Expressing Transactions with Savepoints as Non-Markovian Theories of Actions

Iluju Kiringa
School of Info. Technology and Eng.
University of Ottawa

Alfredo Gabaldon
Department of Computer Science
University of Toronto

Abstract

Flat transactions with savepoints are a variation of the classical flat transactions that allows the user to go undo work done so far back to a certain point within the transaction. This is as opposed to pure classical flat transactions that either commit to whole work done so far or undo it. Recently, this mechanism is being offered by some major database products. Their semantics, however, seem not to be as well studied as the classical flat transactions. In this paper, we show how to use non-Markovian control in the situation calculus to capture flat transactions with savepoints. We also state some of their properties.

1 Introduction

Transaction processing is now a well established domain of research within the area of databases [2]. A classical flat database transaction is a sequence of operations[1] on the database content. This sequence is required to exhibit the so called Atomicity-Consistency-Isolation-Durability (ACID) properties, which are enforced by means of a set of special actions including $Begin$, $Commit$, and $Rollback$. After a user enters a sequence of actions to the database management system (DBMS), the system may either commit, i.e. make the changes done since the beginning of the transaction permanent, or roll back, i.e. discard all the changes done so far since the beginning of the transaction. In addition to this all-or-nothing logic for making changes durable, the DBMS ensures that, provided that the start database state is consistent, the new database state reached after a transaction has committed or rolled back is also consistent. Moreover, the system ensures that transactions running in parallel do not interfere with each other.

Flat transactions with savepoints are a variation of flat transactions which provides the user with a new action $Save(t)$ to establish savepoints in the database log [2]. 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 database 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 database actions including $Save(t)$ and $Rollback(t, n)$, except $Begin(t)$, $Commit(t)$, and $Rollback(t)$; a_{n-2} may be $End(t)$.

The 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, instead of rolling back only to the beginning of the transaction. If a $Rollback(t, n)$ action is executed in the

[1] From the point of view of theories of actions in knowledge representation, database operations are a type action, thus, throughout the paper, we will frequently refer to database operations as actions.

database state s, its effect is that we ignore any sequence of actions executed between some state s' and s, where s' is the database state corresponding to the savepoint n.

Though this mechanism has been described a decade ago in [2], it is only recently that some major database vendors are offering it in their products, e.g. [8]. In Oracle, for example, the user can bookmark the beginning of a named section of a transaction. Later, she has the choice of discarding that section by rolling the transaction back to the bookmarked begin point. Oracle has two SQL constructs to that end:

`SAVEPOINT` $<$ `bookmark` $>$, for creating a bookmark and

`ROLLBACK TO` $<$ `bookmark` $>$, for rolling back to a created bookmark.

This very simple pair of constructs can have intricate consequences when several savepoints have been created. E.g., the system should prevent one from rolling back to savepoints that are in portions of the transaction that have already been discarded by a partial rollback. To fully understand this consequence and many others, one needs a well defined semantics of the savepoint mechanism. The semantics of transactions with savepoints, however, seems not to be as well studied as that of the classical flat transactions. In this paper, we present a logical specification of flat transactions with savepoints as a non-Markovian theory of actions in the situation calculus, and state some of their properties. Non-Markovian theories of the situation calculus [1] are special theories of actions in which one may refer to past states other than the previous one. We provide the formal semantics of flat transactions with savepoints by specifying it as a special case of non-Markovian theories of the situation calculus called *basic relational theories* (BRT) introduced in [3, 4]. A BRT is a set of sentences suitable for non-Markovian control in the context of database transactions. With the formalization of flat transactions with savepoints in hand, it is possible to express the ACID properties in the same language and prove that they are logical theorems of the axiomatization.

2 Non-Markovian Action Theories in the Situation Calculus

The situation calculus [7] is a logical framework that has been successfully employed in the formalization of a wide variety of dynamical systems [12]. In this section we give overviews of the situation calculus basic action theories of Reiter [11] and of non-Markovian theories as presented in [1].

2.1 The Situation Calculus

The situation calculus is a second order logic language with three basic components: actions, situations, and fluents. Actions are responsible for all the changes in the world. Situations are sequences of actions which represent possible histories of the world, or in the case of databases, database logs. Fluents are properties of the world that change from situation to situation as a result of the execution of actions. Formally, the language has three sorts: *action*, *situation* and *fluent*. In addition to variables of these sorts, the language includes functions such as $move(x, y)$ and $accounts\_insert(aid, abal)$ to represent actions, a constant S_0 and a function $do(a, s)$ for situations such as $do(accounts\_insert(aid, abal), S_0)$, and predicates for representing fluents such as $accounts(aid, bid, do(a, s))$. The initial situation, or empty history, is denoted by the constant S_0. Non-empty histories are built by means of the function *do*. For a complete formal description see [9, 12]. We will use the shorthand notation $do([a_1, \ldots, a_k], \sigma)$ to denote the situation term $do(a_k, do(a_{k-1}, \ldots do(a_1, \sigma) \ldots))$. We will say that a term $do([a_1, \ldots, a_k], \sigma)$ is *rooted at* σ if σ is S_0 or a situation variable and is the only (if any) subterm of sort situation occuring in $a_1, \ldots, a_k$.

A situation calculus axiomatization of a domain, includes the following set of axioms[2] :

1. For each action function $A(\vec{x})$, 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 $F(\vec{x}, s)$, 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 $\mathcal{D}_f$, is called a (Markovian) *basic action theory*.

2.2 Non-Markovian Action Theories

For a basic action theory without the Markov assumption, we need some definitions. These are based on those in [1].

First, we need to introduce the following abbreviations:

$$
\begin{aligned}
(\exists s : \sigma' \sim \sigma'' \sim \sigma)W &\stackrel{\text{def}}{=} (\exists s)[\sigma' \sim \sigma'' \wedge \sigma'' \sim \sigma \wedge W] \\
(\forall s : \sigma' \sim \sigma'' \sim \sigma)W &\stackrel{\text{def}}{=} (\forall s)[(\sigma' \sim \sigma'' \wedge \sigma'' \sim \sigma) \supset W]
\end{aligned}
\tag{1}
$$

where $\sim$ stands for $\sqsubset$, $=$ or $\sqsubseteq$, and variable s appears in σ''. If σ' is S_0 then we may write $(\exists s : \sigma'' \sim \sigma)W$ and $(\forall s : \sigma'' \sim \sigma)W$ instead.

Definition 1 (Bounded Formulas) For $n \geq 0$, let σ be a term $do([\alpha_1, \ldots, \alpha_n], \lambda)$ rooted at λ. The formulas of $\mathcal{L}_{sitcalc}$ *bounded* by σ are the smallest set of formulas such that:

1. If W is an atom whose situation terms are all rooted at λ, then W is bounded by σ.

2. If W', W'' are formulas bounded by situation terms rooted at s and λ, respectively, then $(\exists s : \sigma' \sim \sigma'' \sim \sigma)W$ and $(\forall s : \sigma' \sim \sigma'' \sim \sigma)W$ are formulas bounded by σ, where σ'' is rooted at s and $W = (\neg)(W' \wedge W'')$.

3. If W_1, W_2 are formulas bounded by situation terms rooted at λ, then $\neg W_1$, $W_1 \wedge W_2$ and $(\exists v)W_1$, where v is of sort *action* or *object*, are formulas bounded by σ.

The set of formulas *strictly bounded* by σ is similarly defined by requiring in item (1) above that all situation terms of W be subterms of σ, in item (2) that W' be strictly bounded by a subterm of σ'' and W'' by a subterm of σ; and in item (3) that W_1, W_2 be strictly bounded by subterms of σ.

[2] We will adopt the convention that free variables in formulas are implicitely universally quantified from the outside.

Non-Markovian basic action theories differ from those in the previous subsection in that preconditions and effects of actions may depend on any past situation, not only on the current one.

Formally, the rhs, $\Pi_A(x_1, \ldots, x_n, s)$, of an action precondition axiom in a non-Markovian basic action theory is a formula bounded by situation term s which does not mention predicate $Poss$ and may refer to past situations through abbreviations (1). Similarly, the rhs, $\Phi_F(x_1, \ldots, x_n, a, s)$, of a successor state axiom is a formula strictly bounded by s.

A final comment on notation: the abbreviations (1) help defining the class of bounded formulas, but frequently it is helpful to use further notation shorthands. An instance we will use later is the following: we will write $(\exists s_1, s_2, s_3 : \sigma_0 \sqsubseteq \sigma_1 \sqsubseteq \sigma_2 \sqsubseteq \sigma_3 \sqsubseteq \sigma)\phi$ as a shorthand for:

$$
\begin{aligned}
&(\exists s_1 : \sigma_0 \sqsubseteq \sigma_1 \sqsubseteq \sigma) \\
&\qquad (\exists s_2 : \sigma_1 \sqsubseteq \sigma_2 \sqsubseteq \sigma) \\
&\qquad\qquad (\exists s_3 : \sigma_2 \sqsubseteq \sigma_3 \sqsubseteq \sigma)\phi.
\end{aligned}
$$

3 Flat Transactions Models

This section introduces a characterization of flat transactions in terms of the non-Markovian action theories described in Section 2.

To represent database actions, we distinguish the primitive internal actions, which are fluent-changing actions (e.g. $F\_insert(\vec{x}, t)$), from *primitive external actions*, which are $Begin(t)$, $Commit(t)$, $End(t)$, and $Rollback(t)$, and whose meaning will be clear in the sequel of the paper; these are external as they do not specifically affect the content of the database. The argument t is a unique transaction identifier. 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. Finally, database fluents have an official transaction argument that denotes the transaction that made the last changes to the fluent.

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

Foundational Axioms. These are the same as in BATs.

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 $A(\vec{x}, t)$, with syntactic form

$$Poss(A(\vec{x}, t), s) \equiv (\exists t')\Pi_A(\vec{x}, t', s) \wedge IC_e(do(A(\vec{x}, t), s)) \wedge running(t, s). \tag{2}$$

Here, $\Pi_A(\vec{x}, t, s)$ is a first order formula with free variables among $\vec{x}, t$, and s; it is similar to the right hand side of action precondition axioms of Section 2. These axioms characterize the preconditions of the update A; $IC_e(s)$ and $running(t, s)$ are abbreviations expanded as follows:

Abbreviation 1 $IC_e(s) \stackrel{\text{def}}{=} \bigwedge_{IC \in \mathcal{IC}_e} IC(s)$.

Here, $IC(s)$ is a formula whose only free variable is s and this is also its only term of sort *situation*.

Abbreviation 2

$$running(t,s) \stackrel{\text{def}}{=} (\exists s' : do(Begin(t),s') \sqsubseteq s)$$
$$(\forall a)(\forall s'' : do(Begin(t),s') \sqsubset do(a,s'') \sqsubset s)[a \neq Rollback(t) \wedge a \neq End(t)].$$

Example 1 Suppose we have a Banking database with personal accounts information stored in a relation (i.e., a fluent) $accounts(aid, abal, t, s)$ and tuples are inserted and deleted by executing actions a_insert and a_delete, respectively. The following axiom states that it is possible to delete a tuple specifying the teller identity aid and balance $abal$ into the $accounts$ relation relative to the database log s iff, as a result of performing the actions in the log, that tuple is present in the $accounts$ relation, the integrity constraints are satisfied by the resulting log after executing the delete, and transaction t is running.

$$\begin{aligned} Poss(a\_delete(aid, abal, t), s) \equiv & (\exists t')accounts(aid, abal, t', s) \wedge \\ & IC_e(do(a\_delete(aid, abal, t), s)) \wedge running(t, s). \end{aligned} \tag{3}$$

Successor State Axioms. These have the syntactic form

$$\begin{aligned} F(\vec{x}, t, do(a, s)) \equiv & (\exists \vec{t_1})\Phi_F(\vec{x}, a, \vec{t_1}, s) \wedge \neg(\exists t'')a = Rollback(t'') \vee \\ & (\exists t'')\{a = Rollback(t'') \wedge restoreBeginPoint(F, \vec{x}, t'', s)\}. \end{aligned} \tag{4}$$

There is one such axiom for each database relational fluent F. Formula $\Phi_F(\vec{x}, a, \vec{t}, s)$ has free variables among $\vec{x}, a, \vec{t}, s$ and will typically have the canonical form [12]:

$$\gamma_F^+(\vec{x}, a, \vec{t}, s) \vee F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, \vec{t}, s), \tag{5}$$

where $\gamma_F^+(\vec{x}, a, \vec{t}, s)$ ($\gamma_F^-(\vec{x}, a, \vec{t}, s)$) 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.

The predicate $restoreBeginPoint(F, \vec{x}, t, s)$ is defined as follows:

Abbreviation 3

$$\begin{aligned} & restoreBeginPoint(F, \vec{x}, t, s) \stackrel{\text{def}}{=} \\ & \{(\exists a_1, a_2, t')(\exists s', s_1, s_2 : do(Begin(t), s') \sqsubset do(a_2, s_2) \sqsubset do(a_1, s_1) \sqsubseteq s). \\ & \quad writes(a_1, F, \vec{x}, t) \wedge writes(a_2, F, \vec{x}, t') \wedge \\ & \quad\; [(\forall a'')(\forall s'' : do(a_2, s_2) \sqsubset do(a'', s'') \sqsubset do(a_1, s_1))\neg writes(a'', F, \vec{x}, t)] \wedge \\ & \quad\; [(\forall a'')(\forall s'' : do(a_1, s_1) \sqsubset do(a'', s'') \sqsubseteq s)\neg(\exists t'')writes(a'', F, \vec{x}, t'')] \wedge (\exists t'')F(\vec{x}, t'', s_1)\} \vee \\ & \{[(\forall a^*)(\forall s^*, s' : do(Begin(t), s') \sqsubset do(a^*, s^*) \sqsubseteq s).\neg writes(a^*, F, \vec{x}, t)] \wedge (\exists t')F(\vec{x}, t', s)\}. \end{aligned}$$

Here, $writes(a, F, \vec{x}, t)$ stands for $a = F\_insert(\vec{x}, t) \vee F\_delete(\vec{x}, t)$. This formula is clearly bounded by s and can be rewritten to fit the form described in Definition 1.

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 (4). Intuitively, $restoreBeginPoint(F, \vec{x}, t, s)$ means that the system restores the value of the database fluent F with arguments $\vec{x}$ in a particular way:

- The first disjunct in Abbreviation 3 captures the scenario where the transactions t and t' running in parallel, and writing into and reading from F are such that t overwrites whatever t' writes before t rolls back. Suppose that t and t' are such that t begins, and eventually writes into F before rolling back; t' begins after t has begun, writes into F before the last write action of t, and commits before t rolls back.
- The second disjunct in Abbreviation 3 captures the case where the value F had at the beginning of the transaction that rolls back is kept.

Given the actual situation s, the successor state axiom characterizes the truth values of the fluent F in the next situation $do(a, s)$ in terms of all the past situations. Notice that Abbreviation 3 captures the intuition that $Rollback(t)$ affects all tuples within a table.

Example 2 Consider again the banking example. The following successor state axiom (6) states that the tuple $(aid, abal)$ will be in the $accounts$ 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 $accounts$ 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 $accounts$ relation will get a value according to the logic of Abbreviation (3).

$$\begin{aligned} & accounts(aid, abal, t, do(a, s)) \equiv \\ & \quad ((\exists t_1)a = a\_insert(aid, abal, t_1) \vee (\exists t_2)accounts(aid, abal, t_2, s) \wedge \\ & \quad\quad \neg(\exists t_3)a = a\_delete(aid, abal, t_3)) \wedge \neg(\exists t')a = Rollback(t') \vee \\ & \quad (\exists t').a = Rollback(t') \wedge restoreBeginPoint(accounts, (aid, abal), t', s). \end{aligned} \tag{6}$$

In this successor state axiom, the formula

$$(\exists t_1)a = a\_insert(aid, abal, t_1) \vee (\exists t_2)accounts(aid, abal, t_2, s) \wedge \neg(\exists t_3)a = a\_delete(aid, abal, t_3)$$

corresponds to the canonical form (5).

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:

$$Poss(Begin(t), s) \equiv \neg(\exists s' : do(Begin(t), s') \sqsubseteq s)True, \tag{7}$$

$$Poss(End(t), s) \equiv running(t, s), \tag{8}$$

$$\begin{aligned} & Poss(Commit(t), s) \equiv (\exists s' : do(End(t), s') = s) \\ & \bigwedge_{IC \in \mathcal{IC}_v} IC(s) \wedge (\forall t')[sc\_dep(t, t', s) \supset (\exists s'' : do(Commit(t'), s'') \sqsubseteq s)True], \end{aligned} \tag{9}$$

$$\begin{aligned} & Poss(Rollback(t), s) \equiv (\exists s' : do(End(t), s') = s) \\ & \quad \neg \bigwedge_{IC \in \mathcal{IC}_v} IC(s)] \vee (\exists t')(\exists s'' : do(Rollback(t'), s'') \sqsubseteq s)r\_dep(t, t', s) \end{aligned} \tag{10}$$

Notice that our axioms (7)–(10) assume that the user will only use internal actions $Begin(t)$ and $End(t)$ and the system will execute either $Commit(t)$ or $Rollback(t)$. The predicates $r\_dep(t, t', s)$, $sc\_dep(t, t', s)$ are called dependency predicates: $r\_dep(t, t', s)$ intuitively says that transaction t is rollback dependent on transaction t' in the situation s, and $sc\_dep(t, t', s)$

says that transaction t is strong commit dependent on transaction t' in the situation s.[3]

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

$$dep(t, t', s) \equiv \mathcal{C}(t, t', s), \tag{11}$$

where $\mathcal{C}(t, t', s)$ is a condition involving a relationship between 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. In the classical case, we have the following axioms:

$$r\_dep(t, t', s) \equiv transConflict(t, t', s), \tag{12}$$

$$sc\_dep(t, t', s) \equiv readsFrom(t, t', s). \tag{13}$$

The first axiom says that a transaction conflicting with another transaction generates a rollback dependency of t on t', i.e., if t' rollbacks, then t must also rollback. The second says that a transaction reading from another transaction generates a strong commit dependency of t on t', i.e., if t' commits, then t must also commit. These are called 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

$$(\forall \vec{x}, t).F(\vec{x}, t, S_0) \equiv \vec{x} = \vec{C}^{(1)} \vee \ldots \vee \vec{x} = \vec{C}^{(r)}, \tag{14}$$

one for each (database or system) fluent F. Here, the $\vec{C}^i$ are tuples of constants. Also, $\mathcal{D}_{S_0}$ includes unique name axioms for constants of the database. Axioms of the form (14) say that our theories accommodate a complete initial database state, which is commonly the case in relational databases as unveiled in [10]. 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 transaction models 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 2 *(***Basic Relational Theory***) A situation calculus theory $\mathcal{D}$ 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. *$\mathcal{D}$ mentions, 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] A transaction t is rollback depend on transaction t' iff, whenever t' rolls back in a log, then t must also roll back in that log; t is strong commit depend on t' iff, whenever t' commits in s, then t must also commit in s

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 (2), one for each primitive internal action.*

5. *$\mathcal{D}_{ss}$ is a set of non-Markovian successor state axioms of the form (4), one for each database fluent. 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.*

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*

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

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

3.1 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*:

Abbreviation 4

$$systemAct(a, t) \stackrel{\mathrm{def}}{=} a = Commit(t) \vee a = Rollback(t).$$

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

Abbreviation 5

$$\begin{array}{l} legal(s) \stackrel{\mathrm{def}}{=} (\forall a, s^* : do(a, s^*) \sqsubseteq s) Poss(a, s^*) \wedge \\ \quad (\forall a', a'', t)[systemAct(a', t) \wedge responsible(t, a') \wedge responsible(t, a'') \wedge \\ \quad (\forall s' : do(a'', s') \sqsubset s)[Poss(a', s') \supset a' = a'']. \end{array} \tag{15}$$

Here, $responsible(t, a)$ means that transaction t is responsible for the action a, i.e., a occurred as part of this transaction. For each action function $A(\vec{x}, t)$, we include an axiom $responsible(t, A(\vec{x}, t)) \equiv True$ to define this relation.

In proving the different properties of transaction models, the following three lemmas exhibiting simple properties of legal logs are useful.[4]

[4] Notice that we will only state these lemmas and the subsequent properties for which they are useful without proofs.

Lemma 1 *Let $\mathcal{D}$ be a basic relational theory. Then*

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

Lemma 2 *Suppose $\mathcal{D}$ is a basic relational theory. Then*

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

Lemma 3 *Suppose $[a_1, \cdots, a_n]$ is a sequence of ground action terms. Then*

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

Simple properties such as well-formedness of atomic transactions [6] can be formulated in the situation calculus and proven as logical consequences of basic relational theories [4]. We leave this issue out of the scope of this paper, except that we can establish the following:

Theorem 1 *(**Atomicity**) Suppose $\mathcal{D}$ is a relational theory. Then for every database fluent F*

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

This says that rolling back restores any 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.

4 Flat Transactions with Savepoints

Recall that the external action $Rollback(t, n)$ brings the database back to the database state corresponding to the savepoint n. 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:

$$Poss(Save(t), s) \equiv running(t, s), \tag{16}$$

$$\begin{aligned} Poss(Rollback(t), s) \equiv\ & (\exists s' : do(End(t), s') = s)True \wedge \neg \bigwedge_{IC \in \mathcal{IC}_v} IC(s)] \vee \\ & (\exists t')(\exists s'' : do(Rollback(t'), s'') \sqsubseteq s)r\_dep(t, t', s) \vee \\ & (\exists t', n)(\exists s', s^*, s^{**} : s' \sqsubseteq s^* \sqsubset do(Rollback(t', n), s^{**}) \sqsubseteq s) \\ & \quad [r\_dep(t, t', s^*) \wedge sitAtSavePoint(t', n, s')] \end{aligned} \tag{17}$$

$$\begin{aligned} Poss(Rollback(t, n), s) \equiv\ & running(t, s) \wedge \\ & (\exists s' : s' \sqsubset s).sitAtSavePoint(t, n, s') \wedge \\ & numOfSavePoints(t, s) \geq n \wedge \\ & \neg(\exists s^*, s^{**} : s^* \sqsubseteq s' \sqsubseteq s^{**} \sqsubseteq s)Ignore(t, s^*, s^{**}), \end{aligned} \tag{18}$$

$$\begin{aligned} Ignore(t, s', do(a, s)) \stackrel{\text{def}}{=}\ & s' \sqsubseteq do(a, s) \wedge \\ & (\exists n).sitAtSavePoint(t, n, s') \wedge a = Rollback(t, n), \end{aligned} \tag{19}$$

$$\begin{aligned} numOfSavePoints(t, do(a, s)) = n \equiv\ & a = Begin(t) \wedge n = 1 \vee \\ & a = Save(t) \wedge n = numOfSavePoints(t, s) + 1 \vee \\ & numOfSavePoints(t, s) = n \wedge a \neq Begin(t) \wedge a \neq Save(t), \end{aligned} \tag{20}$$

$$\begin{aligned} sitAtSavePoint(t, n, s) \stackrel{\text{def}}{=}\ & numOfSavePoints(t, s) = n \wedge \\ & (\exists a)(\exists s' : do(a, s') = s)(a = Begin(t) \vee a = Save(t)). \end{aligned} \tag{21}$$

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.

$$\begin{aligned} & F(\vec{x}, t, do(a, s)) \equiv \\ & \quad (\exists \vec{t_1}).\Phi_F(\vec{x}, a, \vec{t_1}, s) \wedge \neg(\exists t'')a = Rollback(t'') \wedge \neg(\exists t'', n)a = Rollback(t'', n) \vee \\ & \quad (\exists t'')a = Rollback(t'') \wedge restoreBeginPoint(F, \vec{x}, t'', s) \vee \\ & \quad (\exists n, t'').a = Rollback(t'', n) \wedge restoreSavePoint(F, \vec{x}, n, t'', s), \end{aligned} \tag{22}$$

one for each relation F, where $\Phi_F(\vec{x}, a, \vec{t_1}, s)$ is a formula with free variables among $a, s, \vec{x}, \vec{t_1}$; Abbreviation (3) defines $restoreBeginPoint(F, \vec{x}, t'', s)$, and $restoreSavePoint(F, \vec{x}, n, t'', s)$ is defined as follows:

Abbreviation 6

$$\begin{aligned} & restoreSavePoint(F, \vec{x}, n, t, s) \stackrel{\text{def}}{=} \\ & \qquad (\exists s' : s' \sqsubset s)sitAtSavePoint(t, n, s') \wedge (\exists t')F(\vec{x}, t', s'), \end{aligned} \tag{23}$$

where $sitAtSavePoint(t, n, s')$ is a formula that provides the log relative to the transaction t at the savepoint n as defined by (21); $restoreSavePoint(F, \vec{x}, n, t, s)$ means that the value of the fluent F with arguments $\vec{x}$ 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 (11) 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

$$\begin{aligned} &dep(t,t',do(a,s)) \equiv \mathcal{C}(t,t',s) \wedge a \neq Rollback(t) \wedge a \neq Rollback(t')\wedge \\ &\quad \neg(\exists n)(\exists s' : s' \sqsubseteq s)[(a = Rollback(t,n) \vee a = Rollback(t',n))\wedge \\ &\quad sitAtSavePoint(t',n,s') \wedge (\forall s'' : s' \sqsubseteq s'' \sqsubseteq s)\neg dep(t,t',s'')], \end{aligned} \tag{24}$$

where $\mathcal{C}(t,t')$ and $dep(t,t',s)$ are defined as in (11). We have one such axiom for each dependency predicate.

A basic relational theory for flat transactions with savepoints is as in Definition 2, but where the relational language includes $Save(t)$ and $Rollback(t,n)$ as further actions, the axioms (16) – (21) are added to $\mathcal{D}_{FT}$, the set $\mathcal{D}_{ss}$ is a set of successor state axioms of the form (22), and the set $\mathcal{D}_{dep}$ is a set of dependency axioms of the form (24). All the other axioms of Definition 2 remain unchanged.

4.1 Properties

Now we turn to some of the ACID properties of flat transactions with savepoints. The introduction of the $Rollback(t,n)$ action modifies the previous atomicity theorem as follows.

Lemma 4 *Suppose $\mathcal{D}$ is a relational theory. Then for every relational fluent F*

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

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 1 continues to hold for flat transactions with savepoints. Hence, from Theorem 1 and Lemma 4, we have the following

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

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

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

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

5 Conclusion

We have presented a logical specification of flat transactions with savepoints using non-Markovian theory of actions. We have also captured some of their properties as non-Markovian sentences. The formal semantics of flat transactions with savepoints has been given as a special case of basic relational theories, which are non-Markovian theories of the situation calculus. Finally the sentences capturing the properties of flat transactions with savepoints were shown to be logical consequences of the basic relational theories.

For basic action theories with the Markov assumption, Pirri & Reiter [9] define a provenly correct *regression* mechanism that takes a situation calculus sentence and, under certain restrictions on the form of this sentence, transforms it into an equivalent sentence whose only situation term is S_0. This allows proving sentences without appealing to the foundational axioms $\mathcal{D}_f$. This regression operator was generalized for non-Markovian theories in [1]. A future avenue that we would like to follow is to provide an interpreter for simulating transaction programs that uses non-Markovian theories as background axioms in the sense of [5]. Such an interpreter needs the kind of regression defined in [1] for efficient theorem proving.

References

[1] A. Gabaldon. Programming hierarchical task networks in the situation calculus. In AIPS'02, 2002.

[2] J. Gray and Reuter A. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers, San Mateo, CA, 1995.

[3] I Kiringa. Simulation of advanced transaction models using golog. In DBPL'01, 2001.

[4] I. Kiringa. Towards a theory of advanced transaction models in the situation calculus (extended abstract). In KRDB'01, 2001.

[5] H. Levesque, R. Reiter, Y. Lespérance, Fangzhen Lin, and R.B. Scherl. Golog: A logic programming language for dynamic domains. *J. of Logic Programming*, 31(1-3):59–83, 1997.

[6] N. Lynch, M.M. Merritt, W. Weihl, and A. Fekete. A theory of atomic transactions. In ICDT'88, 1988.

[7] J. McCarthy. Situations, actions and causal laws. Technical report, Stanford University, 1963.

[8] J. Morrison and M. Morrison. *Guide to Oracle 9i*. Thomson, Boston, MA, 2003.

[9] F. Pirri and R. Reiter. Some contributions to the metatheory of the situation calculus. *Journal of the ACM*, 46(3):325–364, 1999.

[10] 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, Springer, 1984.

[11] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380, San Diego, Academic Press, 1991.

[12] R. Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. MIT Press, Cambridge, 2001.