_dep(t, t', s)$, $c\_dep(t, t', s)$, $sc\_dep(t, t', s)$, and $wr\_dep(t, t', s)$ are:

$$r\_dep(t, t', s) \equiv transConflictNT(t, t', s), \tag{42}$$

$$sc\_dep(t, t', s) \equiv readsFrom(t, t', s), \tag{43}$$

$$\begin{aligned}c\_dep(t, t', do(a, s)) \equiv{}& a = Spawn(t, t')\vee \\ & c\_dep(t, t', s) \wedge \neg termAct(a, t) \wedge \neg termAct(a, t'),\end{aligned} \tag{44}$$

$$\begin{aligned}wr\_dep(t, t', do(a, s)) \equiv{}& a = Spawn(t', t)\vee \\ & wr\_dep(t, t', s) \wedge \neg termAct(a, t) \wedge \neg termAct(a, t').\end{aligned} \tag{45}$$

As an example of what they mean, the last axiom says that a transaction spawning another transaction generates a weak rollback dependency of the later one on the first one, and this dependency ends when either transactions execute a terminating action.

The successor state axioms for nested transactions are of the form:

$$\begin{aligned}F(\mathbf{x}, t, do(a, s)) \equiv{}& (\exists \mathbf{t'})\Phi_F(\mathbf{x}, a, \mathbf{t'}, s) \wedge \neg(\exists t'')a = Rollback(t'')\vee \\ & [(\exists t'').a = Rollback(t'') \wedge \neg(\exists t^*)parent(t^*, t'', s)\wedge \\ & restoreBeginPoint(F, \mathbf{x}, t'', s)]\vee \\ & [(\exists t'').a = Rollback(t'')\wedge(\exists t^*)parent(t^*, t'', s)\wedge \\ & restoreSpawnPoint(F, \mathbf{x}, t'', s)],\end{aligned} \tag{46}$$

one for each database fluent of the relational language. Here $\Phi_F(\mathbf{x}, a, \mathbf{t}, s)$ is a formula with free variables among $\mathbf{x}, a, \mathbf{t}$, and $s$; $restoreBeginPoint(F, \mathbf{x}, t, s)$ is defined in (7), and $restoreSpawnPoint(F, \mathbf{x}, t, s)$ is the following

**Abbreviation 14**

$$\begin{aligned}
&restoreSpawnPoint(F, \mathbf{x}, t, s) =_{df} \\
&\quad [(\exists a^*, s', s^*, t', t^*).do(Spawn(t', t), s') \sqsubset do(a^*, s^*) \sqsubseteq s \\
&\qquad \wedge writes(a^*, F, t) \wedge F(\mathbf{x}, t^*, s')] \vee \\
&\quad [(\forall a^*, s^*, s', t').do(Spawn(t', t), s') \sqsubset do(a^*, s^*) \sqsubseteq s \supset \\
&\qquad\qquad \neg writes(a^*, F, t)] \wedge (\exists t^*) F(\mathbf{x}, t^*, s).
\end{aligned}$$

A basic relational theory for nested transactions is defined as in Section 3, but where the relational language includes $Spawn(t, t')$ as a further action, and the axioms (37) – (38) replace axioms (10) – (13), the axioms (42) – (45) replace the axioms (15) – (16), and the set $\mathcal{D}_{ss}$ is a set of successor state axioms of the form (46). All the other axioms of Section 3 remain unchanged.

Now we state some of the properties of nested transactions as an illustration of how such properties are formulated in the situation calculus. Similarly to Theorem 2, we can show that a basic relational theory for nested transactions logically implies the commit and weak rollback dependency properties.

**Theorem 10** *(***Atomicity of Nested Transactions***) Suppose $\mathcal{D}$ is a relational theory for nested transactions. Then for every database fluent $F$*

$$\begin{aligned}
&\mathcal{D} \models legal(s) \supset \\
&\quad (\forall t, s_1, s_2)\{[s' = do(Begin(t), s_1) \vee s' = do(Spawn(t), s_1)] \wedge \\
&\quad s' \sqsubset do(a, s_2) \sqsubset s \wedge \\
&\quad (\exists a^*, s^*)[s' \sqsubset do(a^*, s^*) \sqsubset do(a, s_2) \wedge writes(a^*, F, t)] \supset \\
&\qquad [(a = Rollback(t) \supset ((\exists t_1) F(\mathbf{x}, t_1, do(a, s_2)) \equiv (\exists t_2) F(\mathbf{x}, t_2, s_1))) \wedge \\
&\qquad (a = Commit(t) \supset ((\exists t_1) F(\mathbf{x}, t_1, do(a, s_2)) \equiv (\exists t_2) F(\mathbf{x}, t_2, s_2)))]\}.
\end{aligned}$$

**Theorem 11 (No-Orphan-Commit: [9])** *Suppose $\mathcal{D}$ is a relational theory. Then, whenever a child's parent terminates before the parent does, the child is rolled back;i.e.,*

$$\begin{aligned}
&\mathcal{D} \models legal(s) \supset \\
&\quad \{parent(t, t', s) \wedge termAct(a, t) \wedge \\
&\qquad do(Commit(t'), s') \not\sqsubseteq do(a, s'') \sqsubset s \supset (\exists s^*) do(Rollback(t'), s^*) \sqsubset s\}.
\end{aligned}$$

This property, combined with the atomicity of all subtransactions of the nested transaction tree (i.e. Theorem 10), leads to the fact that, should a root transaction roll back, then so must all its subtransactions, also the committed ones. This is where the D in the ACID acronym is relaxed for subtransactions.

**Definition 6 (Serializability of Nested Transactions)**

$$transConflictNT^*(t,t',s) =_{df} (\forall C)[(\forall t)C(t,t,s) \wedge (\forall s,t,t',t'')[C(t,t'',s) \wedge transConflictNT(t'',t',s) \supset C(t,t',s)] \supset C(t,t',s)],$$

$$serializableNT(s) =_{df} (\forall t).do(Commit(t),s') \sqsubset s \supset \neg transConflictNT^*(t,t,s).$$

**Theorem 12 (Isolation of Nested Transactions)** *Suppose $\mathcal{D}$ is a relational theory for nested transactions. Then*

$$\mathcal{D} \models legal(s) \supset serializableNT(s).$$

## *4.2 Cooperative Transaction Hierarchy*

The *cooperative transaction hierarchy* (CTH: [29]) model has been proposed for supporting cooperative applications in the context of CAD. A cooperative nesting of transactions is a nesting, where sibling subtransactions are allowed to interact. A cooperative transaction hierarchy is structured as a rooted tree whose leaf nodes, the *cooperative transactions* (CTs), represent the transactions at the level of individual designers,and whose internal nodes, the *transaction groups* (TGs), are each a set of children (CTs or CGs) that cooperate to perform a single task. There is no central correctness criterion in a CTH; instead, each TG has its own, user-defined correctness criteria. A CG is not atomic; it performs specific tasks via its members, enforces its own correctness criterion, and organizes cooperation among its members; moreover, it keeps private copies of the objects that its members acquire at creation time, and is a unit of recovery by managing its own recoverability. A TG correctness is expressed in terms of patterns and conflicts. Patterns specify interleavings of actions that must occur, and conflicts specify those interleavings that must not occur. A TG's log is correct iff it satisfies all its pattern specifications and satisfies none of its conflict specifications. A child passes its copy to the parent upon committing, at which time that copy subsumes the parent's copy. Ultimately, the copy of the root TG will be subsumed when the entire design commits. We now give a situation calculus characterization of CTHs.

We have two new external actions: $Join(t,t',n)$, and $Leave(t,t')$, where $t$ and $t'$ are transactions, and $n$ indicates whether the joining node is CT or TG. A root transaction $t$ is a sequence $[a_1,\ldots,a_n]$ of primitive actions, where $a_1$ must be $Begin(t)$, and $a_n$ must be either $Commit(t)$, or $Rollback(t)$; $a_i, i = 2,\cdots,n-1$, may be any of the primitive actions, except $Begin(t)$, $Commit(t)$, and $Rollback(t)$, but including $Join(t',t,n)$, $Rollback(t')$, and $Commit(t')$, with $t \neq t'$. A CT or a TG $t$ is a sequence $[a_1,\ldots,a_n]$ of primitive actions, where $a_1$ must be $Join(t,t')$, and $a_n$ must be either $Commit(t)$, or $Rollback(t)$; $a_i, i = 2,\cdots,n-1$, may be any of the primitive actions, except $Join(t,t')$, $Commit(t)$, and $Rollback(t)$, but including $Join(t^*,t^{**})$, $Rollback(t^*)$, and $Commit(t^*)$, with $t \neq t'$, $t^* \neq t^{**}$, and $t^* \neq t$.

The external actions of roots, and CTs and TGs, are enumerated as follows, respectively.

**Abbreviation 15**

$$
\begin{aligned}
&externalActR(a,t) =_{df}\\
&\qquad a = Begin(t) \vee a = End(t) \vee a = Commit(t) \vee a = Rollback(t).
\end{aligned}
$$

**Abbreviation 16**

$$
\begin{aligned}
&externalActC(a,t) =_{df}\\
&\qquad (\exists t', n)a = Join(t,t',n) \vee (\exists t')a = Leave(t,t') \vee a = End(t) \vee\\
&\qquad a = Commit(t) \vee a = Rollback(t).
\end{aligned}
$$

We continue to capture the typical relationships that hold between transactions in the CTH model with the same fluents $parent(t,t',s)$ and $ancestor(t,t',s)$ as in nested transactions, but now with a slightly different successor state axiom for $parent(t,t',s)$.

$$
\begin{aligned}
&parent(t,t',do(a,s)) \equiv\\
&\qquad (\exists n)a = Join(t',t,n) \vee parent(t,t',s) \wedge \neg a \neq Leave(t',t).
\end{aligned} \tag{47}
$$

Furthermore, we need the fluents $transGroup(t,s)$ and $coopTrans(t,s)$ which intuitively tell whether a transaction is a TG or a CT; these have the following successor state axioms:

$$
\begin{aligned}
transGroup(t,do(a,s)) \equiv\ & (\exists t')a = Join(t,t',TG) \vee\\
& transGroup(t,s) \wedge \neg(\exists t')a = Leave(t,t'),
\end{aligned} \tag{48}
$$

$$
\begin{aligned}
coopTrans(t,do(a,s)) \equiv\ & (\exists t')a = Join(t,t',CT) \vee\\
& coopTrans(t,s) \wedge \neg(\exists t')a = Leave(t,t').
\end{aligned} \tag{49}
$$

A user-defined predicate $transConflictCTH(t,t',s)$ must be provided, where $t$ and $t'$ are transactions; intuitively, $transConflictCTH(t,t',s)$ means that transactions $t$ and $t'$ conflict in the log $s$. As an example of such a definition of this predicate, the following axiom captures the cooperative serializability property of [25]:

$$
\begin{aligned}
&transConflictCTH(t,t',s) \equiv (\exists t'').t \neq t' \wedge transGroup(t'',s) \wedge\\
&\quad \{\neg parent(t'',t) \wedge \neg parent(t'',t') \wedge transConflictNT(t,t',s) \vee\\
&\quad (\exists t^*)[\neg parent(t'',t) \wedge parent(t'',t') \wedge parent(t'',t^*) \wedge\\
&\qquad\qquad transConflictNT(t,t^*,s)] \vee\\
&\quad (\exists t^*)[parent(t'',t) \wedge \neg parent(t'',t') \wedge parent(t'',t^*) \wedge\\
&\qquad\qquad transConflictNT(t^*,t',s)]\}.
\end{aligned} \tag{50}
$$

Intuitively, (50) means that transaction $t$ conflicts with transaction $t'$ in the log $s$ iff $t$ and $t'$ are not equal and there is a transaction group $t^*$ such that: (1) either $t$ and $t'$ do not have $t''$ as parent, in which case they conflict in the usual way of closed nested transactions; or (2) $t'$ has $t''$ as parent and $t$ does not, but $t'$ has a cousin $t^*$ with which $t$ is conflicting in the usual way of nested transactions; or else (3) $t$ has

$t''$ as parent and $t'$ does not, but $t$ has a cousin $t^*$ which is conflicting with $t'$ in the usual way of nested transactions.

The pattern specifications that must be verified and the conflict specifications that must be avoided are captured in action precondition axioms. Suppose $\mathcal{P}(t, s)$ and $\mathcal{C}(t, s)$ denote the pattern and conflict specifications for a TG $t$, respectively. Then precondition axioms for internal actions are of the form

$$\begin{aligned}
&Poss(A(\mathbf{x},t),s) \equiv \Pi_A(\mathbf{x},t,s) \wedge IC^e(do(A(\mathbf{x},t),s)) \wedge \\
&\quad \{[(\exists t^*).parent(t^*,t,s) \wedge \mathcal{P}(t^*, do(A(\mathbf{x},t),s)) \wedge \\
&\qquad\qquad \neg\mathcal{C}(t^*, do(A(\mathbf{x},t),s)) \wedge running(t,s)] \vee \\
&\quad [\neg(\exists t^*).parent(t^*,t,s) \wedge \mathcal{P}(t, do(A(\mathbf{x},t),s)) \wedge \\
&\qquad\qquad \neg\mathcal{C}(t, do(A(\mathbf{x},t),s)) \wedge running(t,s)]\}.
\end{aligned} \tag{51}$$

In CTHs, the following dependencies must be maintained among transactions: a rollback dependency of a child on its parent, and a weak commit dependency of a parent on all its children. A **Weak Commit Dependency** of $t$ on $t'$ is characterized as follows:

$$\begin{aligned}
&do(Commit(t),s) \sqsubset s^* \supset \\
&\quad [do(Rollback(t'),s') \not\sqsubseteq s^* \supset do(Commit(t'),s') \sqsubset do(Commit(t),s)];
\end{aligned}$$

i.e., If $t$ commits in a log $s^*$, then, whenever $t'$ does not roll back in $s^*$, $t'$ commits before $t$.

Now the external actions of a CTH have the following precondition axioms:

$$\begin{aligned}
&Poss(Begin(t),s) \equiv \neg(\exists t')parent(t',t,s) \wedge \\
&\qquad [s = S_0 \vee (\exists s',t').t \neq t' \wedge do(Begin(t'),s') \sqsubset s],
\end{aligned} \tag{52}$$

$$\begin{aligned}
&Poss(End(t),s) \equiv \\
&\qquad (\exists s')\{do(Begin(t),s') \sqsubseteq s \wedge \neg(\exists s'')do(End(t),s'') \sqsubseteq s' \wedge \\
&\qquad (\forall a,s^*)[do(Begin(t),s') \sqsubset do(a,s^*) \sqsubset s \supset \\
&\qquad\qquad\qquad\qquad \neg externalActR(a,t)]\} \vee \\
&\qquad (\exists s',t',n)\{do(Join(t',t,n),s') \sqsubseteq s \wedge \neg(\exists s'')do(End(t),s'') \sqsubseteq s' \wedge \\
&\qquad (\forall a,s^*)[(do(Join(t',t,n),s') \sqsubset do(a,s^*) \sqsubset s) \supset \\
&\qquad\qquad\qquad\qquad \neg externalActC(a,t)]\},
\end{aligned} \tag{53}$$

$$\begin{aligned}
&Poss(Commit(t),s) \equiv (\exists s').s = do(End(t),s') \wedge \bigwedge_{IC \in \mathcal{IC}_v} IC(s) \wedge \\
&\qquad (\forall t').wc\_dep(t,t',s) \wedge \neg(\exists s^*)do(Rollback(t'),s^*) \sqsubseteq s' \supset \\
&\qquad\qquad\qquad\qquad (\exists s'')do(Commit(t'),s'') \sqsubseteq s',
\end{aligned} \tag{54}$$

$$\begin{aligned}
&Poss(Rollback(t), s) \equiv \\
&\qquad (\exists s').s = do(End(t), s') \wedge \neg \bigwedge_{IC \in \mathcal{IC}_v} IC(s) \vee \\
&\qquad (\exists t', s'').r\_dep(t, t', s) \wedge do(Rollback(t'), s'') \sqsubset s',
\end{aligned} \tag{55}$$

$$\begin{aligned}
&Poss(Join(t, t', n), s) \equiv t \neq t' \wedge \\
&\qquad (\exists s', t'', n')[do(Begin(t'), s') \sqsubset s \vee \\
&\qquad do(Join(t', t'', n'), s') \sqsubset s \wedge n' \neq CT].
\end{aligned} \tag{56}$$

Dependency axioms characterizing the fluents $r\_dep$ and $wc\_dep$ are:

$$\begin{aligned}
&r\_dep(t, t', do(a, s)) \equiv (\exists n).a = Join(t, t', n) \vee \\
&\quad transConflictCTH(t, t', do(a, s)) \vee r\_dep(t, t', s) \wedge a \neq Leave(t, t'),
\end{aligned} \tag{57}$$

$$\begin{aligned}
&wc\_dep(t, t', do(a, s)) \equiv \\
&\qquad a = Join(t', t) \vee wc\_dep(t, t', s) \wedge a \neq Leave(t, t').
\end{aligned} \tag{58}$$

The successor state axioms for CTHs are of the form:

$$\begin{aligned}
F(\mathbf{x}, &t, do(a, s)) \equiv \Phi_F(\mathbf{x}, a, t, s) \wedge \neg(\exists t'')a = Rollback(t'') \vee \\
&(\exists t')[a = Commit(t') \wedge parent(t, t', s) \wedge F(\mathbf{x}, t', s)] \vee \\
&(\exists t', n)[a = Join(t, t', n) \wedge F(\mathbf{x}, t', s)] \vee \\
&(\exists t'')a = Rollback(t'') \wedge \neg(\exists t')parent(t', t'', s) \wedge \\
&\qquad\qquad restoreBeginPoint(F, \mathbf{x}, t'', s) \vee \\
&(\exists t'')a = Rollback(t'') \wedge (\exists t')parent(t', t'', s) \wedge \\
&\qquad\qquad restoreJoinPoint(F, \mathbf{x}, t'', s),
\end{aligned} \tag{59}$$

one for each relation of the relational language, where $\Phi_F(\mathbf{x}, a, s)$ is a formula with free variables among $a, s, \mathbf{x}$; $restoreBeginPoint(F, \mathbf{x}, t, s)$ is defined in Abbreviation 7, and $restoreJoinPoint(F, \mathbf{x}, t, s)$ is the following:

**Abbreviation 17**

$$\begin{aligned}
&restoreJoinPoint(F, \mathbf{x}, t, s) =_{df} (\exists s', t', n).do(Join(t, t', n), s') \sqsubseteq s \wedge F(\mathbf{x}, s') \wedge \\
&\quad [(\exists a^*, s^*).do(Join(t, t', n), s') \sqsubset do(a^*, s^*) \sqsubseteq s \wedge writes(a^*, F, t)] \vee \\
&\quad [(\forall a^*, s^*).do(Join(t, t', n), s') \sqsubset do(a^*, s^*) \sqsubseteq s \supset \neg writes(a^*, F, t)] \wedge F(\mathbf{x}, s).
\end{aligned}$$

A basic relational theory for CTHs is as in Definition 3, but where the relational language includes $Join(t, t', n)$ as a further external action, and the axioms (52) – (56) replace axioms (10) – (13), the axioms (57) – (58) replace the axiom (14), and the successor state axioms in $\mathcal{D}_{ss}$ are of the form (59). All the other axioms of Definition 3 remain unchanged.

### *4.3 Open Nested Transactions*

Open nested transactions [38] are a generalized version of nested transactions. An open nested transaction is a system of component transactions forming an unbalanced tree whose nodes represent specific tasks that are performed by executing their children. All the other components with exception of the root of the system are called subtransactions. The leaves of such a tree are constituted by primitive actions. Given a node $t$ of an open nested transaction with $n$ children $t_1, \cdots, t_n$, these are classified into the following types:

- **Open Subtransactions**. These allow for unilateral commit and rollback independently of the parent transaction $t$. In other words, they are a collection of top-level transactions that may act independently from each other; but, if $t$ rolls back, they also must rollback. So they are allowed to make their updates visible to other subtransactions before committing.
- **Closed Subtransactions**. These are structured like in Moss nested transactions [28] (See Section 3). They do not allow for unilateral commit independently of the parent transaction $t$. For this reason, they are not allowed to make their updates visible to any other subtransactions before committing, at which point they only make their results available to their parent $t$. Any time, if $t$ rolls back, they also must rollback.
- **Compensatable Subtransactions**. These are associated with compensating subtransactions; that is, rather than simply vanishing when its parent $t$ rolls back, a compensatable $t_i$ that has already committed triggers a compensating transaction $comp_i$ whose semantics is described below. Since their actions can always be undone, they can in general be allowed to be open.
- **Compensating Subtransactions**. These undo any changes done by the committed compensatable subtransactions. So a compensating transaction $comp_i$ undoes any changes done by the corresponding compensatable subtransaction $t_i$. A compensating transaction can again be another open nested transaction. The order of compensating subtransactions must be compatible with the order of their corresponding compensatable transactions; that is, if $t_i$ commits before $t_j$ then whenever $comp_i$ begins, it does so only after $comp_j$ has committed.
- **Non-Compensatable Subtransactions**. These are subtransaction that cannot be compensated for whenever they have already committed. For this reason, they can in general not be allowed to be open.

We now give a situation calculus characterization of open nested transactions.

The external actions are: $Begin(t, t', m, c)$, $End(t)$, $Commit(t)$, and $Rollback(t)$. Among these, $Begin(t, t', mode, class)$ is new; intuitively, it means that the (sub)transaction $t$ begins as a component of the (sub)transaction $t'$ in mode $m$ and class $c$, where the mode argument can be one of $OPEN$, $CLOSED$ and $INV$, and the class argument can be one of $COMP$, $NONCOMP$ and $INV$. We introduce a predicate $compensates(t, comp_t)$, meaning that $comp_t$ compensates the actions of $t$.

A root transaction $t$ is a sequence $[a_1, \ldots, a_n]$ of primitive actions, where the action $a_1$ must be $Begin(t, NIL, INV, INV)$, and $a_n$ must be either $Commit(t)$, or $Rollback(t)$; $a_i, i = 2, \cdots, n-1$, may be any of the primitive actions, except $Begin(t, t', m, c)$, $Commit(t)$, and $Rollback(t)$. A subtransaction $t$ is a sequence $[a_1, \ldots, a_n]$ of primitive actions, where $a_1$ must be $Begin(t, t', m, c)$, and $a_n$ must be either $Commit(t)$, or $Rollback(t)$; $a_i, i = 2, \cdots, n-1$, may be any of the primitive actions, except $Begin(t^*, t^{**}, m^*, c^*)$, $Commit(t^*)$, and $Rollback(t^*)$.

The external actions of (sub)transactions are enumerated as follows.

**Abbreviation 18**

$$\begin{aligned} externalAct(a, t) =_{df} (\exists t', m, c)a = Begin(t, t', m, c) \lor a = End(t) \lor \\ a = Commit(t) \lor a = Rollback(t). \end{aligned}$$

We have to slightly reconsider the axiom for the fluent $parent(t, t', s)$, and the abbreviation $running(t, s)$:

$$\begin{aligned} & parent(t, t', do(a, s)) \equiv (\exists m, c)a = Begin(t, t', m, c) \lor \\ & \quad parent(t, t', s) \land a \neq Rollback(t) \land a \neq Rollback(t') \land \\ & \qquad a \neq Commit(t) \land a \neq Commit(t'); \end{aligned} \tag{60}$$

$$\begin{aligned} & running(t, s) =_{df} (\exists s', t', m, c)\{[s^* = do(Begin(t, t', m, c), s') \lor \\ & \qquad s^* = do(Begin(t', t, m, c), s')] \land \\ & \quad s^* \sqsubseteq s \land (\forall a, s'')[s^* \sqsubset do(a, s'') \sqsubset s \supset a \neq Rollback(t) \land a \neq End(t)]\}. \end{aligned} \tag{61}$$

We also add a predicate $compensatable(t)$ with the characterization

$$compensatable(t) =_{df} (\exists t')compensates(t', t).$$

Furthermore, we need the fluents $closed(t, s)$, and $comp(t, s)$, which intuitively tell whether a transaction is closed or compensatable, respectively. These fluents have the following successor state axioms:

$$\begin{aligned} & closed(t, do(a, s)) \equiv \\ & \quad (\exists t', m, c)a = Begin(t, t', m, c) \land t \neq NIL \land \\ & \qquad m = CLOSED \land c \neq INV \lor \\ & \quad closed(t, s) \land a \neq Commit(t) \land a \neq Rollback(t), \end{aligned} \tag{62}$$

$$\begin{aligned} & comp(t, do(a, s)) \equiv \\ & \quad (\exists t', m, c)a = Begin(t, t', m, c) \land t \neq NIL \land \\ & \qquad m \neq INV \land c = COMP \lor \\ & \quad closed(t, s) \land a \neq Commit(t) \land a \neq Rollback(t). \end{aligned} \tag{63}$$

The conflict predicate $transConflictNT(t, t', s)$ defined in (36) still captures the nature of conflict that may arise in the open nested transaction context.

In open nested transactions, the following dependencies must be maintained among transactions: a weak rollback dependency of a child on its parent, a commit dependency of a parent on all its children, a strong commit dependency of compensating transactions on their corresponding compensatable transactions, and dependency of the order of the beginnings of compensating transactions on the commitments of their corresponding compensatable transactions.

Now the external actions of an open nested transaction have the following precondition axioms:

$$\begin{aligned}
&Poss(Begin(t,t',m,c),s) \equiv \\
&\quad \neg(m = OPEN \wedge c = NONCOMP) \wedge t \neq t' \wedge \\
&\quad \{s = S_0 \vee \\
&\quad (\exists s',t^*,t'',m^*,c^*)[t^* \neq t \wedge do(Begin(t^*,t'',m^*,c^*),s') \sqsubset s] \wedge \\
&\quad [(\forall t'').bc\_dep(t,t'',s) \supset (\exists s'')do(Commit(t''),s'') \sqsubseteq s]\},
\end{aligned} \tag{64}$$

$$Poss(End(t),s) \equiv running(t,s), \tag{65}$$

$$\begin{aligned}
&Poss(Commit(t),s) \equiv (\exists s').s = do(End(t),s') \wedge \bigwedge_{IC \in \mathcal{IC}_v} IC(s) \wedge \\
&\quad (\forall t',s')[c\_dep(t,t',s) \supset (do(Commit(t'),s') \sqsubseteq s^* \supset \\
&\qquad do(Commit(t'),s') \sqsubset s)] \wedge \\
&\quad (\forall t',s').sc\_dep(t,t',s) \supset (\exists s'')do(Commit(t'),s'') \sqsubseteq s,
\end{aligned} \tag{66}$$

$$\begin{aligned}
&Poss(Rollback(t),s) \equiv (\exists s')[s = do(End(t),s') \wedge \neg \bigwedge_{IC \in \mathcal{IC}_v} IC(s)] \vee \\
&\quad (\exists t',s'')[r\_dep(t,t',s) \wedge do(Rollback(t'),s'') \sqsubseteq s] \vee \\
&\quad (\exists t',s')[wr\_dep(t,t',s) \wedge do(Rollback(t'),s') \sqsubseteq s \wedge \\
&\qquad (\forall s'')(do(Commit(t),s'') \not\sqsubseteq do(Rollback(t'),s'))],
\end{aligned} \tag{67}$$

Now we give dependency axioms characterizing the fluents $r\_dep(t,t',s)$, $wr\_dep(t,t',s)$, $sc\_dep(t,t',s)$, $c\_dep(t,t',s)$, and $bc\_dep(t,t',s)$:

$$r\_dep(t,t',s) \equiv transConflictNT(t,t',s), \tag{68}$$

$$\begin{aligned}
&wr\_dep(t,t',do(a,s)) \equiv \\
&\quad (\exists m,c)a = Begin(t,t',m,c) \wedge t' \neq NIL \wedge c = CLOSED \vee \\
&\quad wr\_dep(t,t',s) \wedge \neg termAct(a,t) \wedge \neg termAct(a,t'),
\end{aligned} \tag{69}$$

$$\begin{aligned}
&sc\_dep(t,t',do(a,s)) \equiv readsFrom(t,t') \vee \\
&\quad (\exists t'').parent(t'',t',s) \wedge compensates(t,t') \wedge a = Rollback(t'') \vee \\
&\quad sc\_dep(t,t',s) \wedge \neg termAct(a,t) \wedge \neg termAct(a,t'),
\end{aligned} \tag{70}$$

$$\begin{aligned}
&c\_dep(t,t',do(a,s)) \equiv \\
&\quad (\exists m,c)a = Begin(t',t,m,c) \wedge m = CLOSED \vee \\
&\quad transConflictNT(t,t',do(a,s)) \vee \\
&\quad c\_dep(t,t',s) \wedge \neg termAct(a,t) \wedge \neg termAct(a,t'),
\end{aligned} \tag{71}$$

$$\begin{aligned}
&bc\_dep(t,t',do(a,s)) \equiv (\exists s',s'',t^*,t^{**}).compensates(t,t^*) \wedge \\
&\quad compensates(t',t^{**}) \wedge do(Commit(t^*),s') \sqsubseteq s \wedge a = Commit(t^{**}) \vee \\
&\quad bc\_dep(t,t',s) \wedge a \neq Commit(t) \wedge a \neq Commit(t').
\end{aligned} \tag{72}$$

The successor state axioms for open nested transactions are of the form:

$$\begin{aligned}&F(\mathbf{x}, t, do(a, s)) \equiv \Phi_F(\mathbf{x}, a, t, s) \wedge \neg(\exists t'')a = Rollback(t'') \vee \\ &\quad (\exists t')[a = Commit(t') \wedge parent(t, t', s) \wedge closed(t, s) \wedge F(\mathbf{x}, t', s)] \vee \\ &\quad (\exists t'')a = Rollback(t'') \wedge restoreBeginPoint(F, \mathbf{x}, t'', s).\end{aligned} \tag{73}$$

one for each fluent $F$ of the relational language, where $\Phi_F(\mathbf{x}, a, t, s)$ is a formula with free variables among $a, t, s, \mathbf{x}$; $restoreBeginPoint(F, \mathbf{x}, t, s)$ is slightly different than in Abbreviation 7 and introduced now as follows:

**Abbreviation 19**

$$\begin{aligned}&restoreBeginPoint(F, \mathbf{x}, t, s) =_{df} \\ &\quad [(\exists a^*, s', s^*, t', t'', m, c).do(Begin(t, t'', m, c), s') \sqsubset do(a^*, s^*) \sqsubseteq s \wedge \\ &\qquad\qquad writes(a^*, F, t) \wedge F(\mathbf{x}, t', s')] \vee \\ &\quad [(\forall a^*, s^*, s', t'', m, c).do(Begin(t, t'', m, c), s') \sqsubset do(a^*, s^*) \sqsubseteq s \supset \\ &\qquad\qquad \neg writes(a^*, F, t)] \wedge (\exists t')F(\mathbf{x}, t', s).\end{aligned}$$

A basic relational theory for open nested transactions is as in Definition 3, but where the relational language includes $Begin(t, t', m, c)$ as a further action, and the axioms (64) – (67) replace axioms (10) – (13), the axioms (68) – (72) replace the axiom (14), and the successor state axioms in $\mathcal{D}_{ss}$ are of the form (73). All the other axioms of Definition 3 remain unchanged.

It is important to note that open nested transactions have one more system action which is $Begin(t, t', m, c)$ in the special case where $t'$ compensates some other transaction. Thus our definition for $systemAct(a, t)$ must capture this novelty:

**Abbreviation 20**

$$\begin{aligned}&systemAct(a, t) =_{df} a = Commit(t) \vee a = Rollback(t) \vee \\ &\quad (\exists s, t', t'')bc\_dep(t, t', s) \wedge parent(t'', t', s) \supset a = Begin(t'', t, OPEN, INV).\end{aligned}$$

## 5 Example

We consider a Debit/Credit example which illustrates how to formulate a relational theory for closed nested transactions.

The database involves a relational language with:

**Fluents**: $served(aid, s)$, $branches(bid, bbal, bname, t, s)$, $tellers(tid, tbal, t, s)$, $accounts(aid, bid, abal, t, s)$.
**Situation Independent Predicate**: $requested(aid, req)$.
**Action Functions**: $b\_insert(bid, bbal, bname, t)$, $b\_delete(bid, bbal, bname, t)$, $t\_insert(tid, tbal, t)$,
$t\_delete(tid, tbal, t)$, $a\_insert(aid, bid, abal, tid, t)$, $a\_delete(aid, bid, abal, tid, t)$, $report(aid)$.
**Constants**: $Ray$, $Iluju$, $Misha$, $Ho$, etc.

The meaning of the arguments of fluents are self explanatory; and the relational language also includes the external actions of nested transactions. Among the fluents above, $served(aid, s)$ is a system fluent, and the remaining ones are database fluents.

To be brief, we skip unique name axioms and concentrate ourself on the remaining axioms of the basic relational theory. We enforce the following ICs ($\mathcal{IC}_e$):

$$\begin{array}{l}
accounts(aid, bid, abal, tid, t, s) \wedge accounts(aid, bid', abal', tid', t', s) \supset \\
\qquad\qquad\qquad bid = bid', abal = abal', tid = tid', \\
branches(bid, bbal, bname, t, s) \wedge branches(bid, bbal', bname', t', s) \supset \\
\qquad\qquad\qquad bbal = bbal', bname = bname', \\
tellers(tid, tbal, t, s) \wedge tellers(tid, tbal', t', s) \supset tbal = tbal';
\end{array}$$

and we have to verify the IC ($\mathcal{IC}_v$)

$$accounts(aid, bid, abal, tid, t, s) \supset abal \geq 0$$

at transaction's end. The following are the update precondition axioms:

$$\begin{array}{l}
Poss(a\_insert(aid, bid, abal, tid, t), s) \equiv \\
\quad \neg(\exists t')accounts(aid, bid, abal, tid, t', s) \wedge \\
\quad IC^e(do(a\_insert(aid, bid, abal, tid, t), s)) \wedge running(t, s), \\
Poss(a\_delete(aid, bid, abal, tid, t), s) \equiv \\
\quad (\exists t')accounts(aid, bid, abal, tid, t', s) \wedge \\
\quad IC^e(do(a\_delete(aid, bid, abal, tid, t), s)) \wedge running(t, s), \\
Poss(b\_insert(bid, bbal, bname, t), s) \equiv \\
\quad \neg(\exists t')branches(bid, bbal, bname, t', s) \wedge \\
\quad IC^e(do(b\_insert(bid, bbal, bname, t), s)) \wedge running(t, s), \\
Poss(b\_delete(bid, bbal, bname, t), s) \equiv \\
\quad (\exists t')branches(bid, bbal, bname, t', s) \wedge \\
\quad IC^e(do(b\_delete(bid, bbal, bname, t), s)) \wedge running(t, s), \\
Poss(t\_insert(tid, tbal, t), s) \equiv \neg(\exists t')tellers(tid, tbal, t', s) \wedge \\
\quad IC^e(do(t\_insert(tid, tbal, t), s)) \wedge running(t, s), \\
Poss(t\_delete(tid, tbal, t), s) \equiv (\exists t')tellers(tid, tbal, t', s) \wedge \\
\quad IC^e(do(t\_delete(tid, tbal, t), s)) \wedge running(t, s),
\end{array}$$

The successor state axioms ($\mathcal{D}_{ss}$) are:

$$
\begin{array}{l}
accounts(aid, bid, abal, tid, t, do(a, s)) \equiv \\
\quad ((\exists t_1)a = a\_insert(aid, bid, abal, tid, t_1) \vee \\
\qquad (\exists t_2)accounts(aid, bid, abal, tid, t_2, s) \wedge \\
\qquad\quad \neg(\exists t_3)a = a\_delete(aid, bid, abal, tid, t_3)) \wedge \neg(\exists t')a = Rollback(t') \vee \\
\quad (\exists t').a = Rollback(t') \wedge \neg(\exists t'')parent(t'', t', s) \wedge \\
\qquad\qquad restoreBeginPoint(accounts, (aid, bid, abal, tid), t', s) \vee \\
\quad a = Rollback(t') \wedge (\exists t'')parent(t'', t', s) \wedge \\
\qquad\qquad restoreSpawnPoint(accounts(aid, bid, abal, tid), t', s),
\end{array}
$$

$$
\begin{array}{l}
branches(bid, bbal, bname, t, do(a, s)) \equiv \\
\quad ((\exists t_1)a = b\_insert(bid, bbal, bname, t_1) \vee \\
\qquad (\exists t_2)branches(bid, bbal, bname, t_2, s) \wedge \\
\qquad\quad \neg(\exists t_3)a = b\_delete(bid, bbal, bname, t_3)) \wedge \neg(\exists t')a = Rollback(t') \vee \\
\quad (\exists t').a = Rollback(t') \wedge \neg(\exists t'')parent(t'', t', s) \wedge \\
\qquad\qquad restoreBeginPoint(branches, (bid, bbal, bname), t', s) \vee \\
\quad a = Rollback(t') \wedge (\exists t'')parent(t'', t', s) \wedge \\
\qquad\qquad restoreSpawnPoint(branches, (bid, bbal, bname), t', s),
\end{array}
$$

$$
\begin{array}{l}
tellers(tid, tbal, do(a, s)) \equiv \\
\quad ((\exists t_1)a = t\_insert(tid, tbal, t_1) \vee \\
\qquad (\exists t_2)tellers(tid, tbal, t_2, s) \wedge \\
\qquad\quad \neg(\exists t_3)a = t\_delete(tid, tbal, t_3)) \wedge \neg(\exists t')a = Rollback(t') \vee \\
\quad (\exists t').a = Rollback(t') \wedge \neg(\exists t'')parent(t'', t', s) \wedge \\
\qquad\qquad restoreBeginPoint(tellers, (tid, tbal), t', s) \vee \\
\quad a = Rollback(t') \wedge (\exists t'')parent(t'', t', s) \wedge \\
\qquad\qquad restoreSpawnPoint(tellers, (tid, tbal), t', s).
\end{array}
$$

## 6 Related Work

ACTA [9] is a framework similar to ours. It allows to specify effects of transactions on objects and on other transactions. In fact, we use the same building blocks for ATMs as those used in ACTA. However, the reasoning capability of the situation calculus exceeds that of ACTA for the following reasons: (1) the database log is a first class citizen of the situation calculus, and the semantics of all transaction operations – $Commit$, $Rollback$, etc. – are defined with respect to constraints on this log. Nowhere have we seen a quantification over histories in ACTA, so that there is no straightforward way of expressing closed form formulas involving histories in ACTA. (2) Our approach goes far beyond ACTA as it is an implementable specification, thus allowing one to automatically check many properties of the specification using an interpreter. To that end, the main implementation theorems needed are formulated in [34]. Finally, (3) although ACTA deals with the dynamics of database objects, it is never explicitly formulated as a logic for actions.

In [5], Bertossi *et al.* propose a situation calculus-based formalization of database transactions. They extend Reiter's specification of database updates to transactions. Our approach, however, is based on a situation calculus that is explicitly non-Markovian. Moreover, our work goes beyond pure flat transactions to deal with ATMs which are more complex.

Bonner and Kiefer present a *transaction logic* ($\mathcal{TR}$) that includes a model theory, and a sound and complete SLD-like proof theory [6,7]. The execution of a transaction $\psi$ is described by an executional entailment which intuitively means that, given transaction axioms gathered in **P**, the execution of the transaction $\psi$ leads the initial database $\mathbf{D}_0$ to the final database $\mathbf{D}_n$ through a sequence of intermediate states $\mathbf{D}_1, \cdots, \mathbf{D}_{n-1}$.

Like our situation calculus based transaction language, $\mathcal{TR}$ allows the formulation of complex transactions and bulk updates; and its notion of executional entailment corresponds to the logical entailment in the situation calculus. However, $\mathcal{TR}$ differs from our language. First, $\mathcal{TR}$ is both update- and sentence-centered; it allows not only elementary updates, but also additions and removals of rules. By contrast, our approach is solely update-centered.[5] By restricting our concern to updates, we avoid invoking a revision theory. Second, unlike $\mathcal{TR}$, elementary updates in the situation calculus are not predicates, but first order terms instead. Third, unlike in $\mathcal{TR}$ that deals with updates at the physical level, the situation calculus deals with updates at the virtual level. In fact, this limitation can be overcome in the situation calculus by progressing the initial database after each elementary update execution [34]. Fourth, $\mathcal{TR}$ does not deal with ATMs. Finally, ours is a classical predicate calculus logic.

To do full justice to $\mathcal{TR}$ , one should distinguish between the full logic and its Horn fragment. Many of the limitations mentioned above concern the Horn fragment, not the full logic. For example, Santos [36] shows how the full logic is used for reasoning about arbitrary elementary and complex actions (not just tuple insertions and deletions).

*Statelog* [22] is a logic of database state change that includes a model theory. Like ours, this logic allows the formulation of complex transactions, is update-centered, and considers invariance in state changes. However, Statelog differs from our approach in the following way. First, while we allow only primitive updates that are first-order terms, Statelog expresses actions corresponding to our $Begin(t)$, $End(t)$, $Commit(t)$, $Rollback_{sys}(t)$, etc, as relations. In this respect, Statelog is similar to $\mathcal{TR}$. Second, unlike Statelog, we appeal solely to the classical semantics of predicate logic; we have no need of a special-purpose semantics to account for models of database transactions. Third, unlike Statelog, we do not deal with updates at the physical level.

[5] Update-centered approaches specify explicit update operations in the update language; and sentence-centered approaches allow for updates with arbitrary sentences [32].

An early work by Lynch *et al.* reported in [23] and [24] describes an automaton-based theoretical framework for reasoning about atomic transactions. This approach is in spirit close to our general philosophy of providing a framework for reasoning about transactions. It shares with our situation calculus based model a common ground. First, both approaches view the execution of a transactional behavior as a sequence of actions. Second, both include the idea of building a complex transactional behavior from existing, simpler ones by using well defined combination constructs. However, our approach models a transaction in a very different way: while we model these as logical theories, the approach by Lynch *et al.* models them as automata. We believe that automata used in this approach do not offer the same flexibility that the situation calculus logic does. The situation calculus does in fact have connections with automata in a different way than the one developed by Lynch *et al.*: the decision problem for fragments of the language can be related to automata on infinite trees as done in [37]. Here, automata are used to provide a semantics for (fragments of) the modeling language (i.e. the situation calculus) and not as the modeling language itself.

## 7 Conclusion

### *7.1 Summary*

We have noticed that ATMs found in the literature are proposed in an *ad hoc* way for dealing with applications involving long-lived, endless, and cooperative activities. Therefore, it is not obvious to compare ATMs to each other. Also it is difficult to exactly say how an ATM extends the traditional flat model of transaction, and to formulate its properties in a way that one clearly differentiates functionalities that have been added or subtracted. To address these questions, we have introduced a general and common framework within which to specify ATMs, specify their properties, and reason about these properties. Thus far, ACTA [9,8] seems to our knowledge the only framework addressing these questions at a high level of generality. In ACTA, a first order logic-like language is used to capture the semantics of any ATM.

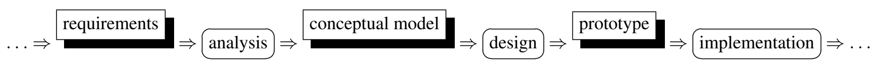
We have given logical foundations for ATMs by capturing the latter as non-Markovian action theories formulated in the situation calculus. The main contributions of this paper with respect to the specification of ATMs can be summarized as follows:

- We constructed logical theories called *basic relational theories* to formalize ATMs along the tradition set by the ACTA framework; basic relational theories are non-Markovian theories in which one may explicitly refer to all past states, and not only to the previous one. They provide the formal semantics of the corresponding ATMs. They are an extension of the classical relational theories of [31] to the database transaction setting.
- We extended the notion of *legal database logs* introduced in [32] to accommodate transactional actions such as $Begin$, $Commit$, etc. These logs are first class citizen of the logic, and properties of the ATM are expressed as formulas

of the situation calculus that logically follow from the basic relational theory representing that ATM. It turns out that ATM properties basically are properties of legal logs.

– We have accounted for several variations of the classical transactions with ACID properties, as well as for several sample ATMs. The resulting theories of ATMs are modular and mostly incremental with respect to theories that capture the classical models. This means that, to capture ATMS, the changes made to the theories capturing classical models usually consists in adding new axioms and minimally modifying existing ones.

**Fig. 1** Relational theories as conceptual models of ATMs



## *7.2 Future Work*

Ideas expressed and developed in this paper may be extended in various ways. we mention a few of them.

– **Running the specifications**. We have used one single logic – the situation calculus — to accounts for all features of ATMs. By combining our theories with the simulation method developed in [18], we also obtain an account of execution models for ATMs. The output of this account is a conceptual model for ATMs in the form of relational theories. Thus, considering the software development cycle as depicted in Figure 1, a relational theory corresponding to an ATM constitutes a conceptual model for that ATM. Since active relational theories are implementable specifications [18], implementing the conceptual model provides one with a rapid prototype of the specified ATM.

– **Progressing Databases**. One must distinguish between our approach which is a purely logical, abstract specification in which all system properties are formulated relative to the database log, and an implementation which normally materializes the database using *progression* ([34]). This is the distinguishing feature of our approach. The database log is a first class citizen of the logic, and the semantics of all transaction operations – $Commit$, $Rollback$, etc. –, primitive updates, and queries are defined with respect to this log. The main mechanism used in this respect is *regression*. However, in order to materialize the database after each update, progressing the database is the way to go. How progression can be defined for basic and active relational theories is completely open.

– **Comparing to other Approaches**. Database transaction processing is now a mature area of research and practice. However, one needs to formally know how our formalization indeed captures any existing theory, such as ACTA, at the same level of generality. Doing so, one proves some form of correctness of our formalization, assuming that ACTA is correct. For example, we need an effective translation of our basic relational theories into ACTA axioms for a relational database and then show that the legal logs for the situation calculus basic relational theory are precisely the correct histories for its translation into a relational ACTA system.

- **Non-relational Data Models**. Thus far, we have given axioms that accommodate a complete initial database state. This, however, is not a requirement of the theory we are presenting. Therefore our account could, for example, accommodate initial databases with null values, open initial database states, initial databases accounting for object orientation, or initial semistructured databases. These are just examples of some of the generalizations that our initial databases could admit.
- **Second Order Features**. It is important to notice that the only place where the second order nature of our framework is needed is in the proof of the properties of the transaction models that rely on a second order induction principle contained in the foundational axioms of the situation calculus (See the Appendix). For the Markovian action theories of the situation calculus, it is shown in [30] that the second order nature of this language is not at all needed in simulating basic action theories. It remains to show that this is also the case for the non-Markovian setting.
- **Accounting for Further ATMs**. We would like to accounting for some of the recent ATMs, for example those reported in [16] and open nested transactions proposed in the context of mobile computing, and reason about the specifications obtained.
- **Systematic Implementation of some of the Semantics**. Being foundational, this work has been theoretical by its nature. We need to show how to implement the theories of this paper, and indeed implement some short programs as an illustration.

## References

1. S. Abiteboul. Updates, a new frontier. In *Proceedings of the Second International conference on Database Theory*, pages 1–18, 1988.
2. S. Abiteboul and V. Vianu. Procedural and declarative database update languages. In *Proceedings of the 7th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 240–250, 1988.
3. S. Abiteboul and V. Vianu. Procedural languages for database queries and updates. *Journal of Computer System Sciences*, 41(2):181–229, 1990.
4. P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*. Addison-Wesley, Reading, MA, 1987.
5. L. Bertossi, J. Pinto, and R. Valdivia. Specifying database transactions and active rules in the situation calculus. In H. Levesque and F. Pirri, editors, *Logical Foundations of Cognitive Agents. Contributions in Honor of Ray Reiter*, pages 41–56, New-York, 1999. Springer Verlag.

6. A. Bonner and M. Kifer. Transaction logic programming. Technical report, University of Toronto, 1992.
7. Anthony J. Bonner and Michael Kifer. A logic for programming database transactions. In *Logics for Databases and Information Systems*, pages 117–166, 1998.
8. P. Chrysanthis and K. Ramamritham. Synthesis of extended transaction models. *ACM Transactions on Database Systems*, 19(3):450–491, 1994.
9. P.K. Chrysanthis. *ACTA, A Framework for Modeling and Reasoning about Extended Transactions*. PhD thesis, Department of Computer and Information Science, University of Massachusetts, Amherst, 1991.
10. A. K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, Sand Mateo, CA, 1992. Selections.
11. R. Fagin, J. Ullman, and M.Y. Vardi. Updating logical databases. In *Proceedings of the second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1983.
12. A. Gabaldon. Non-markovian control in the situation calculus. In *Proceedings of AAAI*, Edmonton, Canada, 2002.
13. G. Grahne. *The Problem of Incomplete Information in Relational Databases*. Springer Verlag, Berlin, 1991.
14. J. Gray and Reuter A. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Mateo, CA, 1995.
15. A. Guessoum and J.W. Lloyd. Updating knowledge bases. *New Generation Computing*, 8(1):71–89, 1990.
16. S. Jajodia and L. Kerschberg. *Advanced Transaction Models and Architectures*. Kluwer Academic Publishers, Boston, 1997.
17. H. Katsuno and A.O. Mendelzon. On the difference between updating a knowledge base and revising it. In *Proceedings of the Second International Conference on Principles of Knowledge Representation an Reasoning*, pages 387–394, Los Altos, CA, 1991. Morgan Kaufmann Publishers.
18. I Kiringa. Simulation of advanced transaction models using golog. In *Proceedings of the 8th Biennial Workshop on Data Bases and Programming Languages (DBPL'01)*, 2001.
19. I. Kiringa and A. Gabaldon. Expressing transactions with savepoints as non-markovian theories of actions. In *KRDB*, 2003.
20. F. Lin and R. Reiter. State constraints revisited. *J. of Logic and Computation*, 4(5):655–678, 1994.
21. B. Ludäscher, U. Hamann, and G. Lausen. A logical framework for active rules. In *Proceedings of the Seventh International Conference on Management of Data*, Pune, 1995. Tata and McGraw-Hill.
22. B. Ludäscher, W. May, and G. Lausen. Nested transactions in a logical language for active rules. Technical Report Jun20-1, Technical Univ. of Munich, June 1996.
23. N. Lynch, M.M. Merritt, W. Weihl, and A. Fekete. A theory of atomic transactions. In M. Gyssens, J. Parendaens, and D. Van Gucht, editors, *Proceedings of the Second International Conference on Database Theory*, pages 41–71, Berlin, 1988. Springer Verlag. LNCS 326.
24. N. Lynch, M.M. Merritt, W. Weihl, and A. Fekete. *Atomic Transactions*. Morgan Kaufmann, San Mateo, 1994.
25. B.E. Martin and C Pedersen. Long-lived concurrent activities. Technical report, HP Laboratories, 1990. HPL-90-178.
26. J. McCarthy. Situations, actions and causal laws. Technical report, Stanford University, 1963.

27. J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.
28. J. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing. Information Systems Series*. The MIT Press, Cambridge, MA, 1985.
29. M.H. Nodine, S. Ramaswamy, and Zdonik. A cooperative transaction model for design databases. In A.K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 53–85, San Mateo, CA, 1992. Morgan Kaufmann.
30. F. Pirri and R. Reiter. Some contributions to the metatheory of the situation calculus. *Journal of the ACM*, 46(3):325–364, 1999.
31. R. Reiter. Towards a logical reconstruction of relational database theory. In M. Brodie, J. Mylopoulos, and J. Schmidt, editors, *On Conceptual Modelling*, pages 163–189, New-York, 1984. Springer Verlag.
32. R. Reiter. On specifying database updates. *J. of Logic Programming*, 25:25–91, 1995.
33. R. Reiter. Natural actions, concurrency and continuous time in the situation calculus. In L.C. Aiello, J. Doyle, and S.C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96)*, pages 2–13, San Francisco, CA, 1996. Morgan Kaufmann.
34. R. Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. MIT Press, Cambridge, 2001.
35. Abiteboul S. and V. Vianu. A transaction language complete for database update and specification. In *Proceedings of the sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 260–268, San Diego, CA, 1987.
36. M.V. Santos. Specifying and reasoning about actions in open-worlds using transaction logic. In *Proceedings of the ECAI 2000 Workshop on Cognitive Robotics*, Berlin, Germany, 2000.
37. E. Ternovskaia. Automata theory for reasoning about automata. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 153–158, 1999.
38. G. Weikum and H.J. Schek. Concepts and applications of multilevel transactions and open nested transactions. In A.K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 516–553, San Mateo, CA, 1992. Morgan Kaufmann.
39. M. Winslett. *Updating Logical Databases*. Cambridge University Press, Cambridge, MA, 1990.

## A Proofs

In proving the different (relaxed) ACID properties, the following three lemmas exhibiting simple properties of legal logs will be useful.

**Lemma 3** *Let $\mathcal{D}$ be a basic relational theory. Then*

$$\begin{aligned}\mathcal{D}_f \cup \{(9)\} \models (\forall s, a)\{&legal(S_0)\wedge\\ &[legal(do(a, s)) \equiv legal(s) \wedge Poss(a, s)\wedge\\ &(\forall a', t).systemAct(a', t) \wedge responsible(t, a', s) \wedge Poss(a', s) \supset a = a']\}.\end{aligned}$$

**Proof**: We must prove two goals:

$$\mathcal{D}_f \cup \{(9)\} \models legal(S_0), \tag{74}$$

and

$$\begin{aligned}\mathcal{D}_f \cup \{(9)\} \models(\forall s, a)[&legal(do(a, s)) \equiv legal(s) \wedge Poss(a, s)\wedge\\ &(\forall a', t)[systemAct(a', t) \wedge responsible(t, a', s)\wedge\\ &Poss(a', s) \supset a = a']].\end{aligned} \tag{75}$$

**Goal (74)**: By Abbreviation (9), we must pursue two subgoals:

$$\mathcal{D}_f \cup \{(9)\} \models (\forall a, s^*)[do(a, s^*) \sqsubseteq S_0 \supset Poss(a, s^*)]$$

and

$$\begin{aligned}\mathcal{D}_f \cup \{(9)\} \models (\forall a', a'', s', t)[&systemAct(a', t) \wedge responsible(t, a', s')\wedge\\ &responsible(t, a'', s') \wedge Poss(a', s') \wedge do(a'', s') \sqsubset S_0 \supset a' = a''].\end{aligned}$$

For the first subgoal, we are lead ultimately to a success in the proof of its consequent by using the foundational axiom (3) and the sentence $S_0 \neq do(a, s)$, a consequence of $\mathcal{D}_f$; and the second subgoal is obviously true by virtue of the fact that $S_0 \neq do(a, s)$.

**Goal (75)**: The proof is by induction on $s$, using the induction axiom (2). The case $s = S_0$ is intuitive enough:

$$\begin{aligned}\mathcal{D}_f \cup \{(9)\} \models (\forall a)[&legal(do(a, S_0)) \equiv legal(S_0) \wedge Poss(a, S_0)\wedge\\ &(\forall a', t)[systemAct(a', t) \wedge responsible(t, a', S_0)\wedge\\ &Poss(a', S_0) \supset a = a']];\end{aligned}$$

its proof, though tedious, is straightforward. Now, assume the formula in (75) proven for $do(a, s)$. We must prove it for $do(a^*, do(a, s))$; i.e. we must prove

$$\begin{aligned}\mathcal{D}_f \cup \{(9)\} \models (\forall s, a, a^*)[&legal(do(a^*, do(a, s))) \equiv legal(do(a, s))\wedge\\ &Poss(a^*, do(a, s)) \wedge (\forall a', t)[systemAct(a', t)\wedge\\ &responsible(t, a', do(a, s)) \wedge Poss(a', do(a, s)) \supset a^* = a']].\end{aligned} \tag{76}$$

$\Longleftarrow$ :
Assume for fixed $a$, $a^*$, and $s$ $legal(do(a, s))$, $Poss(a^*, do(a, s)))$, and

$$\begin{aligned}(\forall a', t).systemAct(a', t) \wedge responsible(t, a', do(a, s)) \wedge\\ Poss(a', do(a, s)) \supset a^* = a'.\end{aligned} \quad \dagger$$

By induction hypothesis, $legal(do(a,s))$ can be replaced by $legal(s)$, $Poss(a,s)$, and

$$\begin{aligned}&(\forall a',t).systemAct(a',t) \wedge responsible(t,a',s) \wedge\\ &\qquad\qquad Poss(a',s) \supset a = a'. \qquad\qquad \ddagger\end{aligned}$$

From $legal(s)$, $Poss(a,s)$, and $Poss(a^*, do(a,s))$ we obtain

$$(\forall s^{**}).do(a,s^{**}) \sqsubseteq do(a^*, do(a,s))) \supset Poss(a,s^{**}). \qquad (\$)$$

From $legal(s)$, (†), and (‡) we have

$$\begin{aligned}&(\forall a_1,a_2,s',t)[systemAct(a_1,t) \wedge responsible(t,a_1,do(a,s)) \wedge Poss(a_1,s') \wedge\\ &\qquad do(a_2,s') \sqsubset do(a^*, do(a,s)) \supset a_1 = a_2]. \qquad (\$\$)\end{aligned}$$

Now, from ($) and ($$) follows $legal(do(a^*, do(a,s)))$.

$\Longrightarrow$ :
Assume for fixed $a$, $a^*$, and $s$ $legal(do(a, do(a^*,s)))$. Then, by Definition (9),

$$\begin{aligned}&(\forall a_1,s_1)[do(a_1,s_1) \sqsubseteq do(a^*, do(a,s)) \supset Poss(a_1,s_1)] \wedge\\ &\quad (\forall a_2,a_3,s_2,t)[systemAct(a_2,t) \wedge responsible(t,a_2,s_2) \wedge Poss(a_2,s_2) \wedge \qquad (77)\\ &\qquad do(a_3,s_2) \sqsubset do(a^*, do(a,s)) \supset a_2 = a_3].\end{aligned}$$

Thus we must show that (77) implies $legal(do(a,s))$, $Poss(a^*, do(a,s))$, and

$$\begin{aligned}&(\forall a',t).systemAct(a',t) \wedge responsible(t,a',do(a,s)) \wedge\\ &\qquad Poss(a', do(a,s)) \supset a^* = a'.\end{aligned} \qquad (78)$$

Subgoal $Poss(a^*, do(a,s))$ follows from the first conjunct of (77) alone; subgoal (78) follows from the second conjunct of (77), and subgoal $legal(do(a,s))$ follows from both conjuncts of (77). Foundational axioms and the induction hypothesis are involved in this proof whose details are omitted here. □

**Lemma 4** *Suppose $\mathcal{D}$ is a basic relational theory. Then*

$$\mathcal{D} \cup \{(9)\} \models legal(s) \supset (\forall s')[s' \sqsubseteq s \supset legal(s')].$$

**Proof**:

We conduct an induction on $s$.

For the case $s = S_0$, we must prove $legal(S_0) \supset (\forall s')[s' \sqsubseteq S_0 \supset legal(s')]$. By Lemma 3, $legal(S_0)$. Therefore, $(\forall s')[s' \sqsubseteq S_0 \supset legal(s')]$ which is clearly true by the foundational axiom (3).

Now assume the result for $s$ and suppose for fixed $a$ and $s$ that $legal(do(a,s))$ holds. Then we must prove $(\forall s')[s' \sqsubseteq do(a,s) \supset legal(s')]$. Since $legal(do(a,s))$, then, by Lemma 3, $Poss(a,s)$ and $legal(s)$. Assume, for fixed $s'$, $s' \sqsubseteq do(a,s)$. Henceforth we must show that $legal(s')$. For the assumption $s' = do(a,s)$, since $legal(do(a,s))$, we have immediately $legal(s')$. For the assumption $s' \sqsubset do(a,s)$, by the foundational axiom (4), we get $s' \sqsubset s$; and since $legal(s)$, we obtain, by induction hypothesis, $(\forall s^*)[s^* \sqsubseteq s \supset legal(s^*)]$. Henceforth $legal(s')$. □

**Lemma 5** *Suppose $a_1, \cdots, a_n$ is a sequence of ground action terms. Then*

$$\begin{aligned}&\mathcal{D}_f \cup \{(9)\} \models \\ &\qquad legal(do([a_1, \cdots, a_n], s)) \equiv \\ &\bigwedge_{i=1}^{n} \{Poss(a_i, do([a_1, \cdots, a_{i-1}], s)) \wedge \\ &\qquad (\forall a, t)[systemAct(a, t) \wedge responsible(t, a, do([a_1, \cdots, a_{i-1}], s)) \wedge \\ &\qquad\qquad Poss(a, do([a_1, \cdots, a_{i-1}], s)) \supset a_i = a]\}.\end{aligned}$$

**Proof**: by induction on the length of the sequence of actions.
For the case $n = 1$, we must prove

$$\begin{aligned}\mathcal{D}_f \models & legal(do(a, s)) \equiv Poss(a, s) \wedge (\forall a', t)[systemAct(a', t) \wedge \\ & responsible(t, a', s) \wedge Poss(a', s) \supset a = a'].\end{aligned}$$

The proof is immediate by Lemma 3.

Assume the result for $n$. We must prove that $\mathcal{D}_f$ and (9) entail

$$\begin{aligned}& legal(do([a_1, \cdots, a_{n+1}], s)) \equiv \\ &\quad \bigwedge_{i=1}^{n+1} \{Poss(a, do([a_1, \cdots, a_{i-1}], s)) \wedge \\ &\quad (\forall a', t)[systemAct(a', t) \wedge responsible(t, a', do([a_1, \cdots, a_{i-1}], s)) \wedge \\ &\qquad Poss(a', do([a_1, \cdots, a_{i-1}], s)) \supset a_i = a']\}.\end{aligned}$$

$\Longrightarrow$ :
Assume for fixed $s$ $legal(do([a_1, \cdots, a_{n+1}], s))$. By Lemma 4 and the fact, provable by induction, that $s \sqsubset do(a, s)$, we have $legal(do([a_1, \cdots, a_n], s))$. Henceforth, by induction hypothesis,

$$\begin{aligned}&\bigwedge_{i=1}^{n} \{Poss(a, do([a_1, \cdots, a_{i-1}], s)) \wedge \\ &\quad (\forall a', t)[systemAct(a', t) \wedge responsible(t, a', do([a_1, \cdots, a_{i-1}], s)) \wedge \\ &\qquad Poss(a', do([a_1, \cdots, a_{i-1}], s)) \supset a_i = a']\}.\end{aligned}$$

Thus, we must now only show that $Poss(a_{n+1}, do([a_1, \cdots, a_n], s))$ and

$$\begin{aligned}&(\forall a', t)[systemAct(a', t) \wedge responsible(t, a', do([a_1, \cdots, a_n], s)) \wedge \\ &\qquad Poss(a', do([a_1, \cdots, a_n], s)) \supset a_{n+1} = a'].\end{aligned}$$

Both of these claims follow from Lemma 3.

$\Longleftarrow$ :
This part of the proof is symmetric to the previous case.
This completes the proof of the inductive case. □

**Theorem** 1

**1.** Assume, for fixed $s$, $legal(s)$ and let $s = do([B_1, \cdots, B_m], S_0)$. Then, by Lemma 5,

$$\bigwedge_{i=1}^{m} Poss(a_i, do([B_1, \cdots, B_{i-1}], S_0)). \qquad (79)$$

Now assume, for fixed $a, s', s''$, and $t$, $do(a, s') \sqsubset s$, $do(a, s'') \sqsubset s$, and $externalAct(a, t)$. Therefore we must prove $s' = s''$. From the assumption $externalAct(a, t)$ and Abbreviation 10, we must consider four cases.

**Case** $a = Begin(t)$. Assume, contrary to our goal, that $s' \neq s''$. Henceforth, either $s' \sqsubset s''$ or $s'' \sqsubset s'$. Suppose $s' \sqsubset s''$. Then, by the foundational axioms (1), (4), and the assumptions $do(Begin(t), s') \sqsubset s$ and $do(Begin(t), s'') \sqsubset s$, we have $s' \sqsubset do(Begin(t), s') \sqsubset do(Begin(t), s'')$. By (79), we have $Poss(Begin(t), s'')$; henceforth, by the precondition axiom (10) for the action $Begin(t)$, we have $\neg(\exists s^*)do(Begin(t), s^*) \sqsubseteq s''$. Now, this contradicts the fact that $do(Begin(t), s') \sqsubset do(Begin(t), s'')$. The subcase $s' \sqsubset s''$ is proven in an analog way.

**Case** $a = End(t)$. Assume that $s' \neq s''$. Henceforth, either $s' \sqsubset s''$ or $s'' \sqsubset s'$. Suppose $s' \sqsubset s''$. Then, similarly to the previous case, we get $s' \sqsubset do(End(t), s') \sqsubset do(End(t), s'')$. By (79), we have $Poss(End(t), s'')$; henceforth, by the precondition axiom (11) for the action $End(t)$, we have $running(t, s'')$, and, by Abbreviation 6, we have

$$(\exists s^*).do(Begin(t), s^*) \sqsubseteq s'' \wedge \\ (\forall a, s^{**})[do(Begin(t), s^*) \sqsubset do(a, s^{**}) \sqsubset s'' \supset a \neq Rollback(t) \wedge a \neq End(t)]. \qquad (80)$$

Since by the previous case, the log $do(Begin(t), s^*)$ that exists must be the same for both $do(End(t), s')$ and $do(End(t), s'')$, we clearly get a contradiction between the fact that $s' \sqsubset do(End(t), s') \sqsubset do(End(t), s'')$ and (80). The subcase $s' \sqsubset s''$ is proven in an analog way.

**Cases** $a = Commit(t)$ and $a = Rollback(t)$ . Both cases follow immediately from the case $a = End(t)$, as both $Commit(t)$ and $Rollback$ are possible only in logs following the execution of $End(t)$. □

**Theorem** 2

Assume, for fixed $s$, $legal(s)$. Suppose, for fixed $t$,$t'$, and $s'$, that $sc\_dep(t, t')$, and that $do(Commit(t'), s') \sqsubset s$. Then we must show that $(\exists s^*)do(Commit(t), s^*) \sqsubseteq s$. Since $legal(s)$ and $do(Commit(t'), s') \sqsubset s$, we have, by Lemma 4, $legal(do(Commit(t'), s'))$ and, by Lemma 3, $Poss(Commit(t'), s')$. By the action precondition axiom for $Commit(t')$, we have

$$(\forall t^*)[sc\_dep(t, t^*, s) \supset (\exists s'')do(Commit(t^*), s'') \sqsubseteq s].$$

Since $sc\_dep(t, t')$, this leads easily to what we had to prove.

The second conjunct involving $r\_dep(t, t', s)$ and $Rollback(t)$ is proven in a similar way and we omit it. ■

**Theorem** 3

Assume, for fixed $s$, $legal(s)$. Moreover, assume, for fixed $t$, $a$, $s_1$, and $s_2$, that

$$do(Begin(t), s_1) \sqsubset do(a, s_2) \sqsubset s, \qquad (\dagger)$$

and

$$(\exists a^*, s^*, \mathbf{x})[do(Begin(t), s_1) \sqsubset do(a^*, s^*) \sqsubset do(a, s_2) \wedge writes(a^*, F, \mathbf{x}, t)]. \qquad (\ddagger)$$

We must prove that

$$a = Rollback(t) \supset ((\exists t_1)F(\mathbf{x}, t_1, do(a, s_2)) \equiv (\exists t_2)F(\mathbf{x}, t_2, s_1)), \qquad (\$)$$

and

$$a = Commit(t) \supset ((\exists t_1)F(\mathbf{x}, t_1, do(a, s_2)) \equiv (\exists t_2)F(\mathbf{x}, t_2, s_2)). \qquad (\$\$)$$

**1.** We first prove (\$). Assume $a = Rollback(t)$; then we must show that

$$(\exists t_1)F(\mathbf{x}, t_1, do(a, s_2)) \equiv (\exists t_2)F(\mathbf{x}, t_2, s_1).$$

$\Longrightarrow$ :
Suppose, after eliminating existentials in the conclusion, for fixed $\mathbf{x}$, that $F(\mathbf{x}, t_1, do(a, s_2))$. Then, by the assumption that $a = Rollback(t)$ and that $legal(s)$ holds, Theorem 1 assures us that there is no other $Rollback(t)$ neither a $Commit(t)$ between $do(Begin(t), s_1)$ and $do(a, s_2)$. Furthermore, from (†), the assumption that $a = Rollback(t)$, and the axiom (7), we get, for fixed $F$,

$$\begin{array}{l} a = Rollback(t) \wedge \\ [(\exists a^*, s^*, \mathbf{x}).do(Begin(t), s_1) \sqsubset do(a^*, s^*) \sqsubseteq s \wedge writes(a^*, F, \mathbf{x}, t)] \wedge \\ \qquad\qquad (\exists t')F(\mathbf{x}, t', s_1) \vee \qquad\qquad (*) \\ [(\forall a^*, s^*, \mathbf{x}).do(Begin(t), s_1) \sqsubset do(a^*, s^*) \sqsubseteq s \supset \neg writes(a^*, F, \mathbf{x}, t)] \wedge \\ \qquad\qquad (\exists t')F(\mathbf{x}, t', s). \end{array}$$

Therefore, by assumption (‡), we have to pursue the following case:

$$\begin{array}{l} a = Rollback(t) \wedge \\ [(\exists a^*, s^*, \mathbf{x}).do(Begin(t), s') \sqsubset do(a^*, s^*) \sqsubseteq s \wedge writes(a^*, F, \mathbf{x}, t)] \wedge (\exists t')F(\mathbf{x}, t', s'). \end{array}$$

From this case, we get the following formulas in a straightforward way (by performing some variable renaming): $(\exists t_2)F(\mathbf{x}, t_2, s_1)$, $a = Rollback(t)$, $(\exists a^*, s^*, \mathbf{x}).do(Begin(t), s_1) \sqsubset do(a^*, s^*) \sqsubseteq s \wedge writes(a^*, F, \mathbf{x}, t)$. Henceforth we conclude that $(\exists t_2)F(\mathbf{x}, t_2, s_1)$.

$\Longleftarrow$ :
Assume, for fixed $\mathbf{x}$, that $(\exists t_2)F(\mathbf{x}, t_2, s_1)$. Then, by (‡) and the assumption that $a = Rollback(t)$, we get

$$\begin{array}{l} a = Rollback(t) \wedge (\exists t_2)F(\mathbf{x}, t_2, s_1) \wedge \qquad\qquad (**) \\ \quad (\exists a^*, s^*, \mathbf{x})[do(Begin(t), s_1) \sqsubset do(a^*, s^*) \sqsubset do(a, s_2) \wedge writes(a^*, F, \mathbf{x}, t)]. \end{array}$$

From $(**)$, we get, by assumption (†), the following:

$$\begin{array}{l} a = Rollback(t) \wedge (\exists t_2)F(\mathbf{x}, t_2, s_1) \wedge do(Begin(t), s_1) \sqsubset do(a, s_2) \wedge \\ (\exists a^*, s^*, \mathbf{x})[do(Begin(t), s_1) \sqsubset do(a^*, s^*) \sqsubset do(a, s_2) \wedge writes(a^*, F, \mathbf{x}, t)]. \end{array}$$

We therefore conclude, by axiom (7), that $(\exists t_1)F(\mathbf{x}, t_1, do(a, s_2))$.

**2.** Now we prove (\$\$). Assume that $a = Commit(t)$; then we must prove that
$(\exists t_1)F(\mathbf{x}, t_1, do(a, s_2)) \equiv (\exists t_2)F(\mathbf{x}, t_2, s_2))$.

$\Longrightarrow$ :
Suppose, after removing all the existentials in the conclusion, for fixed $\mathbf{x}$, that $F(\mathbf{x}, t_1, do(a, s_2))$. Since $a = Commit(t)$, by axiom (7), we have

$$(\gamma_F^+(\mathbf{x}, Commit(t), \mathbf{t}_1, s) \vee (\exists t_1)F(\mathbf{x}, t_1, s_2) \wedge \neg\gamma_F^-(\mathbf{x}, Commit(t), \mathbf{t}_1, s)). \qquad (\dagger\dagger)$$

We set $\gamma_F^+(\mathbf{x}, Commit(t), \mathbf{t}_1, s) \equiv false$ and $\neg\gamma_F^-(\mathbf{x}, Commit(t), \mathbf{t}_1, s) \equiv false$.[6] Thus (††) is equivalent to $F(\mathbf{x}, t_2, \mathbf{t}_1, s_2)$, for some $t_2$.

$\Longleftarrow$ : This case is symmetric to the first one.
Suppose, for fixed $\mathbf{x}$, that $(\exists t_2)F(\mathbf{x}, t_2, \mathbf{t}_1, s_2)$. Since $a = Commit(t)$, with the setting of the if-part, $F(\mathbf{x}, t_2, \mathbf{t}_1, s_2)$ is equivalent to

$$(\gamma_F^+(\mathbf{x}, Commit(t), \mathbf{t}_1, s) \vee (\exists t_2)F(\mathbf{x}, t_2, s_2) \wedge \neg\gamma_F^-(\mathbf{x}, Commit(t), s)),$$

which, by axiom (7), is equivalent to $F(\mathbf{x}, t_1, do(a, s_2))$, for some $t_1$. ■

**Theorem** 4

By assuming, for fixed $s$, $s'$, and $t$ $legal(s)$ and $do(Commit(t), s') \sqsubseteq s$, we must prove the claim

$$\bigwedge_{IC \in \mathcal{D}_{IC_e}} IC(do(Commit(t), s')) \wedge \bigwedge_{IC \in \mathcal{D}_{IC^v}} IC(do(Commit(t), s')).$$

A. Assume that $legal(s)$ and $do(Commit(t), s') \sqsubseteq s$. Then, by Lemma 4, we have $legal(do(Commit(t), s'))$. Henceforth, by Lemma 3, $Poss(Commit(t), s')$. Now, by the action precondition axiom for $Commit(t)$, $\bigwedge_{IC \in \mathcal{D}_{IC^v}} IC(s')$ holds. Since $\gamma_F^+(\mathbf{x}, Commit, \mathbf{t}_1, s) \equiv false$ and $\gamma_F^-(\mathbf{x}, Commit, \mathbf{t}_1, s) \equiv false$, then, by axiom (7), we have $F(\mathbf{x}, t^*, s') \equiv F(\mathbf{x}, t, do(Commit(t), s'))$, for any $t^*$. Therefore, $\bigwedge_{IC \in \mathcal{D}_{IC^v}} IC(do(Commit, s'))$.

B. Suppose $s = do(T, s^*)$, where $T$ is a ground transaction. Since $legal(do(T, s^*))$ and suppose $T$ is the sequence $[A_1, \cdots, A_m]$, then, by Lemma 5, we have

$$\bigwedge_{i=1}^{m} Poss(A_i, do([A_1, \cdots, A_{i-1}], s^*)).$$

Since $do(Commit, s') \sqsubseteq do(T, s^*)$, by repeatedly applying axiom (4), we find out that $s^* \sqsubseteq do(Commit, s') \sqsubseteq do(T, s^*)$.
Suppose there are $n$ actions before $Commit(t)$ in $T$. Thus, by the action precondition axioms for updates,

$$\bigwedge_{i=1}^{n} \bigwedge_{IC \in \mathcal{D}_{IC_e}} IC(do([A_1, \cdots, A_{i-1}], s^*)).$$

Henceforth, $\bigwedge_{IC \in \mathcal{D}_{IC_e}} IC(s')$. From this point, we obtain $\bigwedge_{IC \in \mathcal{D}_{IC_e}} IC(do(Commit, s'))$ by a reasoning similar to part A.

By combining A and B, we conclude that the claim holds. ■

**Theorem** 5

We use the relative satisfiability theorem for non-Markovian basic action theories ([12]) stating that a basic action theory $\mathcal{D}$ is satisfiable iff $\mathcal{D}_{una} \cup \mathcal{D}_{S_0}$ is satisfiable. Since the relative satisfiability theorem deals with the general case of first order initial databases, take

[6] In general, whenever an update $a$ does not have any influence on the truth value of a fluent $F$, we set $\gamma_F^+(\mathbf{x}, a, \mathbf{t}_1, s) \equiv false$ and $\neg\gamma_F^-(\mathbf{x}, a, \mathbf{t}_1, s) \equiv false$.

the initial database as being $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \cup \mathcal{D}_{IC}[S_0]$. Therefore, we obtain as immediate consequence that $\mathcal{D}$ is satisfiable. ■

**Theorem** 6

Suppose we fix $s$, $s'$, $t$, and $a'$, and assume $legal(s)$ and $do(Rollback(t), s') \sqsubseteq s$. Thus we must prove

$$committed(a, s') \equiv committed(a, do(Rollback(t), s')), \qquad (*)$$

and

$$rolledBack(a, s') \equiv rolledBack(a, do(Rollback(t), s')). \qquad (**)$$

1. First prove $(*)$.

   $\Longrightarrow$:
   Assume that $committed(a, s')$. Then, by Abbreviation (18), we have

   $$(\exists t^*, s^*).responsible(t^*, a, s') \wedge do(Commit(t^*), s^*) \sqsubseteq s'.$$

   Since $do(Rollback(t), s') \sqsubseteq s$, this implies that

   $$(\exists t^*, s^*).responsible(t^*, a, s') \wedge do(Commit(t^*), s^*) \sqsubseteq s' \wedge do(Rollback(t), s') \sqsubseteq s,$$

   which, by the foundational axioms, and the transitivity of $\sqsubseteq$, implies that

   $$(\exists t^*, s^*).responsible(t^*, a, s') \wedge do(Commit(t^*), s^*) \sqsubseteq s' \sqsubseteq do(Rollback(t), s') \sqsubseteq s.$$

   By (18), the later clearly implies that $committed(a, do(Rollback(t), s'))$.
   $\Longleftarrow$:
   Assume that $committed(a, do(Rollback(t), s'))$. Then, by Abbreviation (18), we have

   $$(\exists t^*, s^*).responsible(t^*, a, do(Rollback(t), s')) \wedge$$
   $$do(Commit(t^*), s^*) \sqsubseteq do(Rollback(t), s').$$

   Since clearly $responsible(t^*, a, do(Rollback(t), s'))$ implies that $responsible(t^*, a, s')$, and, by assumption, $do(Rollback(t), s') \sqsubseteq s$ holds, we conclude that

   $$(\exists t^*, s^*).responsible(t^*, a, s') \wedge do(Commit(t^*), s^*) \sqsubseteq do(Rollback(t), s') \sqsubseteq s,$$

   which, by the foundational axioms, implies that

   $$(\exists t^*, s^*).responsible(t^*, a, s') \wedge do(Commit(t^*), s^*) \sqsubseteq s' \wedge do(Rollback(t), s') \sqsubseteq s.$$

   Again, by (18), the later implies that $committed(a, s')$.
2. The case $(**)$ is proven in a similar way. ■

**Theorem** 7

Let us assume, for fixed $s$, $s'$, and $t$, that $legal(s)$ and $do(Commit(t), s') \sqsubset s$. By Definition 5, we must prove that $\neg transConflict^*(t, t, s)$.

By Lemma 4 and Lemma 5, we get $legal(do(Commit(t), s') \sqsubset s)$ and $Poss(Commit(t), s')$, respectively. Therefore, by the action precondition axiom (12) for $Commit$, we obtain

$$(\forall t')[sc\_dep(t, t', s') \supset (\exists s'')do(Commit(t'), s'') \sqsubseteq s']. \qquad (\dagger)$$

Now, assume, by contradiction that $transConflict^*(t, t, s)$. By Definition 5, we have two cases to pursue:

**case** $transConflict(t, t, s)$. By Abbreviation 4, the contradiction is immediate since $t = t$.
**case** $(\exists t')transConflict(t, t', s) \land transConflict(t', t, s)$. Let us skolemize the existential $(\exists t')$ using the constant $T'$. By Abbreviation 4, we will get some actions $a$, $a'$ for which $t$ and $T'$ are responsible, respectively, such that, for some situations $s^*$ and $s^{**}$, $do(a, s^*) \sqsubset do(a', s^{**}) \sqsubset do(a, s^*)$; moreover, $a$ and $a'$ are conflicting operations. In the classical relational databases, this amounts to $t$ reading from itself. Therefore, by (†), we have $(\exists s'')do(Commit(t'), s'') \sqsubseteq s'$. This however, violates the wellformedness of flat transactions (Theorem 1). ■

**Lemma** 1

Assuming, for fixed $s$, $legal(s)$, and, for fixed $t, s_1, a, s_2, s'$, and $F$ that

$$do(Begin(t), s_1) \sqsubset do(a, s_2) \sqsubset s, \qquad \text{(a)}$$
$$(\exists a^*, s^*, \mathbf{x})[do(Begin(t), s_1) \sqsubset do(a^*, s^*) \sqsubset do(a, s_2) \land writes(a^*, F, \mathbf{x}, t)], \qquad \text{(b)}$$

and

$$(\exists n).a = Rollback(t, n) \land sitAtSavePoint(t, n) = s', \qquad \text{(c)}$$

we must prove that

$$(\exists t_1)F(\mathbf{x}, t_1, do(a, s_2)) \equiv (\exists t_2)F(\mathbf{x}, t_2, s').$$

By skolemizing the existentials in the antecedents and some logical manipulation, we get the following set of assumptions

$$do(Begin(t), s_1) \sqsubset do(Rollback(t, N), s_2) \sqsubset s, \qquad \text{(a')}$$
$$do(Begin(t), s_1) \sqsubset do(a^*, s^*) \sqsubset do(Rollback(t, N), s_2) \land writes(a^*, F, \mathbf{x}, t)], \qquad \text{(b')}$$
$$sitAtSavePoint(t, N) = s', \qquad \text{(c')}$$

With (a')-(c'), we must show that

$$(\exists t_1)F(\mathbf{x}, t_1, do(Rollback(t, N), s_2)) \equiv (\exists t_2)F(\mathbf{x}, t_2, sitAtSavePoint(t, N)).$$

$\Longrightarrow$ :
Suppose that $(\exists t_1)F(\mathbf{x}, t_1, do(Rollback(t, N), s_2))$. Then from (26), we get

$$restoreSavePoint(F, \mathbf{x}, N, t, s),$$

and, by Abbreviation 27, we have $F(\mathbf{x}, t, sitAtSavePoint(t, N))$; henceforth

$$(\exists t_2)F(\mathbf{x}, t_2, sitAtSavePoint(t, N)).$$

$\Longleftarrow$ :
Suppose that $(\exists t_2)F(\mathbf{x}, t_2, sitAtSavePoint(t, N))$. Then we have

$$(\exists s^*)sitAtSavePoint(t, N) = s^* \land F(\mathbf{x}, t_2, s^*).$$

Since $legal(s)$, by assumption (a'), we have $Poss(Rollback(t, N), s_2)$; thus, by Axiom (22), we obtain $sitAtSavePoint(t, N) \sqsubset s$. Therefore, by Abbreviation 27, we conclude that

$$restoreSavePoint(F, \mathbf{x}, N, t, s).$$

So, with the fact that $Poss(Rollback(t,N),s_2)$, we draw the conclusion, by Axiom (26), that $(\exists t_1)F(\mathbf{x},t_1,do(Rollback(t,N),s_2))$. ∎

**Corollary** 1

This follows from Theorem 3, which continues to hold for flat transactions with savepoints, and Lemma 1, by using the fact that $[(P\supset Q)\wedge(P\supset R\wedge S)]\supset[P\supset(Q\wedge R\wedge S)]$. ∎

**Theorem** 8

Asume, for fixed $s$, $t$, $n$, and $s'$, that

$$legal(s), \tag{a}$$
$$do(Rollback(t,n),s')\sqsubset s. \tag{b}$$

Now assume by contradiction that

$$\begin{aligned}&(\exists n^*,s^*).do(Rollback(t,n),s')\sqsubset do(Rollback(n^*),s^*)\sqsubset s\wedge\\ &\quad sitAtSavePoint(n)\sqsubseteq sitAtSavePoint(n^*)\sqsubset do(Rollback(t,n),s').\end{aligned}\tag{\$}$$

By Lemma 4, and assumptions (a), (b), and ($), we have (after skolemizing the existentials in ($)), $legal(Rollback(t,n),s')$ and $legal(Rollback(t,N^*),S^*)$. Therefore, by Lemma 5, we conclude that $Poss(Rollback(t,n),s')$ and $Poss(Rollback(t,N^*),S^*)$. Now, by the action precondition axiom (22), from the fact that $Poss(Rollback(t,N),S^*)$ we get

$$\begin{aligned}&(\exists s_1).s_1 = sitAtSavePoint(t,N)\wedge s_1\sqsubset S^*\wedge\\ &\qquad\neg(\exists s_2,s_3).s_2\sqsubseteq s_1\sqsubseteq s_3\wedge Ignore(t,s_2,s_3),\end{aligned}\tag{c}$$

which is equivalent to

$$\begin{aligned}&sitAtSavePoint(t,N)\sqsubset S^*\wedge\\ &\neg(\exists s_2,s_3).s_2\sqsubseteq sitAtSavePoint(t,N)\sqsubseteq s_3\wedge Ignore(t,s_2,s_3).\end{aligned}\tag{d}$$

Now notice that ($) also implies the following formula:

$$\begin{aligned}&(\exists s_4)s_4 = sitAtSavePoint(t,n)\wedge\\ &sitAtSavePoint(t,n)\sqsubseteq do(Rollback(t,n),s').\end{aligned}\tag{e}$$

Since $sitAtSavePoint(t,n)\ \sqsubset\ do(Rollback(t,n),s')$, by Axiom (23) formula (e) is equivalent to

$$Ignore(t,sitAtSavePoint(t,n),do(Rollback(t,n),s')). \tag{f}$$

However, recall that we have

$$sitAtSavePoint(t,n)\sqsubset sitAtSavePoint(t,N^*)\sqsubset do(Rollback(t,n),s')).$$

This fact, combined with (f) gives

$$(\exists s_7,s_8).s_7 = sitAtSavePoint(t,N^*)s_8\wedge Ignore(t,s_7,s_8)), \tag{g}$$

which by Axiom (22) would make $Poss(Rollback(t,N^*),S^*)$ false, thus leading to a contradiction. ∎

**Theorem** 9

Assume, for fixed $s, s', s''$, and $t$ that

$$do(Chain(t), s') \sqsubset do(Rollback(t), s'') \sqsubseteq s, \tag{a}$$

$$(\exists a^*, s^*, \mathbf{x})[do(Chain(t), s') \sqsubset do(a^*, s^*) \sqsubset do(Rollback(t), s'') \land writes(a^*, F, \mathbf{x}, t)], \tag{b}$$

$$\neg(\exists s^*) do(Chain(t), s') \sqsubset do(Chain(t), s^*) \sqsubset do(Rollback(t), s''). \tag{c}$$

We must prove that

$$(\exists t')F(\mathbf{x}, t', do(Rollback(t), s'')) \equiv (\exists t'')F(\mathbf{x}, t'', do(Chain(t), s')). \qquad (\$\$)$$

$\Longrightarrow$ :
Suppose, for fixed $\mathbf{x}$, that $(\exists t')F(\mathbf{x}, t', do(Rollback(t), s''))$. Then, by the assumption (a), Axiom 7, and Abbreviation 12 that

$$\begin{aligned}
&(\exists s_1).(\forall a^*, s^*)[s_1 \sqsubset do(a^*, s^*) \sqsubset s'' \supset a^* \neq Chain(t) \land a^* \neq Begin(t)] \land \\
&\{[(\exists a^*, s^*, t^*, \mathbf{x}).do(Chain(t), s_1) \sqsubset do(a^*, s^*) \sqsubseteq s'' \land \\
&\qquad writes(a^*, F, \mathbf{x}, t) \land F(\mathbf{x}, t^*, s_1)] \lor \\
&[(\forall a^*, s^*, \mathbf{x}).do(Chain(t), s_1) \sqsubset do(a^*, s^*) \sqsubseteq s'' \supset \neg writes(a^*, F, \mathbf{x}, t)] \land \\
&\qquad (existst^*)F(\mathbf{x}, t^*, s'')\}.
\end{aligned}$$

Therefore, by assumption (b) and (d), we conclude that

$$\begin{aligned}
&(\exists s_1).(\forall a^*, s^*)[s_1 \sqsubset do(a^*, s^*) \sqsubset s'' \supset a^* \neq Chain(t) \land a^* \neq Begin(t)] \land \\
&[(\exists a^*, s^*, t^*, \mathbf{x}).do(Chain(t), s_1) \sqsubset do(a^*, s^*) \sqsubseteq s'' \land writes(a^*, F, \mathbf{x}, t) \land F(\mathbf{x}, t^*, s_1)].
\end{aligned}$$

This entails

$$(\exists s_1, a^*, s^*, t^*, \mathbf{x}).do(Chain(t), s_1) \sqsubset do(a^*, s^*) \sqsubseteq s'' \land writes(a^*, F, \mathbf{x}, t) \land F(\mathbf{x}, t^*, s_1)],$$

which in turn, by assumption (a) and (b), entails $(\exists t^*)F(\mathbf{x}, t^*, s')$.

$\Longleftarrow$ :
Suppose for fixed $\mathbf{x}$ and $F$ that $(\exists t'')F(\mathbf{x}, t'', do(Chain(t), s'))$. Then, by conjoining this with assumptions (b) and (a), we can conclude by Axiom 7 and Abbreviation 12 that $(\exists t')F(\mathbf{x}, t', do(Rollback(t), s''))$. ■

**Theorem** 10

The proof is similar to that of Theorem 3. The major difference lies in the fact that in addition to the assumptions (†) and (‡) made in the proof of Theorem 3, we must also draw the consequences of assuming, for fixed $s$, that $legal(s)$ holds, and, for fixed $t$, $a$, $s_1$, and $s_2$, that

$$do(Spawn(t, t'), s_1) \sqsubset do(a, s_2) \sqsubset s, \tag{†'}$$

and

$$(\exists a^*, s^*, \mathbf{x})[do(Spawn(t, t'), s_1) \sqsubset do(a^*, s^*) \sqsubset do(a, s_2) \land writes(a^*, F, \mathbf{x}, t)]. \tag{‡'}$$

The rest of the proof is as for Theorem 3, but with the successor state axiom 46 for closed nested transactions to be used instead of axiom 7. ■

**Lemma 6** *Suppose $\mathcal{D}$ is a basic relational theory for closed nested transactions. Then any legal log satisfies the weak rollback and commit dependency properties; i.e.,*

$$\begin{array}{l}
\mathcal{D} \models legal(s) \supset \\
\quad (\forall t, t').\{wr\_dep(t,t',s) \supset [do(Rollback(t'),s') \sqsubset s \supset \\
\qquad [(\forall s^*)[s^* \sqsubset s \wedge do(Commit(t),s^*) \not\sqsubseteq do(Rollback(t'),s')] \supset \\
\qquad\qquad (\exists s'')do(Rollback(t),s'') \sqsubseteq s]]\} \wedge \\
\quad \{c\_dep(t,t',s) \supset [do(Commit(t),s^*) \sqsubset s \supset \\
\qquad [do(Commit(t'),s') \sqsubseteq s \supset [do(Commit(t'),s') \sqsubset do(Commit(t),s^*)]]]\}.
\end{array}$$

**Proof**: This is provable in a similar way as for Theorem 2 and we omit the proof here. ■

**Theorem** 11

By Abbreviation 2, we must establish two entailments:

$$\begin{array}{l}
1.\ \mathcal{D} \models legal(s) \supset \\
\quad \{parent(t,t',s) \wedge do(Commit(t'),s') \not\sqsubseteq do(Commit(t),s'') \sqsubset s \supset \\
\qquad (\exists s^*)do(Rollback(t'),s^*) \sqsubset s\}.
\end{array}$$

and

$$\begin{array}{l}
2.\ \mathcal{D} \models legal(s) \supset \\
\quad \{parent(t,t',s) \wedge do(Commit(t'),s') \not\sqsubseteq do(Rollback(t),s'') \sqsubset s \supset \\
\qquad (\exists s^*)do(Rollback(t'),s^*) \sqsubset s\}.
\end{array}$$

1. Assume, for fixed $s, t, t', a, s'$, and $s''$ that

$$legal(s), \tag{a}$$
$$parent(t,t',s), \tag{b}$$
$$do(Commit(t'),s') \not\sqsubseteq do(Commit(t),s'') \sqsubset s. \tag{c}$$

We must prove that $(\exists s^*)do(Rollback(t'),s^*) \sqsubset s$.

By Axiom (33), and the assumptions (b) and (c), we conclude that

$$(\exists s_1)do(Spawn(t,t'),s_1) \sqsubset do(Commit(t),s'') \sqsubset s).$$

Therefore, by the dependency axiom (44), we have $c\_dep(t,t',s'')$.

Since $c\_dep(t,t',s'')$, by Lemma 6 and assumption (a), we obtain

$$\begin{array}{l}
do(Commit(t),s_1) \sqsubset s \supset \\
\quad [do(Commit(t'),s_2) \sqsubseteq s \supset [do(Commit(t'),s_2) \sqsubset do(Commit(t),s_1)]],
\end{array}$$

which is logically equivalent to

$$\begin{array}{l}
do(Commit(t),s_1) \not\sqsubset s \vee do(Commit(t'),s_2) \not\sqsubseteq s \vee \\
do(Commit(t'),s_2) \sqsubset do(Commit(t),s_1),
\end{array}$$

which, in turn, is equivalent to

$$\begin{array}{l}
do(Commit(t),s_1) \sqsubset s \wedge do(Commit(t'),s_2) \not\sqsubset do(Commit(t),s_1) \supset \\
\qquad do(Commit(t'),s_2) \not\sqsubseteq s.
\end{array} \tag{d}$$

By assumption (c), appropriate unification, (d), and Modus Ponens, we get

$$(\forall s_2)do(Commit(t'),s_2) \not\sqsubseteq s.$$

Finally, by Theorem 1, we conclude that $(\exists s^*)do(Commit(t'), s^*) \sqsubseteq s$, which implies QED.

2. We make the same assumptions as in Part 1 of the proof, except that the following:

$$do(Commit(t'), s') \not\sqsubseteq do(a, s'') \sqsubset s \qquad \text{(c')}$$

replaces (c). We must prove that $(\exists s^*)do(Rollback(t'), s^*) \sqsubset s$.
By Axiom (33), and assumptions (b) and (c'), we get

$$(\exists s_1)do(Spawn(t, t'), s_1) \sqsubset do(Rollback(t), s'') \sqsubset s).$$

Thus, by the dependency axiom (45), we conclude that $wr\_dep(t', t, s'')$.
Since $c\_dep(t', t, s'')$, by Lemma 6 and assumption (a), we obtain

$$\begin{aligned}&[do(Rollback(t), s_1) \sqsubset s \supset \\ &\qquad [(\forall s^*)[s^* \sqsubset s \wedge do(Commit(t), s^*) \not\sqsubseteq do(Rollback(t), s_1)] \supset \\ &\qquad\qquad (\exists s'')do(Rollback(t), s'') \sqsubseteq s]],\end{aligned}$$

which is logically implies that

$$\begin{aligned}&do(Rollback(t), s_1) \sqsubset s \wedge do(Commit(t'), s^*) \not\sqsubseteq do(Rollback(t), s_1)] \supset \\ &\qquad (\exists s'')do(Rollback(t'), s'') \sqsubseteq s.\end{aligned} \qquad \text{(d')}$$

By assumption (c), appropriate unification, (d'), and Modus Ponens, we conclude that

$$(\exists s^*)do(Commit(t'), s^*) \sqsubseteq s,$$

which implies QED. ■

**Theorem** 12

The proof is very similar to that of Theorem 7 and needs not be repeated here. ■