A Linear Temporal Logic Approach to Objects with Transactions

Grit Denker[1], Jaime Ramos[2], Carlos Caleiro[2], and Amílcar Sernadas[2]

[1] Technische Universität Braunschweig, Informatik, Abt. Datenbanken
Postfach 3329, D–38023 Braunschweig, Germany
e–mail: G.Denker@tu-bs.de
[2] Instituto Superior Técnico, Departamento de Matemática
Av. Rovisco Pais, P-1096 Lisboa Codex, Portugal
e–mail: {jabr|ccal|acs}@math.ist.utl.pt

Abstract. Our concern is the high level specification of reactive software systems such as information systems. We adopt an object oriented, temporal logic based approach to specification. The notion of transaction incorporates various application domains, for instance transactions as abstractions from processes as known from refinement theory, transactions as abstractions from business processes as known in business process modelling or database transactions. In this paper we investigate object specifications with transactions. We illustrate the use of transactions by examples given in an object oriented style and introduce a linear temporal logic with transactions (TOSL) which serves as denotional model for such object specifications with transactions. We explain how TOSL is semantically defined in terms of life cycles and illustrate by example the translation of object specifications to TOSL. Using TOSL for system specification results in sets of formulae which are independent from the level of granularity.

Keywords: linear temporal logic, formal specification language, object orientation, specification method, refinement, transaction, reactive system

1 Introduction and related work

Applications of temporal logics can be found in various independent problem domains. Due to its natural expressiveness of dynamics it is especially suitable as a semantic domain for object oriented approaches. Objects are understood as units of structure and behaviour. Denotation of behaviour can be given in terms of temporal logics. We adopt a formal, object oriented, logic based approach to reactive system specification such as information systems. Information systems are understood as reactive systems with underlying databases and application programs for storage, manipulation, and retrieval of data. Object oriented modelling techniques are accepted as a promising approach to software design. This is the reason for new languages, methods and models emerging. The various existing design notations range from informal analysis methods to formal specification languages. Typical design issues include specification, verification, testing, refinement, etc. One of the requirements to handle these issues is formality. There are formal specification languages defined on top of different frameworks, including algebraic and process models, equational and temporal logics, and others. The specification languages Troll [6, 12] and Gnome [22] are being developed at TU Braunschweig and IST Lisbon, respectively. Both languages are supposed to be used in early phases of the design and modelling of information system. Theoretical investigations accompanied the design during the whole time. Object Specification Logic (OSL [23, 24]) is the semantical basis for Gnome. It is a linear time temporal logic. Recently, Troll and its underlying semantics are focussed towards specifying distributed systems. An appropriate Distributed Temporal Logic (Dtl [8, 7] based on n-agent logic [18]) has been developed.

In this paper we investigate the concept of transaction in object oriented specifications. We understand transactions as complex processes composed of smaller units, i.e., actions, by using operators like sequence, choice, and loop. The notion of transaction is not tied to a specific task. Firstly, transactions come along with refinement. Refinement is a technique often applied in the design process. A system is described as a sequence of specifications on different levels of abstraction, where each specification is implemented by its successor. Notions of refinement for structured programming, abstract data types, and process theory are well known. The object oriented framework puts additional complexity to this technique. In this case, refinement includes data refinement (structural refinement) as well as action refinement (behavioural refinement). Action refinement is understood as the principle of implementing abstract actions through complex processes using operators like sequence, choice, and loop. Such processes can be regarded as transactions. Secondly, the notion of transaction is supported by many database management systems. We aim at modelling information systems with object oriented techniques. Thus, the concept of transaction known from databases is important for our work. A database transaction follows the ACID-principle. It is executed atomically from the users viewpoint (Atomicity), it transforms the database from one consistent state to another consistent state (Consistency), its effect is the same as it would have taken place in isolation (Isolation), and its effect is persistent in the database (Durability). Thirdly, business processes may be understood as transactions. Significant behavioural parts of an enterprise can be described as business processes. The behaviour of such processes is known a priori. That is the reason, why such processes are already specifiable in the design phase. A business process uses several resources and puts together different services to construct a new service. An appropriate concept of transaction could be used to give semantics to business processes.

The focus of this paper is to deal with transactions in an object oriented, temporal logic based setting. Work done in the field of temporal logics about refinement and change of granularity ([14, 1, 9, 10, 17]) is related to our work. One of the main problems in this area can be described as follows: temporal operators such as `Y` (yesterday,previous) and `X` (tomorrow,next) in a temporal formula are strongly connected to the underlying granularity. Thus, if the execution of an action takes one step, then `Y` refers to the state before the action execution, but if the execution takes several steps (as for a transaction), then `Y` refers to the penultimate state which is usually not the state before the transaction began. Thus, a formula like "Before the action occurred φ hold true" will change its meaning if the action is refined to a transaction. There are different solutions to this problem. Either the use of this operators together with refinement is forbidden (see eg TLA [15]), or others propose to specify different formulae for the different abstraction levels (eg [9]). Another solution is, to introduce different local clocks and associate formulae with these clocks (eg [17]). This way temporal operators are bound to different granularities and, thus, a next operator can refer to different states depending on the formula it belongs to. There has been done also work about refinement in the area of formal specification and object orientation (eg [2, 16, 9, 3, 25] to mention a few, most of which are based on algebraic techniques). The differences to our work are that either they do not work with temporal logics (eg [2, 3, 25]) or for those who use temporal logics [16, 9] they use one of the approaches mentioned above.

Our approach is different and innovative in the following sense. We aim to write only formulae which are independent of the level of granularity. That means, we do not want to change the formula because the granularity has changed, rather we would like to write formulae in such a way that possible changes of granularity are already taken into account. Thus, the formulae we intend to reach are those which are independent from the level of abstraction or granularity. The reason why this is possible is the following: The logic is used as an underlying semantics for object oriented specifications. Therefore, formulae are not just freely built, rather than they are derived by translating concepts of the specification language into logical expressions. Thus, if we find an appropriate, granularity-independent translation of the language constructs to temporal formulae, we succeeded. This is exactly what we are going to do in the rest of the paper. We investigate the main concepts needed in object oriented specification (such as communication, effects and preconditions of actions) and show how they are translated appropriately to temporal formulae which are preserved under refinement, i.e., which are granularity-independent. The logic we use for this purpose is a linear temporal logic which incoporates the concept of transaction, so-called Transactional Object Specification Logic (TOSL). This logic is the basis for formalizing dynamics. Together with an algebraic part which serves as semantics for structural aspects this constitutes a firm mathematical framework to give semantics to object specifications including transactions.

The rest of the paper is organized as follows. Section 2 illustrates the main problems when temporal logic is combined with different levels of granularity. We explain the problems by means of examples and propose a first solution. In section 3 we introduce TOSLand define its semantics. This establishes the basis for dealing with the problems as illustrated before. Thus, in section 4 we use TOSL as a tool and formalize our first solution. We translate object specifications with transactions into TOSL formulae and show that we yield formulae which are compatible with refinement. Finally, we give a short summary and point out future work.

2 The problem: temporal logic and levels of granularity

We demonstrate basic specification features using notations in the spirit of Troll or Gnome. Especially for transactional parts we use an ad-hoc notation influenced by the intended semantics. The example does not exploit all features of object oriented specification (eg inheritance, etc.), rather we emphasize on the transactional aspect. But the theoretical framework is capable of dealing with fully object oriented models.

Example 1 (Casino – Abstract level). We specify part of a casino world in an object oriented way (cf. [11]). We start with an informal description of the Universe of Discourse (UoD) which has to be modelled.

Slot machines are very famous games in casinos. Players put money in the machine, pull the handle and wait for the reaction, i.e., winning money or not. Playing the machine implies that the pot of the slot machine is increased and a decision is made whether money is paid out to the player.

A first object oriented approach to model this may result in two types of objects: `player` and `slot_machine`.

```
object class player
  actions play(nat)
  behavior play(n)  synchronizes with move(n)
end
```

A player can play the slot machine. Whenever a player plays, he/she synchronises with the slot machine, i.e., `play` *synchronizes with* `move`. Thus, the behavior of a player is such that he/she synchronously communicates with the machine when playing it. `play` has a parameter which will be set by the action `move` of slot machine. Randomly the slot machine assigns a value to the parameter. A very rough specification of slot machine is:

```
object class slot_machine
  attributes pot:nat initialized 0
             lucky:bool
  actions move(n)
  behavior move(0) onlyIf  not lucky do pot=pot+1 od
           move(n) (n#0) onlyIf (lucky and pot=n-1) do pot=0 od
end
```

The slot machine has a `pot` as attribute to store the current amount of money. Attribute `lucky` presents the randomized selection whether money is paid off or not. The action move `move` will occurr with the parameter zero only if the player is not lucky (*onlyIf* clause of `move(0)`). In this case it will increase the pot (*do-od* clause of `move(0)`). The player gets no money because the synchronisation is done with parameter zero. The slot machine performs a move with a parameter `n` different from zero only if the player is lucky and the pot already stored the amount `n-1` of money (*onlyIf* clause of `move(n)`). Thus, by putting the coin, the pot is increased, all the money is given to the player, and the pot is afterwards set to zero (*do-od* clause of `move(0)`).

The given example illustrates the following concepts of object oriented approaches: Objects are units of structure (eg attributes) and behavior (eg actions plus behavior clauses). They are encapsulated in the sense that they change their state (represented by attributes) only through action execution. Objects communicate with each other by synchronous action calling.

Object specifications such as the one give above can be formalized as a pair $Spec = (\Sigma, \Phi)$, where Σ incorporates structural parts such as identities, attribute and action symbols, among others, and Φ is a set of temporal formulae derived from the specification. We adopt an algebraic approach for the structural part (for details see [8]). Concerning the dynamics each language concept is translated into a temporal formula. Thus, axioms about occurrences of actions, their effects on attributes and interactions between actions, respectively, are given.

Example 2 (Casino – Abstract level – Temporal formulae). The following natural language expressions about the casino behaviour can be derived from the specification: "`play(n)` synchronizes with `move(n)`, i.e., whenever `play(n)` happens also `move(n)` happens", and "moving the slotmachine increments the pot and depending on the value of lucky money is paid out or not", etc.

There are different possibilities to express such statements in logic: eg dynamic logic [20, 13], situation calculus [19, 21], among others. On the basis of temporal logic we introduced in former papers [23, 8] a special predicate on actions $\odot\alpha$ (occurrence of action α). $\odot\alpha$ is true in the state where α has just been executed. Thus, the given player specification could be translated to

$$\odot\mathtt{play(n)} \Rightarrow \odot\mathtt{move(n)}. \tag{1}$$

I.e., `play(n)` can only happen with `move(n)` happening synchronously. Analogously, the behaviour of slot machine could be formalized as

$$[\mathsf{Y}(\mathtt{lucky} = \mathtt{false} \wedge \mathtt{pot} = \mathtt{n}) \wedge \odot\mathtt{move(k)}] \Rightarrow (\mathtt{k} = \mathtt{0} \wedge \mathtt{pot} = \mathtt{n} + \mathtt{1}) \tag{2}$$
$$[\mathsf{Y}(\mathtt{lucky} = \mathtt{true}) \wedge \odot\mathtt{move(n)}] \Rightarrow \mathtt{pot} = \mathtt{0} \tag{3}$$

The first axiom says that if lucky does not hold then moving the machine will increase the pot and no money will be paid out (`k=0`). Y stands for the temporal logic operator *yesterday*. The second axiom states that after moving the slot machine from a state where `lucky` held implies that all money is paid out, i.e., `pot=0`.

What we have done so far is formalizing the semantics of an object specification by translation to temporal logic formulae. The former specification was a first rough approach to a formalization of the given UoD. To get a more concrete, closer to implementation description one might apply refinement techniques.

Example 3 (Casino – Refined level). The action `move` is refined to a transaction `MOVE` composed of two steps. First, the pot is incremented. The next step in the transaction depends on the value of attribute `lucky`. Either all the money is paid off (*then*-case) or nothing is paid (*else*-case).

```
object class slot_machine
  attributes pot:nat initialized 0
             lucky:bool
  actions inc
          payoff(nat)
          zeropay(nat)
  transactions MOVE(nat)
  behavior MOVE(n)=<inc;if(lucky,payoff(n),zeropay(n))>
           inc do pot=pot+1 od
           payoff(n) onlyIf pot=n do pot=0  od
           zeropay(n) onlyIf  n=0
end
```

`MOVE` is specified as a transaction composed of several actions. The first step increments the pot. The effect of the `inc` action is specified in a *do-od* clause. The second step deals with the paying off. The amount `n` can only be paid off (`payoff(n)`) if this amount has been stored in the pot before. Afterwards the pot will be reset to zero. The formalization is given in the behavior clause of `payoff(n)`. A zeropay can only happen with the parameter zero.

Now we have a second specification which intuitively does the same as the previous. We also put some effort in formalizing the behavior of the abstract specification as temporal formulae. In real-size case studies the set of formulae increases rapidly. Thus, instead of re-doing the formalization one would wish to reuse the set of temporal formulae by just substituting transactions for the actions. But usually this does not yield the intended result.

Example 4 (Casino – Refined level – Temporal formulae). The temporal formulae given in example 2 are no longer meaningful in the presence of transactions. A substitution of the transaction `MOVE(n)` for the action `move(n)` yields the following formulae:

$$\odot\mathtt{play(n)} \Rightarrow \odot\mathtt{MOVE(n)}, \tag{4}$$

$$[\mathsf{Y}(\mathtt{lucky} = \mathtt{false} \wedge \mathtt{pot} = \mathtt{n}) \wedge \odot\mathtt{MOVE(k)}] \Rightarrow (\mathtt{k} = 0 \wedge \mathtt{pot} = \mathtt{n} + 1), \tag{5}$$

$$[\mathsf{Y}(\mathtt{lucky} = \mathtt{true}) \wedge \odot\mathtt{MOVE(n)}] \Rightarrow \mathtt{pot} = 0. \tag{6}$$

The following problem arise:

Synchronisation: The first problem concerns the interaction between `player` and `slot_machine`. What does it intuitively mean, that an action (`play`) and a transaction (`MOVE`) occur together? Does it mean that the action `play(n)` occurs simultaneously with the first (or the last) action of the transaction `MOVE(n)`? It seems that the translation of interaction statements to an implication between action occurrences, as done in equation (1), is too strict.

At least, it could be more appropriate to give another formalisation of interaction which will also be meaningful in the presence of transactions. For instance, synchronisation of two transactions is formalized as "they overlap during their execution".

Effects and preconditions of actions: The temporal operator Y always refers to the preceding state. But the expression "preceding state" has different meanings depending on the level of granularity. On the abstract level in equation (3) the temporal operator Y refers to the state *before the execution* of `move(n)`. This is what one expects to test a precondition. Whereas in (6) the temporal operator refers to the last state before the execution of `MOVE(n)` is finished. Intuitively, $\odot$`MOVE(n)` holds in a state where all actions belonging to the transaction have been performed one after each other. Thus, in (6) Y does not refer to the state before the execution started, rather than the state where `inc` has happened. Thus, temporal operators have to be used carefully when different levels of granularity are involved as it is the case of object specifications with transactions.
Similar investigations show that the formula for the effect of `MOVE(n)` (see (5)) is not appropriate.

Re-investigating the formulae of our example yields in the following more appropriate expressions: "There is an interaction between `play(n)` and `MOVE(n)`", "if `lucky=false` and `pot=n` in the state just before the execution of `MOVE(n)` started, and if `MOVE(n)` has finished then `pot=n+1`", and "if `lucky=true` in the state before the execution of `MOVE(n)` started, then `pot=0` holds in the state where `MOVE(n)` has just finished". Thus, an appropriate logic must be able to specify these subtle kind of expressions referring to states before and after the execution of transactions (for preconditions and effects) as well as it must be capable of specifying when two transactions interact, i.e., when they overlapp.

The conclusions we draw from these investigations are the following:

1. The formalization of system requirements is a time-consuming task. Therefore, we can only increase productivity by using refinement technique if that what has been specified once on a specific level of abstraction remain under refinement.
2. The formalization of object oriented system specifications is done by translation to temporal formulae. Therefore, we have to translate object oriented features in such a way that the resulting formulae are independent of the level of granularity.
3. An appropriate logic must be able to express statements about overlapping of transactions (for modelling synchronisation) as well as it must be able to refer to the beginning and the end of transactions to get the right states for expressing preconditions and effects.

Therefore, in the following we define TOSL a linear temporal logic enriched by the notion of transaction, in which such statements are expressible. Its semantics is based on life cycles, i.e., sequences of action occurrences. In section 4 we illustrate the use of TOSL by formalizing the casino example.

3 The logic: TOSL

TOSL is a linear temporal logic with added features for specifying transactions. As usual for defining a logic we first have to define what are the signatures from which one can built terms, and then how formulae are built. Since we are going to use TOSL as denotational semantics for object specifications, we need to talk about data signatures (eg for the domains of attributes), as well as object signatures (i.e., attribute, action, and transaction symbols of objects), and signatures of object communities. Every action can be understood as a transaction, i.e., that one which consists only out of one action, but not vice versa. Therefore, one may think that transactions are sufficient. But to define semantics of TOSL we will associate sequences of actions to objects. Therefore, we still have to distinguish between actions and transactions in our approach. Due to space limitations we will not go into all details concerning data-, object-, and object system signatures, terms, etc. Rather we will focus on the those points in which we differ from the classical stuff and which are innovative in our approch. The formal definitions for signatures, terms, etc. are straightforward once the main ideas have been made clear. Moreover, we want to emphasize that the way of specifying changes when transactions come into play. Thus, we decided to spend some space on illustrating specification with transaction by means of examples in section 4.

Let $\langle ACT, TRAC, ATT\rangle$ be an **object signature** which consists of sets of action symbols, transaction symbols and attribute symbols as defined in the specification. Thus, an **object community signature** is a pair $\Sigma = \langle ID, \{\Sigma^i\}_{id \in ID}\rangle$ where ID is a finite set of object identifiers and each Σ^i is an object signature. Because of space limitations we cannot detail how object community signatures are derived from object specifications such as TROLL or GNOME (for details concerning TROLL see [8]). Given a signature Σ and a set of variables X, terms in $T_\Sigma(X)$ are built as usual. Thus, in our approach we have **attribute terms, action terms**, and **transaction terms**. Those are constructed by applying attribute, action or transaction symbols, respectively, to terms. There is one special action term $i.\bot \in T_{\Sigma,act^i}(X)$ (act^i means action of object i) to indicate that an object is currently quiet, i.e., it is not performing an action. We need this because of the life cycle semantics where objects may be quiet while others perform actions (*stuttering*). Our focus is on transactions. Transactions are built by using operators such as sequence, choice or loop. Thus, $\langle t\rangle, \langle t_1; \ldots; t_n\rangle, \mathsf{if}(t_0, t_1, t_2), \mathsf{while}(t_0, t_1) \in T_{\Sigma,trac^i}(X)$ are transaction terms provided that t_k, $k = 1, \ldots, n$, are transaction terms, and t_0 is a boolean term.

As for **formulae** in $Tosl_\Sigma(X)$ we have the common things, i.e., equality on data terms, implication ($\Rightarrow$), negation ($\neg$), all-quantifier ($\forall$), since (S), until (U). With the help of these we can define the usual abbreviations such as $\vee, \wedge, \exists, \mathsf{X}, \mathsf{Y}, \ldots$. There are three special formulae which will provide the basis for giving semantics to object specification with transactions, namely

- $i.\mathsf{c} \in Tosl_\Sigma(X)$ provided that $i \in ID$;
- $\mathsf{B}(t), \mathsf{E}(t) \in Tosl_\Sigma(X)$ provided that $t \in T_{\Sigma,trac^i}(X)$ for some $i \in ID$.

Consistency of an object i ($i.\mathtt{c}$) is a notion that comes along with transactions. Consistent states are those where a transaction has not begun or its execution has just finished, respectively. All states in-between a transaction execution are inconsistent. This is a widely used terminology in database theory. $\mathsf{B}(t)$ refers to the state just **before the execution of a transaction** t and $\mathsf{E}(t)$ refers to the state **after its execution**. We will use these formulae to formalize object specifications with transactions.

In the following we briefly sketch the semantics of this logic. As usual data signatures are interpreted by algebras. Applying an action symbol of an object i to values of appropriate sort from the algebra gives an **action of object** i. Analogously, **attributes and transactions of an object** are received. Thus, referring to the casino example actions of the slot machine are `move(0)`, `move(1), ...`, and attributes are `pot=0, pot=1, ...`. In what follows we write $\mathcal{A}_{act^i}$, $\mathcal{A}_{trac^i}$, and $\mathcal{A}_{att^i_s}$ for actions, transactions, and attributes, respectively, of object i.

Interpretation structures for TOSL formulae are life cycles, i.e., sequences of action occurrences. In every step a set of actions is assigned which are currently executed by the objects of the system. We assume that each object performs one action at the moment. Additionally, there is more information in every step of an objects life cycle. For each step we know, whether the object is currently in a consistent state or not and what are the values of the attributes. Moreover, we allowed for choice and loop operator in transactions. Therefore, the denotation, i.e., meaning of a transaction, depends on the current state of attributes. We illustrate this by means of an example.

Example 5 (Casino – Semantics). In figure 1 part of a slot machine life cycle is depicted. All states are labelled with the current attribute values and the actions that have just been performed. In state 1 attribute lucky holds, the pot equals n-1 and the object is quiet. From state 1 to state 2 the object started the execution of `MOVE` by incrementing the pot. In the following step `pot` is set to zero and the money is paid out. Moreover, a demon changed attribute lucky. Thus, the execution of transaction `MOVE(n)` started in state 1 and has been finished in state 3. State 1 and 3 are consistent objects states (black circle), whereas in state 2 the object is still executing the transaction and therefore is in an inconsistent state.

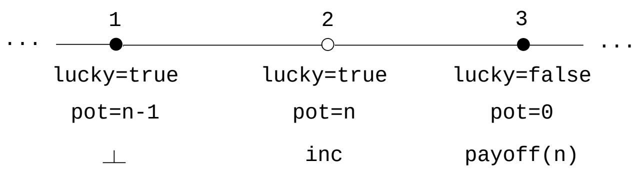


Fig. 1

The interpretation of a transaction that involves choices or loops depends on the state. Eg the denotation of `MOVE(n)=<inc;if(lucky,payoff(n),zeropay(n))>` in state 1 is the sequence `<inc;payoff(n)>` since lucky holds. In contrast, the denotation of `MOVE(n)` in state 3 is the sequence `<inc;zeropay(n)>` since lucky does not hold. Thus, the interpretation structure has to reflect the fact that transactions are interpreted depending on the state.

Formally, an interpretation structure for TOSL, so-called Σ-structure, is an ID-indexed family of maps $\lambda = \{\lambda^i\}_{i \in ID}$, where each λ^i is 4-tuple

$$\lambda^i : \mathrm{N}_o \to \mathcal{A}_{act^i} \otimes \{0,1\} \otimes [\mathcal{A}_{trac^i} \to \mathcal{A}^*_{act^i}] \otimes \{[\mathcal{A}_{att^i_s} \to \mathcal{A}_{DT_s}]\}_{s \in S_{DT}}.$$

For each object i an interpretation structure is a sequential life cycle. For each natural number the 4-tuple consists of the action which has just happened (first component), values of attributes (last component), sequences of actions corresponding to transactions (third component) and a flag stating whether the object is in a consistent state or not (second component).

Denotation of terms is given for a life cycle λ at a specific state k and a given assignment for variables. Let $\xi = \langle \lambda, k, \theta \rangle$ where $k \in \mathrm{N}_o$ and θ is an assignment, with the following conventions

- $\xi @ k' = \langle \lambda, k', \theta \rangle$, i.e., the same life cycle and the same assignment, but a different state;
- $\xi @ \theta' = \langle \lambda, k, \theta' \rangle$, i.e., the same life cycle at the same point in time, but a different assignment.

We will skip the detailed definition of denotation of terms. As already mentioned, the denotation of attribute terms is given corresponding to the last component of the interpretation structure. Analogously, the denotation of transactional terms refers to the third component of the interpretation structure. The intuitive idea is that a transaction term may always be interleaved by finitely many $\perp$ actions. Thus, the denotation of a sequential transaction term is given as the sequence of the denotations of the incorporated terms possibly extended by $\perp$ actions in-between the first and the last term. The denotation of a choice depends on the value of the boolean term t_0. Loop-terms are defined recursively.

We will directly go to the interesting features of our logic, i.e., the transactional concepts. As far as it concerns the normal elements satisfaction of formulae is given as usual. Satisfaction of the three new concepts we introduced (consistency, begin and end of transaction) are defined as follows:

- $\xi \Vdash i.\mathsf{c}$ iff $\lambda^i(k)_2 = 1$;
- $\xi \Vdash \mathsf{B}(t)$ iff exists $k' > k$ such that $\lambda^i[k+1, k'] \in [\![t]\!]_{\xi, trac^i}$;
- $\xi \Vdash \mathsf{E}(t)$ iff exists $k' < k$ such that $\lambda^i[k', k] \in [\![t]\!]_{\xi @ k', trac^i}$.

An object is in a consistent state if this state is labelled as consistent. This information is given in the second component of an interpretation structure λ.

To investigate whether $\mathsf{B}(t)$ holds for a transaction t at point k we have have to look at the denotation of that transaction. The denotation of a transaction is a specific sequence of actions depending on state k. In k all conditions which are part of the specification of t are evaluated. Thus, the denotation of t is a sequence of actions which have to be performed in order to perform t. Now, the formula $\mathsf{B}(t)$ holds for a transaction t at point k if there is a sequence of states starting from state $k+1$ which are labelled with the actions which correspond to the denotation of t. $\lambda^i[k+1,k']$ denotes the life cylce λ^i from state $k+1$ until state k'.

To decide for a state in a life cycle whether it is a state where a transaction has ended, we have to look back to a previous state k'. We get the denotation of the transaction at state k' (its starting point). This sequence of action must correspond to the labels of the life cycle between k' and k. Thus, $\mathsf{E}(t)$ holds at k iff there is a previous state $k^{'}$ such that the states of the life cycle from $k^{'}$ to k are labelled with the denotation of t.

We are almost done with syntax and semantics of our logic so that we can illustrate specification with transaction. There is only one thing missing which is due to the fact that $\mathsf{B}(t)$ and $\mathsf{E}(t)$ have been defined independent of consistent states. Intuitively one would expect, that the end of a transaction is a consistent state as well as the state before the transaction performs actions. All states in-between should be inconsistent. This leads us to the notion of "strong begin" $\mathbb{B}(t)$ and "strong end" $\mathbb{E}(t)$ of transactions. Thus, in a life cycle we are at the "strong begin of a transaction" $\mathbb{B}(t)$ if the necessary sequence of action will follow (i.e., $\mathsf{B}(t)$ holds), the state is consistent (i.e., $i.\mathsf{c}$), and there will be no consistent state until the transaction is finished. Analogously we can define $\mathbb{E}(t)$.

$$\mathbb{B}(t) \equiv \mathsf{B}(t) \wedge i.\mathsf{c} \wedge (((\neg i.\mathsf{c}) \wedge (\neg\, \mathsf{E}(t)))\ \mathsf{U}\ (i.\mathsf{c} \wedge \mathsf{E}(t))),$$
$$\mathbb{E}(t) \equiv \mathsf{E}(t) \wedge i.\mathsf{c} \wedge (((\neg i.\mathsf{c}) \wedge (\neg\, \mathsf{B}(t)))\ \mathsf{S}\ (i.\mathsf{c} \wedge \mathsf{B}(t))).$$

Note, that $\mathbb{B}(t)$ only holds if the whole transactions has been performed. Thus, the semantics of transaction guarantees atomicity. Consistency and isolation are no problems in this approach: (1) If there exists a life cycle model for a specification, then it fulfills all axioms by definition. Thus, it is consistent with the specification and all transition are therefore valid. (2) Isolation is not an issue since transactions operate sequentially on objects. There is no shared data. This will be future work, when we introduce transactions which incorporate different objects and which may perform concurrently. Durability is not an issue since we assume objects to be persistent.

Example 6 (Strong begin/end vs. weak begin/end). $\mathsf{B}(t), \mathsf{E}(t)$ are "weak forms of begin and end" in the sense that this does not imply that the transaction indeed happens. For instance, assume two transaction specifications `X=<a;b;c;d>` and `Y=<b;c>` of one object. The life cycle in figure 2 satisfies the following formulae: "B(`X`) holds at state 1", "B(`Y`) holds at state 2", "E(`Y`) holds at state 4", and "E(`X`) holds at state 5", respectively.

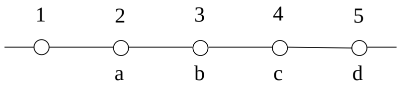


Fig. 2

But only one transaction can happen in one instance of time due to our assumption that objects are sequential. That is the reason for introducing consistency. There are various possibilities for valid life cycles. In figure 3 two such life cycles are illustrated. The first one corresponds to execution of transaction X whereas the second one corresponds to execution of transaction <a>, followed by transaction Y, followed by action <d>.

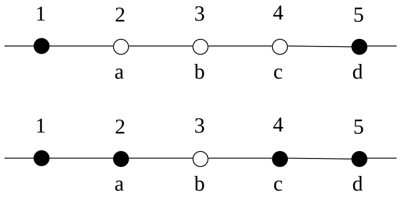
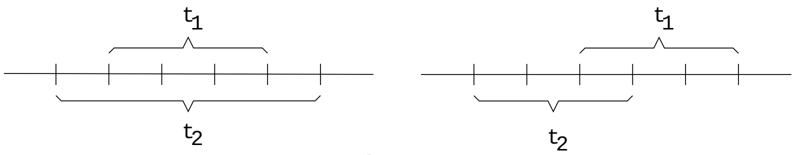


Fig. 3

For that reason we introduce the notions “strong begin” ($\mathbb{B}$) and “strong end” ($\mathbb{E}$) of a transaction. In figure 3 the following “strong” formulae hold. In the first life cycle “$\mathbb{B}$(X) holds at state 1” and “$\mathbb{E}$(X) holds at state 5”. In the second life cycle of figure 3 “$\mathbb{B}$(<a>) holds at state 1”, “$\mathbb{E}$(<a>) holds at state 2”, “$\mathbb{B}$(Y) holds at state 2”, “$\mathbb{E}$(Y) holds at state 4”, “$\mathbb{B}$(<d>) holds at state 4”, “$\mathbb{E}$(<d>) holds at state 5”.

4 The solution: specifying with transactions

We start with synchronization of transactions. A transaction t_1 calls another transaction t_2 if the execution of t_1 *requires* in some sense the execution of t_2. As already pointed out in example 3 this may be appropriately formalized as “t_1 and t_2 overlap *during* their execution”. In figure 4 four possibilities of transaction overlapping are illustrated.



and vice versa

Fig. 4

Using the definitions for strong begin and end, we can define $\mathsf{during}(t)$ as an abbreviation:

$$\mathsf{during}(t) \equiv ((\neg\, \mathbb{E}(t)) \mathsf{S} \mathbb{B}(t)) \vee \mathbb{E}(t).$$

$\mathsf{during}(t)$ incorporates all states where actions of the current transaction t are performed. Together with this notion we can now define t_1 requires t_2, $t_1 \gg t_2$, i.e., synchronization of transactions:

$$t_1 \gg t_2 \equiv \mathbb{E}(t_1) \Rightarrow [\mathsf{during}(t_2) \vee (\mathsf{during}(t_1)\, \mathsf{S}\, (\mathbb{E}(t_2) \wedge \mathsf{during}(t_1)))]).$$

Now we can formalize synchronisation in object systems with transactions.

Example 7 (Casino: synchronisation). The behavior of the player specification as given in example 1 is formalized as the following TOSL formula: <play(n)> $\gg$ MOVE(n).

Effects of *actions* could be easily defined without TOSL. But since the granularity of an action may change through applying refinement techniques etc. we take a different approach. **The formalization is always given in terms of transactions. This guarantees that the formulae will hold independently of the level of granularity.** For instance, the specification `inc` ***do*** `pot=pot+1` ***od*** says that the value of pot will be increased by one (depending on the value of pot before incrementing). Using TOSL for formalization gives an appropriate formula which holds true independently from the granularity of `inc`.

We introduce the following abbreviation: $\mathsf{inside}(t) \equiv (\neg\, \mathbf{E}(t) \mathsf{S}\, \mathbf{B}(t)) \wedge (\neg\, \mathbf{E}(t))$.

Example 8 (Casino: effect and precondition). The behavior of example 3 is translated to the following TOSL axioms:

$$(\mathbf{B}(\texttt{<inc>}) \wedge \texttt{pot=n}) \Rightarrow \;\; [\mathsf{inside}(\texttt{<inc>})\, \mathsf{U}\, (\mathbf{E}(\texttt{<inc>}) \wedge \texttt{pot=n+1})]$$
$$\mathbf{E}(\texttt{<payoff(n)>}) \Rightarrow \texttt{pot=0}$$
$$\mathbf{B}(\texttt{<payoff(n)>}) \Rightarrow \texttt{pot=n}$$
$$\mathbf{B}(\texttt{<zeropay(n)>}) \Rightarrow \texttt{n} = 0$$

The third and fourth formulae describe preconditions for `<payoff>` and `<zeropay>`. These formulae refer to the beginning of those transactions. The first and second formulae correspond to effects of `inc` and `payoff`. In case of incrementing we have to refer to the value before the incrementing started.

We have illustrated by example that TOSL is very well suited for specifying object systems with transactions. Moreover, the given axioms are stable wrt to refinement in the sense that they are independent from the actual granularity of a transaction. We do not aim at specifying with TOSL rather than proposing intuitive language macros at the specification level which are semantically defined in terms of TOSL.

5 Concluding remarks

In this paper we investigated the concept of transaction in object oriented specification. We presented a linear temporal logic with transaction which serves as denotational semantics for object specifications with transactions. We extended Object Specification Logic (OSL) by the notion of transaction. Semantics of Transactional Object Specification Logic (TOSL) has been given in terms of life cycles and its use was illustrated by means of examples. One of the main advantages is that TOSL is capable of specifying formulae independently of the level of granularity.

Future work will be towards simulation and reification of object specification. The former deals with observational equivalence of objects. Abstract specifications are refined to more concrete ones by refining actions and transactions and specifying new axioms. In case of reification we only allow actions to be refined whereas transactions remain. The advantage of this approach is that refined formulae can be derived from abstract ones. Both cases are useful for high level specification. In the current framework, we restricted ourselves to sequential systems. There exists already work about extending DTL [7] with the notion of transaction (see [4, 5]). In this distributed framework serializability of transactions which share data as well as reification has been treated. A totally different way of dealing with transactions would be to provide a new semantics for our specification language. For instance, the work of Liu and Orgun about temporal logics with clocks seems very promising in this respect.

Acknowledgements: We would like to thank the referees for their valuable comments. The first author would like to thank her colleagues for internal discussions on the topic of the paper. Especially, the comments of Hans Dieter Ehrich and Juliana Küster Filipe helped to improve the paper.
This work was partially supported by DFG under Eh75/11-1, the PRAXIS XXI Program and JNICT, as well as by PRAXIS XXI Projects 2/2.1/MAT/262/94 SitCalc, PCEX/P/MAT/46/96 ACL plus 2/2.1/TIT/1658/95 LogComp, and ESPRIT IV Working Groups 22704 ASPIRE and 23531 FIREworks.

References

1. H. Barringer, R. Kuiper, and A. Pnueli. A Really Abstract Concurrent Model and its Temporal Logic. *ACM Symp. Principles of Programming Languages*, pages 173–183, 1989.
2. Bonner, A.J. and Kifer, M. Concurrency and Communication in Transaction Logic. In M. Maher, editor, *Proc. Joint Int. Conf. and Symp. on Logic Programming (JICSLP96), September 2-6, 1996, Bonn, Germany.* The MIT Press, 1996.
3. P. Borba and J. Goguen. On Refinement and FOOPS. Technical Report, PRG-TR-17-94 , Oxford University Computing Laboratory, Programming Research Group, 1994.
4. G. Denker. Reification – Changing Viewpoint but Preserving Truth. In M. Haveraan, O. Owe, and O.-J. Dahl, editors, *Recent Trends in Data Types Specification, Proc. 11th Workshop on Specification of Abstract Data Types joint with the 8th General COMPASS Meeting. Oslo, Norway, September 1995. Selected papers.*, pages 182–199. Springer, 1996. LNCS 1130.
5. G. Denker. Semantic Refinement of Concurrent Object Systems Based on Serializability. In B. Freitag, C. B. Jones, C. Lengauer, and H.-J. Schek, editors, *Object Orientation with Parallelism and Persistence*, pages 105–126. Kluwer Academic Publ., 1996.
6. G. Denker and P. Hartel. TROLL – An Object Oriented Formal Method for Distributed Information System Design: Syntax and Pragmatics. Technical Report 97–03, TU Braunschweig, 1997. `http://www.cs.tu-bs.de/idb/publications/pub_97`
7. H.-D. Ehrich. Object Specification. Technical Report 96–07, TU Braunschweig, 1996. `http://www.cs.tu-bs.de/idb/publications/pub_96`
8. H.-D. Ehrich and A. Sernadas. Local Specification of Distributed Families of Sequential Objects. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Types Specification, Proc. 10th Workshop on Specification of Abstract Data Types joint with the 5th COMPASS Workshop, S.Margherita, Italy, May/June 1994, Selected papers*, pages 219–235. Springer, Berlin, LNCS 906, 1995.
9. J.L. Fiadeiro and T. Maibaum. Sometimes "Tommorrow" is "Sometime" – Action Refinement in a Temporal Logic of Objects. In D. M. Gabbay and H. J. Ohlbach,

editors, *Proc. First Int. Conf. on Temporal Logic, ICTL, Bonn, Germany, July 1994*, pages 48–66. Springer, 1994. LNAI 827.
10. M. Huhn. Action Refinement and Property Inheritance in Systems of Sequential Agents. In *Proc. 7th Int. Conf. on Concurrency Theory, Concur'96, 26-29 August, Pisa, Italy*. Springer, 1996. LNCS 1119.
11. M. Huhn, H. Wehrheim, and G. Denker. Action Refinement in System Specification: Comparing a Process Algebraic and an Object-Oriented Approach. In U. Herzog and H. Hermanns, editors, *GI/ITG-Fachgespräch: "Formale Beschreibungstechniken für verteilte Systeme", 20/21. Juni 1996, Universität Erlangen, Germany*, number 29/9 in Arbeitsbericht des IMMD, pages 77–88, 1996.
12. R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. TROLL – A Language for Object-Oriented Specification of Information Systems. *ACM Transactions on Information Systems*, 14(2):175–211, April 1996.
13. D. Kozen and J. Tiuryn. Logics of Programs. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science B - Formal Models and Semantics*, chapter 789–840. 1990.
14. L. Lamport. Specifying Concurrent Program Modules. *ACM Trans. on Programming Languages and Systems*, 5:190–222, January 1983.
15. L. Lamport. The Temporal Logic of Actions. *ACM Trans. on Programming Languages and Systems*, 16(3):872–923, May 1994.
16. U. Lechner, C. Lengauer, and M. Wirsing. An Object-Oriented Airport: Specification and Refinement in Maude. In Astesiano, E. and Reggio, G. and Tarlecki, A., editor, *Recent Trends in Data Types Specification, Proc. 10th Workshop on Specification of Abstract Data Types joint with the 5th COMPASS Workshop, S.Margherita, Italy, May/June 1994, Selected papers*, pages 351–367. Springer, Berlin, LNCS 906, 1995.
17. C. Liu and M. A. Orgun. Dealing with Multiple Granularity of Time in Temporal Logic Programming. *Journal of Symbolic Computation*, 22(5 & 6):699–720, 1996.
18. K. Lodaya, R. Ramanujam, and P.S. Thiagarajan. Temporal Logics for Communicating Sequential Agents. *Int. Journal of Foundations of Computer Science*, 3(2):117–159, 1992.
19. J. McCarthy and P. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.
20. V. R. Pratt. Semantical Considerations on Floyd-Hoare Logic. In *Proc. 17th Ann. IEEE Symp. on Foundations of Computer Science*, pages 109–121, 1976.
21. R. Reiter. Proving Properties of States in the Situation calculus. *Artificial Intelligence*, 64(2):337–351, 1993.
22. A. Sernadas and J. Ramos. The GNOME Language: Syntax, Semantics and Calculus. Technical Report, Instituto Superior Téchnico (IST), Dept. Mathemática, Av. Roviso Pais, 1096 Lisboa Codex, Portugal, 1994.
23. A. Sernadas, C. Sernadas, and J.F. Costa. Object Specification Logic. *Journal of Logic and Computation*, 5(5):603–630, October 1995.
24. A. Sernadas, C. Sernadas, and J. Ramos. A temporal logic approach to object certification. *Data & Knowledge Engineering*, 19:267–294, 1996.
25. E. Zucca. Implementation of data structures in an imperative framework. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Types Specification, Proc. 10th Workshop on Specification of Abstract Data Types joint with the 5th COMPASS Workshop, S.Margherita, Italy, May/June 1994, Selected papers*, pages 483–498. Springer, Berlin, 1995. LNCS 906.