B. Banieqbal, H. Barringer, and A. Pnueli, editors, Proc. Colloq. on Temporal Logic in Specification, LNCS 398, pages 324–350, Berlin, 1989. Springer–Verlag

ABSTRACT OBJECT TYPES:
A TEMPORAL PERSPECTIVE

A. Sernadas[1], J. Fiadeiro[1], C. Sernadas[1] and H.-D. Ehrich[2]

(1) Dep. Matemática, IST, 1096 Lisboa Codex, Portugal
uucp: mcvax!inesc!acs
(2) Inst. Informatik, TUB, PF 3329, 3300 Braunschweig, FRG
uucp: mcvax!infbs!ehrich

Abstract - The notion of abstract object type (AOT) tends to overlay the already classical concept of abstract data type (ADT) in several fields of application. Objects, although much more complex than data, have the advantage of dealing with states and processes. For that reason, they become useful, for instance, in the design of database applications and in software engineering. The difficulty lies in finding a suitable formalism for the abstract definition of objects, at least as effective as the equational formalism has been in the definition of abstract data types. The purpose of this paper is to present and discuss the main features of such a formalism. Concepts, tools and techniques are provided for the abstract definition of objects. A primitive language is presented allowing structured and rather independent definitions of object types. Each object is described as a temporal entity that evolves because of the events that happen during its life. The interaction between objects is reduced to event sharing. Both liveness and safety requirements can be stated and verified. Two case studies are presented for illustrating every aspect of the approach: the stack example which is very popular in the ADT area, thus allowing the comparison between the concepts of ADT and AOT, and the well known example of the eating philosophers which allows the discussion of the dynamic aspects.

1- Introduction

The notion of abstract object type (AOT) tends to overlay the already classical concept of abstract data type (ADT) in several fields of application. Objects, although much more complex than data, have the advantage of dealing with states and processes. For that reason, they become useful, for instance, in the design of database applications by providing a unifying view of the transactions and the evolving database records. Their usefulness in software engineering is well known. The difficulty lies in finding a suitable formalism for *the abstract specification of objects*, at least as effective as the equational formalism has been in the specification of abstract data types.

A recent approach [SSE87] to the algebraic specification of societies of interacting objects is analysed herein from the point of view of the associated linear tense logic. Each object is described as a temporal entity that evolves because of the events that happen during its life. The interaction between objects is reduced to the simple mechanism of event sharing. Concepts, tools and techniques are provided for the abstract definition of: the universe of objects and their attributes, assuming the necessary abstract data types; the

temporal evolution of those objects in terms of the varying values of their attributes; the space of all possible events and traces; the set of admissible traces; the sharing of events; and, finally, the temporal evolution of the object as a function of the sequence of events that have already occurred (the trace). Behavioural constraints of several kinds can be imposed both at the level of the temporal evolution of the attributes and at the level of the events. It is possible to check the correctness of the latter against the temporal description. Both liveness and safety requirements can be stated and verified.

For this purpose the language of a layered, linear tense logic of objects and events is outlined. Its Kripke interpretation structures [Kri63] are easily built within the context of the algebraic semantics of the universe of objects as first introduced in [SSE87]. Herein, the constructs for behavioural specification are revised and interpreted in the linear Kripke structures. Moreover, special attention is given to the specification and verification of liveness requirements. Finally, the proposed semantics is compared with the trace semantics of deterministic processes as defined in [Hoa85], showing that the former is more complete because it can deal with liveness aspects (total correctness). The usefulness of the temporal approaches to systems design has been recognised as long as [Pnu77,Pnu79]. The results presented in these papers show that the temporal perspective is indeed essential to the concept of *object*.

In Section 2, the abstract specification of an isolated, passive object type (the stack) is presented and commented in order to show the main features of the approach. In Section 3, the abstract specification of concurrent objects is illustrated using a simplified version of the eating philosophers problem originally due to E. Dijkstra. This example is sufficiently rich to allow the discussion of the interaction between concurrent objects, some of them passive (like the forks) and some of them active (like the philosophers). Finally, in Section 4, the algebraic and Kripke semantics are introduced and fully discussed.

2- The AOT Stack

An object is a temporal entity that preserves its own identity throughout time even if some of its attributes are allowed or made to change. Hence, we can say that an object has a time-varying **state**. The state-dependent attributes of the object are called **slots**. On the other hand, some state-independent attributes will exist in order to support the identification mechanism. Such attributes are called **key attributes**. Besides the key attributes, the object may have other state-independent attributes that are *not* essential to the identification of the object: the so-called **constant attributes**. Some behavioural characteristics of the object can be described using temporal formulae constraining the values of the slots. Other behavioural characteristics of the object cannot be described at that level of abstraction. They require the reference to the events that happen in the life

of the object. We shall refer to the former as the **temporal level** and to the latter as the **event level.**

With respect to the event level of abstraction, we assume that, during its lifetime, from its **birth** to its (optional) **death**, the object will evolve according to the **update** events that take place. Moreover, we also assume that the occurrence of events is the only cause of change in the state of the object. That is to say, at each instant, we shall identify the state of the object with the **trace** of the events that have already happened in its life since its birth. This trace is **empty** for the unborn objects. Among the events that are possible for that object we distinguish three categories: the creation (birth) events, the modification (update) events and the destruction (death) events. In every possible trace of an object there will be at most one creation event (its birth event), followed by an indeterminate number of update events (possibly none). If a destruction event occurs it must be the last event.

The mapping between the two levels of abstraction is achieved as follows. The values of the object slots as functions of the possible traces should comply with the temporal constraints on the slots, assuming that we interpret these constraints in the Kripke structure corresponding to the set of the **admissible traces**. This set (henceforth denoted by $\mathbb{T}(b)$ for each object b) may be constrained by **safety** rules, such as the **enabling conditions** that forbid the occurrence of some event unless a (state-dependent or, rather, trace-dependent) condition is fulfilled.

The behaviour of the object at the event level of abstraction is not completely captured by the set of traces. Indeed, the **liveness** characterisation of the object implies a richer model, as discussed later on. On the other hand, objects will not be in isolation and in most interesting cases will run concurrently, interacting with each other. But, for now, let us consider only isolated, passive objects, namely stacks. Naturally, we describe types of objects instead of defining separately each object occurrence.

The stack example is particularly useful at this stage because of its popularity in the ADT literature. Compare the ADT stack specification (signature plus equations), as given for instance in [EM85], with the following description of the corresponding AOT (where we assume a previous definition of the relevant ADTs - booleans and integers).

```
object type  STACK
  invariant
    katt
      name:  string
  state
    slots
      empty?: bool ;
      top:  integer
    scns
      (empty? = true)  ⇒  (top = error_int)
```

```
    change
      generation
        birth
          open
        update
          push(integer)  ;
          pop
        death
          close
      valuation
        { } open { empty? = true } ;
        { } open { top = error_int } ;
        { } push(N) { empty? = false } ;
        { } push(N) { top = N }
      reduction
        push(N); pop >> nil
      safety
        {{ empty? = false }}  pop
end
```

Avoiding any comments on the *syntactic sugar* of the adopted notation (OBLOG from OBject LOGic), we distinguish three main components in the object type definition: the invariant properties of the objects (key mechanism and so on); their behaviour at the temporal level (declaration and constraining of slots); and their behaviour at the event level (events, valuation map relating the slots values with the traces, etc). As expected the third part is the most complex.

The **invariant** part of the description defines the universe of all possible objects (in this case, the set of all possible stacks), including their identification mechanism and, possibly, their constant attributes (in this case, none). The set of all possible stacks is constructed from strings as follows: a different stack is considered for each string. The **key attribute** maps each such stack to the corresponding string (its name). From an algebraic point of view, in this trivial case, we are freely generating the set of stacks using a (hidden) key map corresponding to the reverse of the key attribute. In general, the problem of the universe construction is more difficult as briefly discussed later on (Paragraph 4.1).

The **state** part of the description introduces the **slots**, as well as any of the temporal properties of their values. In this case, we have two slots: empty? and top. The former is intended to tell us when each stack is empty or not. The purpose of the latter should be obvious too. We also have a **slot constraint**:

$$(\text{empty?} = \text{true}) \Rightarrow (\text{top} = \text{error\_int})$$

This state constraint is trivial, namely because it does not involve any tense operator. But we can easily conceive a more complex constraint, such as:

$$(\text{empty?} = \text{true}) \Rightarrow (\mathbf{G}(\text{empty?} = \text{true}))$$

This constraint would impose a rather funny behaviour to the stacks: once empty each stack would remain empty for ever (because the tense operator **G** means always in the future). The semantics of slots and their constraints is given in Paragraph 4.2.

The **change** part is further subdivided into three parts as follows. First, the alphabet of the possible events is introduced by giving the event **generation** functions (dropping the distinguished, compulsory object argument referring to the target of the event). In this case, besides the event generators open, close and pop (mapping each stack into different events), the event generator push (mapping each pair stack/integer into a push event) is also necessary. Hence, for each stack b, the alphabet of its events is as follows:

$$\mathbb{E}(b) = \{ \text{open}(b), \text{close}(b), \text{pop}(b) \} \cup \{ \text{push}(b,n): n \text{ is integer} \}$$

At this point, the **categories** of the events are also fixed (eg open generates creation events). From the algebraic point of view, the construction of the alphabet of events is also achieved by free generation, using the event generators (refer to Paragraph 4.3 below).

After the definition of the events, we have the **valuation** and the **reduction** parts that together establish the mapping from the event sequences to the values of the slots. In this case, for instance, the constraint (a "pre and post condition")

$$\{ \} \text{ push}(N) \{ \text{top} = N \}$$

states that the value of top on any trace terminating with a push event is the argument used in the generation of the event (for each stack occurrence). As an illustration of reduction, consider the following rule:

$$\text{push}(N); \text{pop} >> \textbf{nil}$$

where **nil** denotes the empty event sequence <>. It states that, for the sole purpose of constructing the valuation map, a push followed by a pop is equivalent to nothing happening. These rules together with the pre and post conditions provide the necessary means for stating the values of the slots as functions of the event sequences, as abstractly as possible: the valuation map (see Paragraph 4.4).

Finally, the set $\mathbb{T}(b)$ of the admissible traces (a subset of $\mathbb{E}(b)^*$) can be reduced by imposing **safety** constraints such as the following enabling constraint:

$$\{\{ \text{empty?} = \text{false} \}\} \text{ pop}$$

This constraint excludes from T(b) any event sequence having a pop after an initial part where empty? has the value true (according to the valuation map above). The semantics of the enabling constraints and other types of constraints are dealt with in Paragraph 4.5.

Comparing with a traditional specification of stacks as an ADT, we should stress that some operations on the ADT become slots on the AOT; others become events. Moreover, the identification mechanism described in the AOT is completely missing in the ADT (because it is irrelevant for data). But there is a strong resemblance between the two descriptions, as it should be expected.

The complexity in the AOT description is the price we have to pay for talking about temporal existence. Further capabilities of the AOT approach concerning the interaction between concurrent objects are illustrated in Section 3.

3- Interaction Between Concurrent Objects

Let us consider the example of the **eating philosophers** (a simplification of a rather popular example in the literature on concurrency). At first sight we might think that only two AOTs are relevant: the philosophers and the forks. However, we shall see that other object types are also important. They are all subtypes of the type philosopher (corresponding to working, looking for forks, and having a meal), illustrating very well the case for transient, active objects. They also serve to illustrate the fuzzy, subtle difference of perspective betweeen object-oriented specifications (that rely strongly on specialisation) and process-oriented descriptions (that rely heavily on the modeling of the tasks at hand).

Let us assume, for the sake of simplicity, that we have five philosophers (named unimaginatively 0 to 4) sitting (anticlockwise) around a table and sharing five forks (risking the wrath of the local health authority). Each of them may be thinking (working), looking for the forks, or actually eating (from the traditional central bowl). They seat all the time at their places. The fork arrangement is as usual: each philosopher needs two forks when eating. Hence, the left and right neighbors of a philosopher that is eating are both unable to start eating even if they desire to do so.

In the following OBLOG specification we use simplified definitions of the necessary data types, adopting a Pascal-like notation (eg 0..4), since our main target for discussion is the specification of the abstract object types.

Notice that slots are only locally visible, ie, an object does not have access to the value of a

slot from another object.

```
object type FORK
  invariant
    katt
      lph: PHIL                                   /* left philosopher */
      rph: PHIL                                   /* right philosopher */
    kcns
      rph(self) = next(lph(self))    /* to the dismay of the health authority */
  state
    slots
      free?: bool
  change
    generation
      birth
        eos
      update
        up_by_lph ;
        up_by_rph ;
        dw_by_lph ;
        dw_by_rph
    valuation
      { } eos { free? = true } ;
      { } up_by_lph { free? = false } ;
      { } up_by_rph { free? = false } ;
      { } dw_by_lph { free? = true } ;
      { } dw_by_rph { free? = true }
    safety
      {{ free? = true }} up_by_lph ;
      {{ free? = true }} up_by_rph ;
      {{ free? = false }} dw_by_lph ;
      {{ free? = false }} dw_by_rph
end
```

According to the invariant part of the definition, each fork is identified through the philosopher on its left and the philosopher on its right. The key constraint relates the two philosophers with respect to each other, namely that the philosopher on the right-hand side of a fork is the philosopher sitting next to the philosopher on the left-hand side of the fork (the constant attribute next is defined in the object type PHIL below). From the definition of the object type PHIL below, it results that there is one and only one fork between two philosophers sitting next to each other (!).

Forks are considered to be **passive objects** because there are no liveness requirements on their behaviour. Intuitively, this means that each fork will not make any attempt to evolve, the events occurring in its life being necessarily shared with active objects (see below).

```
object type PHIL                                  /* PHILOSOPHER */
  invariant
    katt
      name: 0..4
    catts
      next: PHIL
      rfk: FORK;                                  /* right fork */
      lfk: FORK                                   /* left fork */
```

```
    ccns
      next(self) = phil(mod(name(self)+1,5))
      rfk(self) = lfk(next(self))
  state
    slots
      cond: (working,hungry) ;
      pay: integer
    scns
      (pay = N) ⇒ (G((pay≥N) = true)) ;
      (cond = working) ⇒ (F(cond = hungry)) ;
      (cond = hungry) ⇒ (F(cond = working))
  change
    generation
      birth
        eos
      update
        bc_hungry ;                          /* become hungry */
        grab_ff ;                            /* grab both forks */
        free_ff                              /* free both forks */
    valuation
      { } eos { cond = working } ;
      { } eos { pay = 0 }
      { } bc_hungry { cond = hungry } ;
      { pay = N } bc_hungry { pay = N } ;
      { cond = C } grab_ff { cond = C } ;
      { pay = N } grab_ff { pay = N } ;
      { } free_ff { cond = working } ;
      { pay = N } free_ff { pay = (N+1) }
    safety
      {{ cond = working }} bc_hungry ;
      {{ cond = hungry }} grab_ff
end
```

According to the invariant part of the definition, the identification mechanisms of philosophers do not depend on other object types but only on data types. The constant attributes identify, for each philosopher, the philosopher that is sitting next to him (to its right) as well as his right and left fork.

The two slots cond and pay give, for each philosopher, the current status, either eating or hungry, and the number of meals he has had. According to the constraints, the number of meals does not decrease. On the other hand, each philosopher is required to be recurrently eating and recurrently hungry.

At this level of the description, philosophers are passive objects. However, it is possible to define subtypes of PHIL, each accounting for one of the different activities that a philosopher may be undertaking: either working, grabbing the forks or having a meal:

```
object subtype WORKING_PHIL of PHIL
  change
    roles
      becomes
        eos ;
        free_ff
      begoes
        bc_hungry
```

```
    liveness
      bc_hungry
end
```

A philosopher becomes working when he is created or when he frees his forks. This is indicated in the clause **becomes**: an event of category becomes marks the point where an object has become an instance of a subtype. On the other hand, a philosopher ceases to be working when he becomes hungry. This is indicated in the clause **begoes**: an event of category begoes marks the point where an object ceases to be an instance of a subtype. An instance of a subtype inherits all the slots and events of its parents. However, it may have additional slots and events of category update, which was not the case above.

Working philosophers are **active objects**: they have **liveness requirements**. In this case, the liveness requirement is a very primitive one: the guarantee of terminating the activity (working). This means that every working philosopher is looking forward to becoming hungry again.

```
object subtype  GRABBING_PHIL
  change
    roles
      becomes
        bc_hungry
      begoes
        grab_ff
    liveness
      grab_ff
end
```

This is another case of active objects. After becoming hungry, a philosopher tries to seize both the forks he needs for eating.

```
object subtype  EATING_PHIL
  change
    roles
      becomes
        grab_ff
      begoes
        free_ff
    liveness
      free_ff
end
```

After seizing both forks, a philosopher becomes eating (until it frees the forks again). The inclusion of the liveness requirement makes eating philosophers active objects, ie, eager to go back working. Notice that if liveness was not required, philosophers could stay eating forever, leaving other philosophers starving.

Finally, the interaction between the different objects is specified as follows:

```
interaction
  between PHIL(P), FORK(F)
```

```
    eqns
      P.eos = F.eos ;
      P.grab_ff  =  lfk(P).up_by_rph  ;
      P.grab_ff  =  rfk(P).up_by_lph  ;
      P.free_ff  =  lfk(P).dw_by_rph  ;
      P.free_ff  =  rfk(P).dw_by_lph
end
```

The first interaction equation means that all philosophers and forks are created simultaneously. When a philosopher P grabs both forks at his side, that event is shared by those forks, as indicated. For instance, the equation

P.grab_ff = lfk(P).up_by_rph

states that a fork grabbing event by P is seen by his left fork as a "up by the right philosopher" event. Such an event is seen by his right fork as a "up by the left philosopher" event. Notice that, for the sake of simplicity, we choose to have both forks picked up (and put down) at the same time by each philosopher.

4- Algebraic Semantics

A specification of a society of interacting objects is of the general form

SOC = DT + OBJ + INT

(that is to say, the specification DT of the data types, plus the specification OBJ of the object types and the specification INT of the object interactions). The specification of the object types can be reorganized as follows

OBJ = INV + STA + EVT + VAL + PRC

where INV, STA, EVT, VAL and PRC are the specifications of the invariant components, the states, the events, the valuations and the processes, respectively, of the object types.

In the subsequent paragraphs, the semantics of SOC will be seen to correspond to: (1) an algebra UNIVERSE of data and objects, containing the data, the occurrences of the object types and their invariant attributes; (2) the class KRIPKE1 of local structures providing the temporal evolution of the slots of every object occurrence in UNIVERSE; (3) the global space EVENT of all possible events in UNIVERSE; (4) for each object occurrence, a local valuation function mapping each sequence of events into the values of the slots after that sequence; (5) the class KRIPKE2 of local structures providing the evolution of the slot values of every object occurrence in UNIVERSE as a function of the admissible sequences of events; (6) and the class KRIPKE3 of global structures that reflect the joint behaviour

of the objects.

The **temporal-level specification** is composed of DT, INV and STA. Its semantics is KRIPKE1 as indicated above. The **event-level specification** is a refinement of the temporal one, containing in addition, EVT, VAL and PRC, as well as INT, but ignoring the temporal constraints on the slots. Its semantics is given by the refinement KRIPKE2 of KRIPKE1 in which each instant corresponds to a sequence of events (the trace of events up to that instant).

The compliance of the event-level specification against the temporal-level specification is verified by checking if the former entails the satisfaction of the slot constraints.

Please refer to the Appendix for a whole picture of this semantic construction.

4.1- The Global Space of Data and Objects: UNIVERSE

The construction of UNIVERSE from DT and the collection INV of specifications of the invariant components of the object types was already described in detail in [SSE87] for the special case of simple object types without constant attributes. The construction is easily extended in order to cater for such constant attributes, as discussed below. We shall not consider the problems raised by the complex objects, such as generalisation, specialisation and aggregation, which are dealt with in [ESS88]. Nevertheless, a few words on subtyping will be given throughout.

Briefly, we can assume that the semantics DATA of DT is the initial DT-algebra. This algebra contains a domain for each data sort and a function for each operation in DT. Moreover, it satisfies the equations in DT. Because of initiality DATA has other interesting properties [EM85] that we shall not discuss herein.

The specification INV is composed of KEY (the specification of the key mechanisms) and CONS (the specification of the constant attributes). Note that, for each object type β, the key constraints (respectively, the invariant constraints) are written in the equational language over the signature of DT + KEY (respectively, the signature of DT + KEY + CONS) taking the symbol **self** as a variable of sort β.

In [SSE87] the specification KEY was taken as an extension of DT, introducing new sorts (the object sorts), new operations (the key maps and the key attributes) and key equations and constraints. Allowing only equational key axioms, the semantics of DT + KEY is *the free extension* OBJECT of DATA with respect to KEY. Further information on this topic can be found in [EDG86,Ehr86].

The algebra OBJECT contains, for each object type (sort) β, the domain $\mathbb{B}(\beta)$ of all possible

object occurrences that can be generated using the respective key map (or, equivalently, that can be named using the respective key attributes). Naturally, the reduct of OBJECT to the signature of DT should be DATA. We shall denote by $\mathbb{B}$ the disjoint union of all $\mathbb{B}(\beta)$.

The specification CONS is a further extension that introduces the constant attributes and their axioms (also assumed herein to be equations). The semantics of DT + KEY + CONS is the set of *all possible extensions* of OBJECT with respect to CONS. The reduct of any such extension to the signature of DT + KEY should be OBJECT. Any such extension provides a function for each constant attribute between the suitable domains.

In the sequel, we shall assume that *one* such extension was chosen as the basis for the rest of the construction. *The* chosen extension is called UNIVERSE.

As an illustration, for the specification of the eating philosophers UNIVERSE will contain the following domain of philosophers:

$$\mathbb{B}(\text{PHIL}) = \{ p_0, p_1, p_2, p_3, p_4 \}$$

The interpretation of the key attribute of the philosophers could be as follows (up to isomorphism):

$$\underline{\text{name}}(p_0) = \underline{0}$$
$$\ldots$$
$$\underline{\text{name}}(p_4) = \underline{4}$$

assuming that $\mathbb{B}(0..4) = \{ \underline{0}, \underline{1}, \underline{2}, \underline{3}, \underline{4} \}$. Note that we are writing $\underline{f}$ instead of f_{UNIVERSE} for the interpretation of f in UNIVERSE.

Moreover, the *universal key constraints* (omitted in the specifications), establish that the key attribute is the reverse of the key map:

$$\underline{\text{PHIL}}(\underline{\text{name}}(p)) = n \quad \text{for every p in } \mathbb{B}(\text{PHIL})$$

Notice that the identifier of the implicit key map is taken to be the identifier of the corresponding object type.

With respect to forks:

$$\mathbb{B}(\text{FORK}) = \{ f_0, f_1, f_2, f_3, f_4 \}$$

where, for instance,

$\underline{rph}(f_0) = p_0 = \underline{lph}(f_1)$
$\underline{rph}(f_1) = p_1 = \underline{lph}(f_2)$
$\underline{rph}(f_2) = p_2 = \underline{lph}(f_3)$
$\underline{rph}(f_3) = p_3 = \underline{lph}(f_4)$
$\underline{rph}(f_4) = p_4 = \underline{lph}(f_0)$

since $\underline{next}(p_0) = p_1$ and so on.

In this example, once the OBJECT is fixed, there is only one choice for UNIVERSE, because the interpretation of the constant attributes (next, lfk, rfk, lph, rph) is completely determined by the equations

next(P) = phil(mod(name(P)+1,5))
rfk(P) = lfk(next(P))
rph(F) = next(lph(F))

Finally, it remains to analyse the domains of the subtypes. For instance,

$\mathbb{B}$(WORKING_PHIL) = $\mathbb{B}$(PHIL)

because the set of all possible occurrences of a subtype is identical to the set of all possible occurrences of the original type (unless further constraints are imposed on the subtype, which was not the case).

Recall that $\mathbb{B}$ is the disjoint union of the $\mathbb{B}(\beta)$ for every object type β. Hence, in this example, $\mathbb{B}$ will contain four different elements for each philosopher (corresponding to the following types: PHIL, WORKING_PHIL, GRABBING_PHIL and EATING_PHIL).

4.2- The Local Temporal Structures: KRIPKE1

Once UNIVERSE is chosen, we can proceed with the interpretation of the state specification STA. This specification is yet another extension that introduces no further sorts. It only introduces the slots and their constraints. These are introduced independently for each object type β. The slot constraints over objects in β are written in the language $TL(\beta)$: the first order linear tense language on the signature of DT + INV enriched with the slots of β taken as 0-ary function symbols and the symbol self taken as a constant of sort β. Moreover, if β is a subtype of some β', then $TL(\beta)$ is further enriched with the slots of β' taken yet again as 0-ary functions.

The semantics KRIPKE1 of DT + INV + STA is the $\mathbb{B}$-indexed family $(\mathbb{K}1_b)_{b:\mathbb{B}}$ such that each $\mathbb{K}1_b$ is the set of *all possible* local temporal structures of the form <UNIVERSE,$[\![\_]\!]^b_{k:\mathbb{N}}$> that satisfy all the slot constraints on b. Each map $[\![\_]\!]^b_k$ provides the values of the slots

of object b at instant k. Hence, we are taking the set of the natural numbers as the time domain, making it not only linear, but also discrete and left closed. Given an assignment A of values to variables, these maps are easily extended to terms and formulae of the language TL(β) (b being of sort β) as proposed below. For notational convenience, we use the same symbol $[\![\_]\!]^b_{k,A}$ for these extensions (thus defining a polymorphic map).

$[\![\mathbf{self}]\!]^b_{k,A} = b$

$[\![x]\!]^b_{k,A} = A(x)$ for every variable x

$[\![S]\!]^b_{k,A} = [\![S]\!]^b_k$ for every slot S of β in STA

$[\![f(t1,...,tn)]\!]^b_{k,A} = f_{UNIVERSE}([\![t1]\!]^b_{k,A},...,[\![tn]\!]^b_{k,A})$ for every f in DT + INV

$[\![(t1=t2)]\!]^b_{k,A} = \mathbf{if}\ ([\![t1]\!]^b_{k,A} = [\![t2]\!]^b_{k,A})\ \mathbf{then}\ 1\ \mathbf{else}\ 0$

$[\![(\neg\ P)]\!]^b_{k,A} = I - [\![P]\!]^b_{k,A}$

$[\![(P1 \wedge P2)]\!]^b_{k,A} = [\![P1]\!]^b_{k,A} * [\![P2]\!]^b_{k,A}$

$[\![(\exists x\ P)]\!]^b_{k,A} = \mathbf{if}\ (\mathbf{there\ is}\ A'\ x\text{-equivalent}^{(+)}\ \mathbf{to}\ A\ \mathbf{such\ that}\ ([\![P]\!]^b_{k,A'} = 1))$
$\mathbf{then}\ 1\ \mathbf{else}\ 0$

$^{(+)}$ identical except possibly in the value given to x

$[\![(F\ P)]\!]^b_{k,A} = \mathbf{if}\ (\mathbf{there\ is\ k'\ such\ that}\ k \leq k'\ \mathbf{and}\ ([\![P]\!]^b_{k',A} = 1))$
$\mathbf{then}\ 1\ \mathbf{else}\ 0$

The rest of the connectives, the universal quantification and the tense operator "always" are easily introduced as abbreviations. For instance,

$(G\ P) =_{abv} (\neg\ (F\ (\neg\ P)))$

It is beyond the scope of this short introduction to define other temporal operators, such as those working into the past.

It is now possible to define satisfaction. A formula P of TL(β) is said to be satisfied by the temporal structure $<UNIVERSE, [\![\_]\!]^b_{k:\mathbb{N}}>$ iff

$[\![P]\!]^b_{k,A} = 1$ for every natural k and every assignment A

In this case the structure $<UNIVERSE, [\![\_]\!]^b_{k:\mathbb{N}}>$ is said to be a local model of P. This notion is easily extended to sets of TL(β)-formulae. Hence, the semantics KRIPKE1 of DT + INV +

STA is the $\mathbb{B}$-indexed family of the collections $\mathbb{K}1_b$ of *all local models* of the constraints on the slots.

As an illustration within the setting of UNIVERSE as outlined above for the eating philosophers, consider the following local temporal structures of philosopher p_0. If the slot cond is given values as follows

$[\![\text{cond}]\!]^{p0}{}_k$ = working for every k in $\mathbb{N}$

the constraint (cond=hungry)⇒(F(cond=working)) is satisfied. However, the constraint (cond=working)⇒(F(cond=hungry)) is not. On the other hand, if

$[\![\text{cond}]\!]^{p0}{}_k$ = if k is even then working else hungry

both constraints are satisfied.

In the case of a subtype β of β', the slot constraints in β' are also imposed on every object of type β.

4.3- The Global Space of Events: EVENT

The space EVENT of all possible events is the semantics of the extension EVT + INT to DT + INV. This extension introduces the sort ε of events, the event generator symbols, their categories (birth, death, etc) and the event equations (the interaction equations). The first argument of an event generator symbol is called its distinguished life parameter. Over this signature it is possible to define the language $\mathbf{ETL}(\beta)$ as follows. The terms of $\mathbf{ETL}(\beta)$ are built from the terms of DT + INV with the n-ary event generators on β (n=1,...) taken as (n-1)-ary function symbols (that is to say, omitting the distinguished target argument of type β). In the interaction equations, each term is prefixed by a term denoting the target argument of the generation (still using symbols in DT + INV). Given a term e of $\mathbf{ETL}(\beta)$, an object b of type β (fixing the target argument of the generators) and an assignment A of values to variables, we shall denote by $e^b{}_{EVENT,A}$ the value of e for A generated from b.

EVENT is defined as the free extension of UNIVERSE with respect to EVT + INV. This algebra associates to the sort of events the domain of the equivalence classes of events induced by the interaction equations on the set of all events freely generated with the event generators. Naturally, the reduct of EVENT with respect to DT + INV is UNIVERSE. The set $\mathbb{E}(b)$ of events of the object b is obtained as follows:

$\mathbb{E}(b) = \{[f_{EVENT}(b,a1,...,an)]:$ f being an event generator$\}$

where by [e] we denote the equivalence class of e induced by the interaction equations.

In the sequel, $\mathbb{E}$ denotes the union of all the sets $\mathbb{E}(b)$ with b in $\mathbb{B}$.

Note that, if b is of a subtype β of β', $\mathbb{E}(b)$ will contain every event of the corresponding element in $\mathbb{B}$ of type β' (inheritance).

As an illustration consider the set $\mathbb{E}(p_0)$ of events of philosopher p_0. This set contains the equivalence classes in $\mathbb{E}$ that contain events generated with the event generators declared for objects of type PHIL over the target p_0. Thus,

$$\mathbb{E}(p_0) = \{ [\underline{eos}(p_0)], [\underline{bc\text{-}hungry}(p_0)], [\underline{grab\text{-}ff}(p_0)], [\underline{free\text{-}ff}(p_0)] \}$$

Moreover, the specification of PHIL imposes that, for p_0, $[\underline{eos}(p_0)]$ is of category birth and $[\underline{bc\text{-}hungry}(p_0)]$, $[\underline{grab\text{-}ff}(p_0)]$, $[\underline{free\text{-}ff}(p_0)]$ are of category update.

Note that, according to the interaction equations,

$$[\underline{free\text{-}ff}(p_0)] = \{ \underline{free\text{-}ff}(p_0), \underline{dw\text{-}by\text{-}rph}(f_0), \underline{dw\text{-}by\text{-}lph}(f_1) \}$$

Hence, this element of $\mathbb{E}$ also belongs to $\mathbb{E}(f_0)$ and to $\mathbb{E}(f_1)$.

Finally, consider the events of objects of subtype WORKING_PHIL of the type PHIL. Let w_0 be such an element. Then, assuming that p_0 is the element of type PHIL corresponding to w_0,

$$\mathbb{E}(w_0) = \mathbb{E}(p_0) = \{ [\underline{eos}(p_0)], [\underline{bc\text{-}hungry}(p_0)], [\underline{grab\text{-}ff}(p_0)], [\underline{free\text{-}ff}(p_0)] \}$$

because no further events are declared in the specification of the subtype: every event of w_0 is inherited. Moreover, the specification of WORKING_PHIL imposes that, for w_0, $[\underline{eos}(p_0)]$ and $[\underline{free\text{-}ff}(p_0)]$ are of category **becomes**; $[\underline{bc\text{-}hungry}(p_0)]$ is of category **begoes**.

4.4- The Local Valuation Maps: VALUATION

The maps that provide, for each object occurrence, the values of the slots after each possible sequence of events, in compliance with the valuation and reduction rules, constitute the semantics of the extension of DT + INV + EVT with respect to Σ_{STA} + VAL, where Σ_{STA} is the signature of the specification STA of the slots (that is to say, STA minus the state constraints).

Given an assignment A of values to the variables and fixing an object b of $\mathbb{B}$, of type β,

such a local map $[\![\_]\!]^b_s$ with s in $\mathbb{E}(b)^*$, is easily extended to terms and formulae of the language $\mathbf{EL}(\beta)$: the first order language on the signature of DT + INV enriched with the slots of β taken as 0-ary function symbols, the terms of $\mathbf{ETL}(\beta)$, the constant symbol **self** of sort β and the predicate **after** on events. Moreover, if β is a subtype of some β', then $\mathbf{EL}(\beta)$ is further enriched with the slots of β' taken yet again as 0-ary functions. Note again the polymorphism of the map $[\![\_]\!]^b_{s,A}$ which also extends to events as follows:

$[\![\mathbf{self}]\!]^b_{s,A} = b$

$[\![x]\!]^b_{s,A} = A(x)$ for every variable x

$[\![S]\!]^b_{s,A} = [\![S]\!]^b_s$ for every slot S of β in STA

$[\![f(t1,...,tn)]\!]^b_{s,A} = f_{UNIVERSE}([\![t1]\!]^b_{s,A},...,[\![tn]\!]^b_{s,A})$ for every f in DT + INV

$[\![f(t1,...,tn)]\!]^b_{s,A} = [f_{EVENT}(b,[\![t1]\!]^b_{s,A},...,[\![tn]\!]^b_{s,A})]$ for every f in EVT

$[\![(t1=t2)]\!]^b_{s,A} = \mathbf{if}\ ([\![t1]\!]^b_{s,A} = [\![t2]\!]^b_{s,A})\ \mathbf{then}\ 1\ \mathbf{else}\ 0$

$[\![\mathbf{after}(e)]\!]^b_{s,A} = \mathbf{if}\ s = (s'^{\wedge}\langle[\![e]\!]^b_{s,A}\rangle)$ for some s' **then** 1 **else** 0

$[\![(\neg P)]\!]^b_{s,A} = 1 - [\![P]\!]^b_{s,A}$

$[\![(P1 \wedge P2)]\!]^b_{s,A} = [\![P1]\!]^b_{s,A} * [\![P2]\!]^b_{s,A}$

$[\![(\exists x\ P)]\!]^b_{s,A} = \mathbf{if}$ (**there is** A' x-equivalent$^{(+)}$ **to** A **such that** $([\![P]\!]^b_{s,A'} = 1)$)
then 1 **else** 0

$^{(+)}$ identical except possibly in the value given to x

Notice that in the case of terms in $\mathbf{ETL}(\beta)$, the maps $[\![e]\!]^b_{s,A}$ do not depend on the trace parameter s. Moreover, $[\![e]\!]^b_{s,A} = e^b_{EVENT,A}$.

The valuation rules constrain these maps (hence, the original maps on the slots). Indeed, a rule of the general form

$\{P\}\ e\ \{Q\}$

where e is a term of $\mathbf{ETL}(\beta)$ and P, Q are formulae of $\mathbf{EL}(\beta)$, imposes the following

$\mathbf{if}\ ([\![P]\!]^b_{s,A} = 1)\ \mathbf{then}\ ([\![Q]\!]^b_{s',A} = 1)$ **where** s' is $(s^{\wedge}\langle e^b_{EVENT,A}\rangle)$

for every sequence s of events and every variable assignment A.

The reduction rules further constrain these maps. For instance, a rule of the form

e1; e2 >> nil

where e1 and e2 are terms of ETL(β), imposes the following, for every slot S of β,

$$[\![S]\!]^b_{s'} = [\![S]\!]^b_{s''}$$

provided that s' is s1^<e1$^b_{EVENT,A}$>^<e2$^b_{EVENT,A}$>^s2 and s'' is s1^s2.

In short, the semantics VALUATION of the extension Σ_{STA} + VAL is the $\mathbb{B}$-indexed family of collections of maps $\lambda s[\![\_]\!]^b_s$. Please note that such a collection may contain more than one map (in case of incompleteness) or may be empty (in case of inconsistency).

As an illustration of VALUATION, consider the valuation rule

{ pay = N } free_ff { pay = (N+1) }

that constrains every local map of a philosopher p as follows:

$$[\![pay]\!]^p_{s^\wedge<[\underline{free\text{-}ff}(p)]>} = [\![pay]\!]^p_s + 1$$

In the case of a subtype β of β', the valuation and reduction rules in β' are also imposed on every object of type β.

4.5- The Local Event Structures: KRIPKE2$^\natural$ and KRIPKE2

The semantics of the extension PRC to DT + INV + EVT + VAL is given through two $\mathbb{B}$-indexed families KRIPKE2$^\natural$=($\mathbb{K}2^\natural_b$)$_{b:\mathbb{B}}$ and KRIPKE2=($\mathbb{K}2_b$)$_{b:\mathbb{B}}$ such that each $\mathbb{K}2_b$ is the set of *all possible* local event structures of the form

<UNIVERSE,T^b,($[\![\_]\!]^b_{T^b(k)}$)$_{k:\mathbb{N}}$>

that satisfy all the *universal birth/death constraints* (discussed below) and all the safety constraints on b, and $\mathbb{K}2^\natural_b$ is the subset of $\mathbb{K}2_b$ that consists of the local event structures that further satisfy all the liveness requirements on b. $\mathbb{K}2_b$ corresponds to partially correct trajectories, and $\mathbb{K}2^\natural_b$ corresponds to totally correct ones.

By T^b we denote a trajectory of b: a mapping from $\mathbb{N}$ to $\mathbb{E}(b)\cup\{\perp\}$ such that if $T^b(k)=\perp$, then $T^b(k+1)=\perp$. Moreover, its counterpart in terms of traces, is as follows:

$\mathbb{T}^b(0) = \langle\rangle$

$\mathbb{T}^b(k+1)$ = **if** $T^b(k)=\perp$ **then** $\mathbb{T}^b(k)$ **else** $(\mathbb{T}^b(k)\hat{}\langle T^b(k)\rangle)$

A local event structure $\langle UNIVERSE, T^b, (\llbracket \_ \rrbracket^b_{\mathbb{T}^b(k)})_{k:\mathbb{N}_\infty}\rangle$ is said to satisfy the universal birth/death constraints iff

$T^b(0)$ is of category birth or becomes;
$T^b(k)$ with $k>0$ is not of category birth or becomes;
if $T^b(k)$ is of category death or begoes, then $T^b(k+1)=\perp$

The safety constraints on b have the following general form:

$\{\{ P \}\}$ e

where P is a formula of $\mathbf{EL}(\beta)$ and e is a term of $\mathbf{ETL}(\beta)$.

A local event structure $\langle UNIVERSE, T^b, (\llbracket \_ \rrbracket^b_{\mathbb{T}^b(k)})_{k:\mathbb{N}_\infty}\rangle$ is said to satisfy the safety constraint above iff, for every assignment A,

$T^b(k)= e^b_{EVENT,A}$ **implies** $\llbracket P \rrbracket^b_{\mathbb{T}^b(k),A}=1$ for every k

The liveness requirements on b are built from the terms in $\mathbf{ETL}(\beta)$ with the disjunction and conjunction operators on goals: | and &, respectively.

Given an assignment A, a local event structure $\langle UNIVERSE, T^b, (\llbracket \_ \rrbracket^b_{\mathbb{T}^b(k)})_{k:\mathbb{N}_\infty}\rangle$ satisfies an atomic liveness requirement e iff there is a k such that

$T^b(k)= e^b_{EVENT,A}$

It satisfies a disjunctive liveness requirement g1|...|gn iff it satisfies at least one of the gi (i=1,...,n) for that A. Finally, it satisfies a conjunctive liveness requirement g1&...&gn iff it satisfies every gi (i=1,...,n) for that A.

A local event structure is said to satisfy a liveness requirement when it satisfies it for every assignment.

Note that in the case of a subtype β of β', the safety constraints and the liveness requirements in β' are also imposed on every object of type β.

From KRIPKE2, the set $\mathbb{T}(b)$ of admissible traces can be characterised as follows:

$\mathbb{T}(b) = \{ \mathbb{T}^b(k)$: $\langle UNIVERSE, T^b, (\llbracket \_ \rrbracket^b_{\mathbb{T}^b(k)})_{k:\mathbb{N}_\infty}\rangle \in \mathbb{K}2_b \}$

Note that T(b) contains only finite sequences of events, thus being less expressive than the set of all admissible trajectories. Indeed, as an illustration, consider the following two cases where SC_b and LR_b denote, respectively, the set of safety constraints and the set of liveness requirements imposed on the object b.

(1) $\mathbb{E}(b1) = \{ c, u, d \}$

where c, u, d are of categories birth, update and death, respectively

$SC_{b1} = \emptyset \quad LR_{b1} = \{ d \}$

(meaning that the liveness requirement of b1 is the guarantee of its death...)

(2) $\mathbb{E}(b2) = \{ c, u, d \}$

where c, u, d are of categories birth, update and death, respectively

$SC_{b2} = \emptyset \quad LR_{b2} = \emptyset$

It is evident that

$$\mathbb{T}(b1) = \mathbb{T}(b2) = \{<>\} \cup \{<c>\hat{}(<u>)^n : n \in \mathbb{N}_*\} \cup \{<c>\hat{}(<u>)^n\hat{}<d> : n \in \mathbb{N}_*\}$$

On the other hand, for instance, a local event structure built with the trajectory

$T(0) = c$
$T(k+1) = u \quad$ for every $k \in \mathbb{N}_*$

belongs to $\mathbb{K}2^{\ell}_{b2}$ but it does not belong to $\mathbb{K}2^{\ell}_{b1}$ (although it belongs to $\mathbb{K}2_{b1}$) because it does not satisfy the liveness requirement.

4.6- Local Correctness: Local Compliance

The compliance of each local event-level specification against the corresponding local temporal-level specification is verified by checking if the former entails the satisfaction of the slot constraints. That is to say:

$$(\forall b: \mathbb{B})(\forall ES: \mathbb{K}2^{\ell}_{b})(\mathfrak{m}(ES) \in \mathbb{K}1_{b})$$

where $\mathfrak{m}$ maps local event structures into local temporal structures as follows:

$$\mathbb{m}(<\text{UNIVERSE},\mathbb{T}^b,(\llbracket\_\rrbracket^b{}_{\mathbb{T}^b(k)})_{k:\mathbb{N}}>) = <\text{UNIVERSE},\llbracket\_\rrbracket^b{}_{k:\mathbb{N}}>$$

where

$$\llbracket S\rrbracket^b{}_k = \llbracket S\rrbracket^b{}_{\mathbb{T}^b(k)} \quad \text{for every slot S of b}$$

As an illustration, the behavioural specification at the event-level of the object type PHIL does comply with the temporal constraint

$$(\text{pay} = N) \Rightarrow (\mathbf{G}((\text{pay} \geq N) = \text{true}))$$

because, according to the valuation rules, the events of any philosopher either do not change the value of pay or increment it.

On the other hand, the behavioural specification at the event-level of the object type PHIL does *not* comply with the temporal constraints

$$(\text{cond} = \text{working}) \Rightarrow (\mathbf{F}(\text{cond} = \text{hungry}))\ ;$$
$$(\text{cond} = \text{hungry}) \Rightarrow (\mathbf{F}(\text{cond} = \text{working}))$$

because the necessary liveness requirements are missing. Indeed, they are stated at the relevant subtypes of PHIL. Hence, they are taken into account only within the setting of the global event semantics. Global compliance is discussed in Paragraph 4.8 below. On the other hand, still at the local semantics, for instance, every local event structure of an object of type WORKING_PHIL does comply with

$$(\text{cond} = \text{working}) \Rightarrow (\mathbf{F}(\text{cond} = \text{hungry}))$$

but not with the other constraint.

4.7- The Global Event Structures: KRIPKE3

The global semantics KRIPKE3 of DT + INV + EVT + VAL is the set of *all possible* global event structures of the form

$$<\text{UNIVERSE},\mathfrak{T},(\llbracket\_\rrbracket^b{}_{\mathfrak{T}^b(k)})_{b:\mathbb{B};k:\mathbb{N}}>$$

that, for each object b in $\mathbb{B}$, are projected into elements of $\mathbb{K}2_b$ when restricted to b and satisfy the universal begoes/becomes constraints with respect to b (see below).

By $\mathfrak{T}$ we denote a global trajectory - a mapping from $\mathbb{N}$ into $\mathbb{E}\cup\{\perp\}$ such that if $\mathfrak{T}(k)=\perp$, then $\mathfrak{T}(k+1)=\perp$. Moreover, in terms of traces, we define $\mathfrak{T}$ from such a $\mathfrak{T}$ as follows:

$\mathfrak{T}(0) = \langle\rangle$

$\mathfrak{T}(k+1)$ = if $\mathfrak{T}(k)=\perp$ then $\mathfrak{T}(k)$ else $(\mathfrak{T}(k)^\wedge\langle\mathfrak{T}(k)\rangle)$

Finally,

$$\mathfrak{T}^b(k)=(\mathfrak{T}(k)\uparrow\mathbb{E}(b))$$

That is to say, for each k, $\mathfrak{T}^b(k)$ is the restriction to $\mathbb{E}(b)$ of the trace $\mathfrak{T}(k)$.

A global event structure $\langle UNIVERSE,\mathfrak{T},([\![\_]\!]^b\mathfrak{T}^b(k))_{b:\mathbb{B};k:\mathbb{N}}\rangle$ is said to satisfy the universal begoes/becomes constraints if, for every $b:\mathbb{B}$ that is an instance of a subtype and for every k, every event of category becomes in $\mathfrak{T}^b(k)$ (except the first one) is immediately preceded by an event of category begoes.

As expected, a restriction to $b\in\mathbb{B}$ of a global event structure

$$\langle UNIVERSE,\mathfrak{T},([\![\_]\!]^b\mathfrak{T}^b(k))_{b:\mathbb{B};k:\mathbb{N}}\rangle$$

is a local event structure

$$\langle UNIVERSE,T^b,([\![\_]\!]^bT^b(k))_{k:\mathbb{N}}\rangle$$

where T^b is a restriction to b of the global trajectory $\mathfrak{T}$ obtained as follows.

Consider the function $\vartheta^b\colon \mathbb{N}\to\mathbb{N}$ defined by

$\vartheta^b(0) = \mu k[\mathfrak{T}(k)\in\mathbb{E}(b)]$
$\vartheta^b(n+1) = \mu k[k>\vartheta^b(n)\wedge\mathfrak{T}(k)\in\mathbb{E}(b)]$

where μ is the minimalisation operator of recursion theory (yielding $\perp$ if no such k exists).

Now we define $S^b\colon \mathbb{N}\to\mathbb{E}(b)\cup\{\perp\}$ as the projection of $\mathfrak{T}$ to events of $\mathbb{E}(b)$:

$$S^b(n) = \mathfrak{T}(\vartheta^b(n))$$

taking $\mathfrak{T}(\perp)=\perp$.

If b is not of a subtype, the unique projection T^b is as follows:

$$T^b(n) = S^b(n)$$

In the case of a subtype, there may exist several projections. Each of those projections T^b of $\mathfrak{T}$ must satisfy, for some i:

$T^b(0) = S^b(i)$ is of category becomes or i=0
$T^b(n+1)$ = **if** $T^b(n)=\perp$
then $\perp$
else if $S^b(i+n+1)$ is of category becomes
then $\perp$
else $S^b(i+n+1)$

It is straightforward to prove, in either case, that T^b is indeed a local trajectory of b (according to the definition in 4.5). Hence, defining $\mathbb{T}^b$ from T^b as before, the envisaged restriction $<UNIVERSE,T^b,(\llbracket\_\rrbracket^b{}_{T^b(k)})_{k:\mathbb{N}_\infty}>$ is indeed a local event structure.

Recall that every element of KRIPKE3, when restricted to b, should be projected, for every object b in $\mathbb{B}$, into an element of $\mathbb{K}2_b$.

4.8- Global Correctness: Global Compliance

Some properties of the societies of objects, such as starvation and deadlock situations, are analysed from a global perspective by looking at the global temporal structures. Consider, for instance, the following definitions:

A **starvation** situation occurs for an object b:$\mathbb{B}$ and a global event structure

$<UNIVERSE,\mathfrak{T},(\llbracket\_\rrbracket^b{}_{\mathfrak{T}^b(k)})_{b:\mathbb{B};k:\mathbb{N}_\infty}>$

iff there is a projection

$<UNIVERSE,T^b,(\llbracket\_\rrbracket^b{}_{T^b(k)})_{k:\mathbb{N}_\infty}>$

in $\mathbb{K}2_b$ but not in $\mathbb{K}2^f_b$ (ie, that fails to satisfy some liveness requirement) and a local trajectory T'^b strictly covering T^b (ie, $T^b \neq T'^b$ and for every k either $T^b(k) = T'^b(k)$ or $T^b(k) = \perp$) such that $<UNIVERSE,T'^b,(\llbracket\_\rrbracket^b{}_{T'^b(k)})_{k:\mathbb{N}_\infty}>$ belongs to $\mathbb{K}2^f_b$.

A **deadlock** situation is said to occur for a global event structure

$<UNIVERSE,\mathfrak{T},(\llbracket\_\rrbracket^b{}_{\mathfrak{T}^b(k)})_{b:\mathbb{B};k:\mathbb{N}_\infty}>$

iff a starvation situation occurs for that global event structure and every object b:$\mathbb{B}$.

KRIPKE3 is said to be necessarily deadlocked if a deadlock situation occurs for every global event structure. KRIPKE3 is said to be possibly deadlocked if there is a global event structure for which a deadlock situation occurs that is not strictly covered by some other global event structure.

Consider the following example, where $\mathbb{B}$ = {b1, b2}:

$\mathbb{E}(b1)$ = { c1, u1, d1 }

where c1, u1, d1 are of categories birth, update and death, respectively. Suppose that, according to safety constraints, d1 is not allowed to occur before the occurrence of u1, and that d2 is not allowed to occur before the occurrence of u2. We have, in consequence, that the admissible traces are given by

$\mathbb{T}(b1) = \{<>\} \cup \{<c1>\hat{}(<u1>)^n : n \in \mathbb{N}_1\} \cup \{<c1>\hat{}(<u1>)^n\hat{}<d1> : n \in \mathbb{N}_1, n>0\}$
$\mathbb{T}(b2) = \{<>\} \cup \{<c2>\hat{}(<u2>)^n : n \in \mathbb{N}_1\} \cup \{<c2>\hat{}(<u2>)^n\hat{}<d2> : n \in \mathbb{N}_1, n>0\}$

If according to interaction equations,

u1 = d2
d1 = u2

it is easy to see that the trajectory of any global event structure in KRIPKE3 must be one of the following:

$<\perp, \perp, \perp, \ldots>$
$<c1, \perp, \perp, \perp, \ldots>$
$<c2, \perp, \perp, \perp, \ldots>$
$<c1, c2, \perp, \perp, \perp, \ldots>$
$<c2, c1, \perp, \perp, \perp, \ldots>$

because, for instance, u1 can only occur after u2 (because u1=d2 and, according to the mentioned safety constraint, d2 must be preceded by u2) and u2 can only occur after u1 (because u2=d1 and, according to the mentioned safety constraint, d1 must be preceded by u1).

If according to liveness requirements d1 and d2 are required to happen, it is easy to see that a deadlock situation necessarily occurs because the projection on, eg, b1 of any of the above mentioned global trajectories is not on $\mathbb{K}2\,\ell_{b1}$ and is strictly covered by the local trajectory

$<c1, u1, d1, \perp, \perp, \perp, \ldots>$

which is in $\mathbb{K}2^{\ell}{}_{b1}$. The same applies with respect to b2. Notice that, according to the definition above, a deadlock situation can only occur in presence of liveness requirements ("those that want something can't get it").

With respect to the example of the the eating philosophers, starvation situations may arise from the liveness requirements on the object types WORKING_PHIL, GRABBING_PHIL and EATING_PHIL. However, it is easy to see that KRIPKE3 is not even possibly deadlocked.

Since starvation is possible, there is no global compliance with, for instance, the following constraint:

$$(\text{cond} = \text{working}) \Rightarrow (\mathbf{F}(\text{cond} = \text{hungry}))$$

Naturally, we could consider only the global event structures without starvation: KRIPKE3^{ℓ}. For those **fair** structures, the global compliance against the temporal specification would be achieved. It should be noted that, in general, the set KRIPKE3^{ℓ} can be empty.

5- Concluding Remarks

Concepts, tools and techniques were provided for the abstract definition of object types, using a primitive language that is being developed as the AOT counterpart to the traditional equational language used for ADT specifications. Seen from this point of view, AOTs look rather complex, but it seems that such a complexity is unavoidable since we are dealing here with the joint behaviour of temporal entities, much more complex than simple values and their operations. For instance, when dealing with interacting objects, behavioural notions such as deadlock have to be considered.

This paper presented the various semantic layers that can be distinguished for the descriptions of societies of objects from an algebraic point of view, ranging from the local semantics of each object description to the global society of objects. Logical calculi are under development for supporting the desired analysis both from the local and the global points of view, capitalysing on previous work on temporal logics for information systems specification and verification [Ser80, CCF82], taking both a linear temporal structure [FS86,FS88] and a branching one [Car85]. Naturally, a basic issue is the detection of deadlocks and the proof of global compliance of the fair structures.

Another important line of research is related to the problems of dealing with complex objects [ESS88] besides those obtained from the specialisation mechanism. Finally, we are

also looking at further developments of the language in order to allow parameterized specifications.

6- References

[Car85] Carmo, J., "The Infolog Branching Logic of Events", **Information Systems: Theoretical and Formal Aspects**, Sernadas, A., Bubenko, J. and Olivé, A. (eds), North Holland, 1985.

[CCF82] Castilho, J., Casanova, M. and Furtado, A., "A Temporal Framework for Database Specification", **Proc. 8th VLDB**, Mexico City, 1982.

[EDG86] Ehrich, H.-D., Drosten, K. and Gogolla, M., "Towards an Algebraic Semantics for Database Specification", **Knowledge and Data (DS-2)**, Meersman, R. and Sernadas, A. (eds), Proc. IFIP WG 2.6 Working Conference, Albufeira, 1986, North-Holland (to appear).

[Ehr86] Ehrich, H.-D., "Key Extensions of Abstract Data Types, Final Algebras and Database Semantics", **Proc. Workshop on Category Theory and Computer Programming**, Pitt, D. et al (eds), Springer-Verlag, 1986.

[EM85] Ehrig, H. and Mahr, B., **Fundamentals of Algebraic Specification I**, Springer-Verlag, 1985.

[ESS88] Ehrich, H.-D., Sernadas, A. and Sernadas, C., "Semantics of Object-Oriented Databases: Complex Objects", 1988 (to be published).

[FS86] Fiadeiro, J. and Sernadas, A., "The Infolog Linear Tense Propositional Logic of Events and Transactions", **Information Systems**, 11[1], 1986.

[FS88] Fiadeiro, J. and Sernadas, A., "Specification and Verification of Database Dynamics", **Acta Informatica**, 1988 (in print).

[Hoa85] Hoare, C., **Communicating Sequential Processes**, Prentice-Hall, 1985.

[Kri63] Kripke, S., "Semantical Considerations on Modal Logics", **Acta Philosophica Fennica - Modal and Many-valued Logics**, 1963.

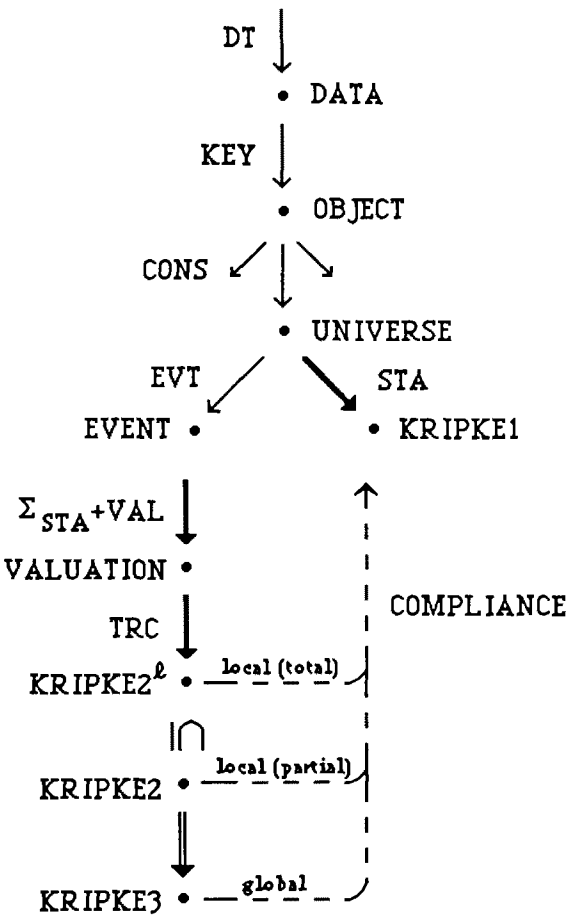
[Pnu77] Pnueli, A., "The Temporal Logic of Program", **Proc. 18th FOCS**, Providence, RI, 1977.

[Pnu79] Pnueli, A., "The Temporal Semantics of Concurrent Programs", **Proc. Symp. on Semantics of Concurrent Computations**, Evian, Springer-Verlag, 1979.

[Ser80] Sernadas, A., "Temporal Aspects of Logical Procedure Definition", **Information Systems**, 5[3], 1980.

[SSE87] Sernadas, A., Sernadas, C. and Ehrich, H.-D., "Object-Oriented Specification of Databases: An Algebraic Approach", **Proc.13th VLDB**, Stocker, P. and Kent, W. (eds), Morgan-Kaufmann Publ. Inc., Los Altos, 1987.

Appendix


DT
DATA
KEY
OBJECT
CONS
UNIVERSE
EVT
STA
EVENT
KRIPKE1
Σ_{STA}+VAL
VALUATION
TRC
COMPLIANCE
KRIPKE2$^{\ell}$
local (total)
KRIPKE2
local (partial)
KRIPKE3
global