A language to reason about long duration transactions and their concurrent execution

Chitta Baral*
Department of Computer Sc.
Univ. of Texas at El Paso
El Paso, TX, 79968, USA
chitta@cs.utep.edu

Jorge Lobo†
Department of EECS
Univ. of Illinois at Chicago
Chicago, IL 60607, USA
jorge@eecs.uic.edu

Abstract

In this paper we describe a language for long duration transactions. Our language is inspired by both action-description languages used in artificial intelligence and earlier transaction specification languages such as NT/PV [7]. We allow nested transactions, specification of ordering between sub-transactions, and constraints that hold at different levels of nesting. The semantics of our language is based on the automata-based semantics of action description languages, and allows us to define an entailment relation between a transaction description and queries that specify if a fluent is true after the execution of a sequence of transactions. This entailment relation forms the basis of our formalization of notions such as the correctness of a transaction definition, serializability of a schedule etc. We also define a language to specify semantic informations about allowable interleaving and show how it increases concurrency.

1 Introduction and motivation

Long-duration transactions and workflow systems applications have made researchers to revisit traditional transaction model definitions. The concept of locks is still important for long-duration transactions to ensure the correctness of the elementary sub-transactions, but its primitive semantics is too restrictive in expressing the conditions to ensure the correctness of the concurrent execution of long-term transactions.

Consistency of traditional concurrent transaction systems is usually enforced by some kind of serializability criteria which only consider syntactic properties of the interleaving trans-

*Support was provided by the National Science Foundation, under grant Nr. IRI-9211662 and IRI-9501577.

†Support was provided by the National Science Foundation, under grant Nr. IRI-92-10220.

actions. In long-lived transaction systems, transactions must be split into smaller sub-transactions to be able to obtain reasonable concurrency through out the duration of the transaction. In this new schema of interleaving sub-transactions simple syntactic criteria are too restrictive to express consistency. Serializability may not be possible although the sequence of transactions may produce a semantically valid result. Consistency in long duration transaction systems is usually maintained based on semantic information provided by the application developer together with the transactions. This information usually comes in the form of preconditions and postconditions attached to each transaction plus an order (not necessarily partial or total) over the transactions. The concurrent execution of transactions is considered correct if it is executed in a database state that satisfies the preconditions of the transactions and is executed consistently with the order. Postconditions specify properties of the database state that results from the execution of transactions. The specification of the pre and postconditions and finding the correct order is the responsibility of the application developer. This task is not simple since the developer needs to take into account the interaction among several transactions and sub-transactions. Even for a single transaction the verification of consistency becomes more complicated since the database can move from a global consistent state to a global inconsistent and back to a consistent state before it reaches the end of the transaction. With standard transactions these inconsistent states were not seen by other transactions because of the *isolation* of the transaction. The application developer must also deal with temporary inconsistent states created by sub-transactions.

The goal of this paper is to introduce a specification language that will assist the system developer to design correct transaction systems. We will call the language $\mathcal{T}_C$. The formal foundations of this language come from the theory of action languages in Artificial Intelligence in which we equate actions with transactions and it was inspired by the language $\mathcal{A}$ [6]. Similar to [2], specifications written in our language will assist the system implementor in the correct decomposition and design of long-lived transactions, but in a much richer environment. We introduce $\mathcal{T}_C$ in two steps. First, we will present a simpler language called $\mathcal{T}$. In $\mathcal{T}$, we can describe nested transactions, specification of ordering between sub-transactions, and constraints that hold at different levels of nesting. No interleaving is allowed among transactions. We then introduce $\mathcal{T}_C$ to specify semantic informations about allowable interleaving and extend the ordering with the specification of sub-transactions that can preempt the execution of other sub-transactions.

2 The language $\mathcal{T}$

In this section we introduce and develop a transaction description language called $\mathcal{T}$ to adequately express long duration transactions.

The language $\mathcal{T}$ consists of two disjoint non-empty sets of symbols called *fluent names* and *transaction names*[1] (or simple fluents and transactions), and four kinds of *propositions*:

[1]We use both T's and t's to denote transaction names.

composition propositions, *effect* propositions, *constraint* propositions and *order* propositions. A composition proposition is an expression of the form:

$$T \textbf{ consists of } \{T_b, T_1, \ldots, T_n, T_e\}. \tag{1}$$

where T, the T_i's ($n \geq 0$), T_b and T_e are transaction names. We will call (1) the *definition* of T. T_b and T_e are special transaction names associated with the transaction denoted by T. They represent the *beginning* and the *end* of the transaction respectively. We will also say that T is the *parent* transaction of T_b and T_e. Composition propositions define nested transactions. They are restricted syntactically to describe hierarchical nesting, and no transaction name may have more than one definition. A transaction name will be called *nested* if it appears in the left hand side of a composition constraint; otherwise it is called *elementary* transaction. We will assume that all the beginning and end transactions are elementary and that the beginning and end transaction of an elementary transaction is the transaction itself. For example, the following proposition defines a transaction to make flight reservations:

$$\begin{aligned}&Reservation(N, Ori, Des) \textbf{ consists of}\\ &\qquad \{Res_b(N), Leaving(Ori, Des, N), Return(Des, Ori, N), Res_e(N)\}.\end{aligned}$$

Effect propositions are expressions of the form:

$$T \textbf{ causes } F \textbf{ if } P_1, \ldots, P_n. \tag{2}$$

where T is an elementary transaction name, and the F and the P_i's ($n \geq 0$) are *fluent literals*. A fluent literal is an fluent name or an fluent name preceded by the symbol $\neg$. Intuitively, this proposition describes the effect of the elementary transaction T on F, given the preconditions $P_1, \ldots, P_n$. If $n = 0$ we will simple write:

$$T \textbf{ causes } F.$$

Continuing our flight reservation example, the following proposition describes one of the consequences of cancelling a reservation:

$$\begin{aligned}&Cancel(N) \textbf{ causes}\\ &\qquad \neg reserved\_seat(F, N, Seat) \textbf{ if } reserved\_seat(F, N, Seat).\end{aligned}$$

There are two kinds of constraint propositions:

$$\textbf{always } \varphi \textbf{ in } T. \tag{3}$$

$$\textbf{always } \varphi \textbf{ after } T. \tag{4}$$

where T is a transaction name and φ a formula formed with fluent names. The first kind of constraints says that φ must be true throughout the execution of T. The second class of constraints requires φ to be true after T. There are also two kinds of order propositions. They are expressions of the following form:

$$T_1 \textbf{ precedes } T_2. \tag{5}$$

$$T_1 \textbf{ justifies } T_2. \tag{6}$$

where T_1 and T_2 are transaction names. Informally, expressions of the form (5) say that transaction T_1 must be executed sometime before transaction T_2. Expressions of the form (6) say that the presence of T_1 in a sequence of transactions makes the later occurrence of the transaction T_2 possible. Although we do not impose restrictions on the relations created by these propositions they will usually relate transactions on the same level of the hierarchy.

Remarks: We have not specified the nature of the fluents in $\mathcal{T}$. Fluents are data items that can change their values in the database and they can take different forms according to the kind of database being used. For example, in a relational database, fluent names are tuples that can appear in a relation. In an object oriented database it could be the name of an object attribute together with a value in the domain of that attribute. Variables can replace the attribute's value in an fluent name, and they will represent parameters that can be replaced by any value in the domain of the attribute. To minimize the introduction of new notation we will present our examples loosely based on the relation model.

A *transaction system* is a finite set of composition, effect, constraint and order propositions.

Example 2.1 *The following is a partial description of a transaction system to make flight reservations.*

$$
\begin{array}{l}
Reservation(N, Ori, Des) \textbf{ consists of} \\
\qquad \{Res_b(N), Leaving(Ori, Des, N), Return(Des, Ori, N), Res_e(N)\}. \\
Leaving(O, D, N) \textbf{ consists of } \{Lv_b, assign\_flight(O, D, N), assign\_seat(N), Lv_e\}. \\
assign\_flight(O, D, N) \textbf{ precedes } assign\_seat(N). \\
assign\_flight(O, D, N) \textbf{ causes} \\
\qquad flight(F, O, D, Av - 1, Res \cup \{N\}) \textbf{ if } flight(F, O, D, Av, Res), Av > 0. \\
assign\_flight(O, D, N) \textbf{ causes} \\
\qquad \neg flight(F, O, D, Av, Res) \textbf{ if } flight(F, O, D, Av, Res), Av > 0. \\
\qquad \vdots
\end{array}
$$

Some observations about the example. The fluent *flight* has four arguments, the first argument is the flight number, the second and third argument are the origin and destination of the flight, the fourth is the number of available reservations in the flight and the last argument is the set of actual reservation that have been made on that flight.

There are two nested transactions, *Reservarions* and *Leaving*. Some other transactions may be nested and there are many effect propositions that have not been included in the example. Note that the departure reservations (i.e. *Leaving* transactions) can be done before or after the return reservations.

2.1 Semantics of $\mathcal{T}$

We will now proceed to formally describe the semantics of $\mathcal{T}$. Given a transaction system S, we will define its models and define when a transaction system entails a value proposition (defined later in the paper).

Elementary transactions are the basic components of a transaction system. It is the execution of a sequence of elementary transactions that ultimately describes the evolution of the database. Thus, we need to specify which sequence of elementary transactions are possible and the effect of each possible sequence in the states of the database. The elementary transactions will posses the standard transaction properties of atomicity and isolation. However, consistency will be maintained according to the constraint propositions defined in the transaction system.

We define a *history* as a sequence $[t_1, \ldots, t_n]$ $(n \geq 0)$ of elementary transaction names where a transaction name can appear at most once, and for any end transaction t_e that appears in the sequence the corresponding beginning transaction t_b must appear in the sequence before t_e (cf. semantic histories in [2]). A history is *complete* if for every beginning transaction in the history the corresponding end transaction is also part of the history.[2]

Definition 2.1 (Transaction System Graph and Transaction Graph) *For a transaction system S, the Transaction System Graph G_S is defined as follows:*

1. *Nodes of G_S are transactions of S, and*
2. *If "t* **consists of** *$\{t_0, \ldots, t_{n+1}\}$" is a composition proposition in S then G_S contains directed edges from t to each of the t_i's.*

For a transaction t in a transaction system S, the transaction graph of t with respect to S denoted by G_S^t is defined as the subgraph of G_S consisting of all nodes that are reachable from t, and all edges that can be traversed from t, in the graph G_S.

Note that by our definition G_S is a forest of trees and for any transaction t, G_S^t is a tree with t as the root.

Definition 2.2 *The set of* elementary members *of a transaction t denoted by M_t is defined as the set of leaf nodes in the tree G_S^t.*

The set of non-elementary members *of a transaction t denoted by N_t is defined as the set of non-leaf nodes in the tree G_S^t.*

[2]Observe that a complete history does not require that all sub-transactions of a nested transaction t initiated in the history appear in the history.

Definition 2.3 (Imprint of Transactions) *The* non-interleaving imprint *of an elementary transactions is the transaction itself.*

For a non-elementary transaction t a non-interleaving imprint of t *is defined as a complete history made up of an ordering of all the elements of M_t such that*
(i) it starts with t_b and ends with t_e and,
(ii) for any non-elementary transaction t' in N_t, an imprint of t' is a substring of the imprint of t.

Remarks: (1) Note that imprints are complete histories where every sub-transaction of a nested transaction initiated in the imprint is also part of the history. (2) The adjective *non-interleaving* is added to emphasize the fact that there is no interleaving among the sub-transactions.

A (database) *state* is a set of fluents. A fluent f is true ($\neg f$ is false) in a state σ if $f \in \sigma$ ($f \notin \sigma$); otherwise is false (true).

We are now ready to define the models of a transaction system. The models have two components; a transaction imprint mapping Σ which maps non-elementary transactions to its imprints; and a *history interpretation* Φ, which is a partial function from histories to states such that:

1. The empty history [] belongs to the domain of Φ, and
2. Φ is prefix-closed, i.e. for any history τ and any transaction t, if the concatenation of τ and t, denoted by $\tau \circ t$, is in the domain of Φ then τ must also be in the domain of Φ.

2.1.1 Interpreting transaction histories

We say that history τ is *possible* in Φ if $\Phi(\tau)$ is defined. For a transaction t_i ($1 \leq i \leq n$) in a history $\tau = [t_1, \ldots, t_n]$ we refer to the prefix $[t_1, \ldots, t_{i-1}]$ by $\tau_{|t_i}$.

The effects of the execution of elementary transaction in database state are derived from the effect propositions. If the preconditions of the effect proposition "t **causes** f **if** $p_1, \ldots, p_n$" for an elementary condition t holds in a state of the database then the effect is that f holds in the resulting state. If there is no effect proposition for t whose precondition holds in a state σ then the transaction t is not executable in that state.

Formally, for a transaction system S, a fluent literal f is an *effect* of an elementary transaction t in a state σ if there exists an effect proposition of the form "t **causes** f **if** $p_1, \ldots, p_n$" in S whose preconditions $p_1, \ldots, p_n$ hold in σ. For every elementary transaction t and a possible history τ in Φ, let us define

$E_t^+(\Phi(\tau)) = \{f : f$ is a fluent name and f is an effect of t in $\Phi(\tau)\}$, and

$E_t^-(\Phi(\tau)) = \{f : f$ is a fluent name and $\neg f$ is an effect of t in $\Phi(\tau)\}$,

For any transaction t and a possible history τ in Φ we define

$$Res(t, \tau) = \Phi(\tau) \cup E_t^+(\Phi(\tau)) \setminus E_t^-(\Phi(\tau)).$$

Res is referred to as the *transition function.*

Definition 2.4 *A history interpretation* Φ satisfies *the effect propositions in a transaction system* S *if for any history* $\tau \circ t$ *from the language of* S

$$\Phi(\tau \circ t) = Res(t, \Phi(\tau)) \text{ if } E_t^+(\Phi(\tau)) \cap E_t^-(\Phi(\tau)) = \emptyset \text{ and } E_t^+(\Phi(\tau)) \cup E_t^-(\Phi(\tau)) \neq \emptyset$$

and undefined otherwise. We say Φ *is an* effect model *of* S *if it satisfies the effect propositions of* S.

2.1.2 Interpreting Transaction Compositions

Definition 2.5 *We say* Σ *satisfies the composition and order propositions in a transaction system* S *if the following conditions hold.*

1. *For all transactions* t *in* S, $\Sigma(t)$ *is an imprint of* t.
2. *If* $\Sigma(t) = t_b, \ldots, t'_b, t_1, \ldots, t_n, t'_e, \ldots, t_e$, *then* $\Sigma(t') = t'_b, t_1, \ldots, t_n, t'_e$.
3. *If there exists an order proposition of the form "*t_1 **precedes** t_2*", then for any* t *if* t_1 *and* t_2 *belong to* N_t, *then* t_{1e} *occurs earlier than* t_{2b} *in* $\Sigma(t)$.
4. *If there are order propositions of the form "*T **justifies** t_2*" such that* T *and* t_2 *belong to* N_t *for a transaction* t, *then either* T_e *occurs earlier than* t_{2_b} *in* $\Sigma(t)$ *or there exists another order proposition "*T' **justifies** t_2*" such that* T' *and* t_2 *belong to* N_t *for a transaction* t *and* T'_e *occurs earlier than* t_{2_b}.

Definition 2.6 (Σ-correct histories) *A history is said to be* Σ*-correct if it is a prefix of* $\Sigma(t_1) \circ \ldots \circ \Sigma(t_n)$, *for a set of transactions* $\{t_1, \ldots, t_n\}$ *in the transaction system.*

We are now ready to define models of a transaction system S.

Definition 2.7 *Given a transaction system* S, *we say the structure* (Σ, Φ), *where* Σ *is a transaction imprint mapping and* Φ *is a history interpretation, is a model of* S *if the following conditions are satisfied.*

- Φ *satisfies the effect propositions in* S.

- *Σ satisfies the composition and order propositions in S.*
- *If S contains a constraint proposition of the form "* **always** φ **in** t*" then for any Σ-correct history τ if t_b appears in τ and t_e does not then φ is true in $\Phi(\tau)$.*
- *If S contains a constraint proposition of the form "* **always** φ **after** t*" then for any Σ-correct history τ if t_e is the last transaction in τ then φ is true in $\Phi(\tau)$.*

Recall that the main goal of the work described in this paper is to develop a language that will permit the automatic or semi-automatic verification of the correct concurrent execution of these inter-related elementary transactions. We will use value propositions for this purpose. *Value* propositions are expressions of the form

$$f \textbf{ after } t_1, \ldots, t_n. \tag{7}$$

where f is a fluent literal and t_i's are transactions. If all the t_i's are elementary transaction we will refer to the proposition as an *elementary value* proposition. We say the above proposition is true w.r.t. a model (Σ, Φ) of a transaction system iff f is true in $\Phi(\tau)$, where $\tau = \Sigma(t_1), \ldots, \Sigma(t_n)$.

We say a transaction system S entails a value proposition q (i.e. $S \models q$) iff q is true in all models of S.

Thus, the execution of a sequence of transactions $[t_1, \ldots, t_n]$ will comply with a transaction system S if "$S \models true$ **after** $t_1, \ldots, t_n$." When $f = true$ in (7), we will write:

$$\textbf{executable } t_1, \ldots, t_n. \tag{8}$$

So far, a transaction system does not impose any conditions in the initial state of the database. In real situations the initial state of the database is known and the question is if a particular sequence of transactions is executable starting from the initial state. Value propositions can be used to specified the initial state of the database if $n = 0$ in (7). In this case we will write:

$$\textbf{initially } f. \tag{9}$$

An initialization set is a finite set of propositions of the form (9). For a transition system S and an initialization set I, we say that (Σ, Φ) is a model of $S \cup I$ iff (Σ, Φ) is a model of S and for each value proposition of the form (9) in I, f is true in $\Phi([])$. Similar to non initialized transaction systems we say that $S \cup I$ entails a value proposition q iff q is true in all models of $S \cup I$.

Definition 2.8 (Categorical) *A transition system S and an initialization set I are said to be categorical if $S \cup I$ has a unique model.*

Definition 2.9 (F-Categorical) *A transition system S and an initialization set I are said to be F-categorical with respect to a set of fluents F, if for any f in F, and any sequence of transactions $T_1, \ldots, T_n$ in S, either $S \cup I \models f$ **after** $T_1, \ldots, T_n$ or $S \cup I \models \neg f$ **after** $T_1, \ldots, T_n$.*

3 $\mathcal{T}_C$ – Extending $\mathcal{T}$ to incorporate concurrency

In this section we discuss concurrent execution of transactions. For that purpose we extend $\mathcal{T}$ to allow the specification of semantic constraints about concurrent execution of transactions. We denote the concurrent execution of a set of transactions $\{T_1, \ldots, T_n\}$ by $\{T_1 | \ldots | T_n\}$.

A concurrent transaction systems is a transaction system augmented with ordering constraints of the following form:

$$T_1 \textbf{ precedes } T_2. \tag{10}$$

$$T_1 \textbf{ justifies } T_2. \tag{11}$$

$$\textbf{atomic } T_1. \tag{12}$$

$$T_1 \textbf{ preempts } T_2. \tag{13}$$

The intuitive meaning of (10) and (11) is as before. The intuitive meaning of (12) is that no other transaction can interleave with T_1. The intuitive meaning of (13) is that during the interleaving of two transactions, T_2 can not execute after T_1. We will require **preempts** to be antisymmetric.

Consider Example 2.1. Let us consider another transaction *Cancel* necessary for canceling reservations. We now give the effect propositions corresponding to this transaction and also give ordering constraints to aid in the concurrent execution of the transactions *Cancel* and *Reservations*.

$$\begin{aligned} &Cancel(N) \textbf{ causes} \\ &\qquad flight(F, O, D, Av + 1, Res \setminus \{N\}) \textbf{ if } flight(F, O, D, Av, Res), N \in Res. \end{aligned}$$

$$\begin{aligned} &Cancel(N) \textbf{ causes} \\ &\qquad \neg flight(F, O, D, Av, Res) \textbf{ if } flight(F, O, D, Av, Res), N \in Res. \end{aligned}$$

$$\begin{aligned} &Cancel(N) \textbf{ causes} \\ &\qquad \neg reserved\_seat(F, N, Seat) \textbf{ if } reserved\_seat(F, N, Seat). \end{aligned}$$

$Leaving(Ori, Des, N)$ **justifies** $Cancel(N)$.
$Return(Des, Ori, N)$ **justifies** $Cancel(N)$.

$Cancel(N)$ **preempts** $Leaving(Ori, Des, N)$.
$Cancel(N)$ **preempts** $Return(Des, Ori, N)$.

$assign\_flight(O, D, N)$ **justifies** $Cancel(N)$.
$assig\_flight(O, D, N)$ **precedes** $assign\_seat(N)$.

The fluent $reserved\_seat$ has three parameters, the flight number, the reservation number and the seat. Observe that the cancelation of a reservation can be made at any time after the reservation process has started, even in the middle of making the departure arrangements. The effects of a cancelation depends on when the cancelation occurs. This is captured by having several effect propositions for $Cancel$. Now, making a reservation not necessarily means that we need to execute all the sub-transactions Res_b, $Leaving$, $Return$, and Res_e of $Reservation$. If in the middle of a reservation $Cancel$ is executed, $Reservation$ can terminate correctly without executing all its sub-transactions. Note that similar mechanisms can be exploited to recover from crashes or failures. This feature is important since rollbacks are not desirable in long-lived transaction systems.

3.1 Semantics of $\mathcal{T}_C$

The semantics of concurrent transaction systems is very similar to the semantics of transaction systems and builds up on it. As before models of a concurrent transaction system have two components: the history interpretation Φ and the transaction imprint mapping Σ. The main difference is that Σ now also maps concurrent transactions. To make our definitions precise we first need to define the *imprint of a concurrent transaction*.

As observed above, one important aspect is that an imprint of a concurrent transaction $\{T_1|T_2\}$ may not necessarily be an interleaving of an imprint of T_1 and an imprint of T_2. Consider the transactions $Reservation$ and $Cancel$ from example 2.1. We allow $[Res_b, Leaving, Cancell, Res_e]$ to be an imprint of $\{Reservation|Cancel\}$, even though it is not an interleaving of an imprint of $Reservation$ and an imprint of $Cancel$. The imprint of a concurrent transaction is formally defined as follows:

Definition 3.1 *An imprint τ of a concurrent transaction $T^c = \{T_1|\ldots|T_n\}$ is an interleaving of $\tau_1, \ldots, \tau_n$, where τ_i is a complete history of transactions in M_{T_i}.*

N_{T^c} and M_{T^c} are defined to be $N_{T_1} \cup \ldots \cup N_{T_n}$ and $M_{T_1} \cup \ldots \cup M_{T_n}$ respectively.

The fundamental modification of the original definitions of imprints is that not all the elementary transactions of each T_i appears in the imprint of T^c. Also, we now allow concurrent execution of subtransactions of each ofthe T_i's.

Definition 3.2 *We say Σ satisfies the composition and order propositions in a concurrent transaction system S if the following conditions hold.*

1. *For all transactions t in S, $\Sigma(t)$ is an imprint of t.*
2. *For all non-concurrent transactions t in S, $\Sigma(t) = t_b, \ldots, t'_b, t_1, \ldots, t_n, t'_e, \ldots, t_e$, then $\Sigma(t') = t'_b, t_1, \ldots, t_n, t'_e$.*
3. *If there exists an order proposition of the form "t_1 **precedes** t_2", then for any t if t_1 and t_2 belong to $N_t \cup M_t$, then t_{1e} occurs earlier than t_{2b} in $\Sigma(t)$. (Recall that for an elementary transaction t, $t = t_b = t_e$.)*
4. *If there are order propositions of the form "T **justifies** t_2" such that T and t_2 belong to N_t for a transaction t, then either T_e occurs earlier than t_{2_b} in $\Sigma(t)$, or there exists another order proposition "T' **justifies** t_2" such that T' and t_2 belong to N_t for a transaction t and T'_e occurs earlier than t_{2_b}.*
5. *If there exists an order proposition of the form "**atomic** t_1", then $\Sigma(t_1)$ is a substring of $\Sigma(t)$. (Note that sub-transactions in t_1 can interleave.)*
6. *If there exists an order proposition of the form "t_1 **preempts** t_2", then for any t if t_1 and t_2 belong to $N_t \cup M_t$, then in $\Sigma(t)$, no element of M_{t_2} occurs after t_{1e}.*

The definition of Σ-correct histories and models of concurrent transaction systems are the same as before except that they use Definition 3.2 instead of Definition 2.5.

For querying, the syntax of value-propositions are same as before (i.e., as in (7)) except that the t_i's can now be concurrent transaction. The definition of entailment also remains unchanged.

There are many situations which can be expressed using in $\mathcal{T}_C$. $\mathcal{T}_C$ can be used to study standard transaction decompositions as the ones suggested in [2]. In this particular case there will not be **preempts** and **atomic** in the order propositions and **precedes** proposition will be used to ensure that every valid imprint of a concurrent transaction $\{T_1|\ldots|T_n\}$ is the interleaving of imprints of each T_i. Now many properties reduce to the entailment of particular value propositions. For example, that a history $[t_1, \ldots, t_n]$ creates a deadlock can be shown by proving that there does not exist an elementary transaction t such that:

$$S \cup I \models \textbf{executable } t_1, \ldots, t_n, t.$$

A standard technique to implement concurrency control mechanisms that incorporate semantic serializability is to extend algorithms such two-phase locking with *valid successor sets* (i.e. the set of elementary transactions that can follow another transaction t after a decomposition has been done (see [2, 5] for more details)). Valid successor sets can be incrementally generated by first showing for which elementary transactions t_0 it holds that:

$$S \cup I \models \textbf{executable}\ \ t_0.$$

and then for each history $[t_0, \ldots, t_n]$ such that for each i $(0 \leq i \leq n-1)$, t_{i+1} is in the successor set of t_i, we can compute the successor set of t_n by finding the elementary transactions t such that:[3]

$$S \cup I \models \textbf{executable}\ \ t_1, \ldots, t_n, t.$$

We could also use $\mathcal{T}_C$ to capture the transaction model described in [1]. We can verify the consistency of histories or assist proving that a history is orderable by providing valid successor sets.

We can also capture the standard definition of serializability using the entailment relation.

Definition 3.3 *Let S be a concurrent transaction system and $h = [t_1, \ldots, t_m]$ an imprint of the concurrent transaction $T^C = \{T_1 | \ldots | T_n\}$. The imprint h is serializable if there exists a non-interleaving imprint $[t'_1, \ldots, t'_m]$ of $\{T_1, \ldots, T_n\}$ such that for all I and for every fluent f:*

$S \cup I \models f$ **after** $t_1, \ldots, t_m$ *iff* $S \cup I \models f$ **after** $t'_1, \ldots, t'_m$.

We can also define very fine notions of serializability with respect to concurrent transaction systems. We can select parts of the database where we can verify serializability and ignore the serializability with respect to other parts of the database. This is captured by F-serializable specifications.

Definition 3.4 *A concurrent transaction system S is said to contain an F-*serializable *specification of the concurrent transaction $T^C = \{T_1 | \ldots | T_n\}$ if there exist an ordering $T_{i_1}, \ldots, T_{i_n}$ of the elements of T^C such that for all I, and for all $f \in F$:*

$S \cup I \models f$ **after** T_C *iff* $S \cup I \models f$ **after** $T_{i_1}, \ldots, T_{i_n}$.

If F is the set of all fluents, we say S contains a strong *serializable specification of T^C.*

Notice that these serializability criteria apply only to imprints that are permutations of non-interleaving imprints. Take, for example, our reservation transaction system. An imprint of the concurrent transaction $\{Reservation, Cancel\}$ is

[3] For this computation of the valid successors set, it is assumed that the transaction system is deadlock free, however, a better algorithm could be written to interleave the two process.

$[Res_b, Lv_b, assign\_flight, Cancel, Lv_e, Res_e]$. A possible ordering for the the serialization would be $Reservation, Cancel$, with imprint $[Res_b, Lv_b, assign\_flight, Lv_e, Res_e, Cancel]$. However, in normal circumstances a $Reservation$ transaction cannot be ended without completing the round-trip reservation.[4] Hence, our transaction system should disallow the imprint $[Res_b, Lv_b, assign\_flight, Lv_e, Res_e, Cancel]$ through order propositions and the correctness of the imprint will be determined semantically.

4 Final remarks

We have presented in this paper a specification language to reason about long duration transactions and their concurrent execution. We have called this language $\mathcal{T}_C$. The semantics of $\mathcal{T}_C$ is based on the automata-based semantics of action description languages, and allows us to define an entailment relation between a transaction description and queries that specify if a fluent is true after the execution of a sequence of transactions. This entailment relation forms the basis of our formalization of notions such as the correctness of a transaction definition, serializability of a schedule etc. Although we have no versioning in $\mathcal{T}_C$ like in NT/PV [7], we can define nested transactions and use explicit predicates to improve concurrency. Furthermore we can express more sophisticated conditions (than found in the literature [5, 4, 1, 7, 3, 2]) about interleaving. This new interleaving provides room for additional concurrency concurrency and provides ways to naturally specify compensation transactions for failures.

We have efficient algorithms (through translations to logic programs similar to the ones in [6]) to compute the entailment of elementary value propositions for categorical systems. For non categorical systems we also have algorithms but they involve the power of a full theorem prover. We are currently isolating cases where more efficient proofs are possible. We are also investigating algorithms to compute the entailment of more general value propositions. However, the elementary propositions are of special interest since the entailment procedure for a particular transaction system will lead us to the automatic generation of an automaton that will act as the transaction manager for the system.

Another future direction of our research involves carefully extending the language to incorporate additional features. For example, we are considering to allow concurrent transactions in the right hand side of composition propositions. Also, we are considering order propositions of the form T_1 **must follow** T_2. Such order propositions seem to be necessary to capture triggering of one transaction by another in active database systems. The Definition 2.5 will then be extended by the condition "If there exists an order proposition of the form "t_2 **must follow** t_1", then for any t if t_1 and t_2 belong to N_t, then $\Sigma(t_1) \circ \Sigma(t_2)$ is a substring of $\Sigma(t)$." Finaly, our use of a semantic structure similar to the ones used in action-description languages in AI will make it easier to incorporate necessary features of those languages to our language.

[4] For the sake of simplicity we are assuming that all reservations are for round-trip tickets.

References

[1] D. Agrawal, A.E. Abbadi, and A.K. Singh. Consistency and orderability: semantics-based correctness criteria for databases. *ACM Transactions on Database Systems*, 18(3):460–486, 1993.

[2] P. Ammann, S. Jajodai, and I. Ray. Using formal methods to reason about semantics-based decompositions of transactions. In *Proc. of VLDB*, pages 218–227, 1995.

[3] P. Chrysanthis and K. Ramamritham. Synthesis of extended transaction models using ACTA. *ACM, TODS*, 19(3):450–491, 1994.

[4] A. Farrag and M. Ozsu. Using semantic knowledge of transactions to increase concurrency. *ACM, TODS*, 14(4):503–525, 1989.

[5] H. Garcia-Molina. Using semantic knowledge for transaction processing in a distributed database. *ACM, TODS*, 8(2):186–213, 1983.

[6] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–321, 1993.

[7] H. Korth and G. Speegle. Formal aspects of concurrency control in long duration transaction systems using the NT/PV model. *ACM, TODS*, 19(3):492–535, 1994.