Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE
DELLA INFORMAZIONE

PISA

Comparing Formal Specifications Using Temporal Logics: a Case Study on LOTOS and Estelle Protocol Descriptions

Alessandro Fantechi, Stefania Gnesi, Cosimo Laneve

Nota Interna B4-45
Dicembre 1987

COMPARING FORMAL SPECIFICATIONS USING TEMPORAL LOGICS: A CASE STUDY ON LOTOS AND ESTELLE PROTOCOL DESCRIPTIONS

Alessandro FANTECHI* , Stefania GNESI*, Cosimo LANEVE**

* *Istituto di Elaborazione dell'Informazione - C.N.R., Via S. Maria 46 - I-56100 PISA - Italy*
***Dipartimento di Matematica, Universita` di Siena - Via del Capitano 15 - I-53100 SIENA Italy*

Abstract. Temporal logic makes it possible to formally describe systems at a higher abstraction level with respect to formal description techniques based on the specification of possible system behaviour. We thus propose the use of temporal logic in a method for comparing formal descriptions written in LOTOS and Estelle in order to guarantee that they are compatible, i.e. that they cannot be used to generate incompatible implementations.

1. Introduction

A distributed system consists of a number of, normally heterogeneous, computing systems linked by a communication network. The advent of such systems has meant that new problems have emerged and must now be resolved. One important question is how can interactions between processes running on distinct (possibly heterogeneous) systems with very different operating systems be enabled. To resolve this problem the International Organization for Standardization (ISO) has defined a standard architecture to interconnect systems: the Open Systems Interconnection (OSI) model [1]. In this model, in order to reduce the complexity of communication between entities in different systems, every processing node is logically partitioned into functional layers in which *N_layer entities*, using the *N-1_layer functionality* (or N-1_service), will globally provide an N_layer service. For this reason, it must be possible for two N_layer entities to interact. The *N_layer protocol* establishes the rules and agreements used in this interaction.

At their present stage of development, the Open Systems Interconnection (OSI) [1] protocols and services are described by a mixture of natural language (English), graphic representations and state tables (see for example [2]). Generally, the interpretation of a natural language statement is not necessarily unique because a semantics which assigns a unique meaning to it does not exist. The absence of a precise semantics means that a rigorous analysis and verification of the description before attempting an implementation is impossible.

This is particularly damaging when the object we want to specify is a communication protocol, for the following reasons:

- an ambiguous protocol description can lead to incompatible implementations on potentially communicating machines;
- because of their complexity, protocol descriptions may have bugs which, if not detected in the specification phase, may spread to every implementation.

Consequently, in 1979, the ISO organization founded an ad-hoc group with the goal of making formal descriptions of OSI standards possible. Such descriptions should provide a basis for:

- unambiguous and clear specifications of protocol standards;
- verification of specifications in order to find errors and to prove properties;
- functional analysis of specifications in order to detect unintended but correct behaviours of the specifications;
- implementation of a specification and testing of this implementation.

As a result, two Formal Description Techniques (FDTs) have been developed: LOTOS and Estelle.

LOTOS [3], a language based on process algebraic methods and derived from Milner's Calculus of Communicating Systems [4], describes a system by defining the temporal relation between the events that constitute its externally observable behaviour. The language has several operators (sequential composition, non-deterministic choice, synchronized and non-synchronized parallelism, etc.) to shape the temporal ordering of the events. The communication mechanism between a process and the external environment is based upon the general idea of "interaction"; in an interaction, a LOTOS process and the external environment establish together the t-uple of values they want to exchange (synchronous communication).

Estelle [5] is a Pascal-based language with some extensions to encompass parallelism. An Estelle description consists of a set of modules exchanging messages through unlimited queues (the communication is thus asynchronous). Each module is an extended state transition machine which can execute a transition from one state to another when enabled. In addition, the language provides constructs to dynamically create and terminate modules and communications between the modules.

Although the development of both these formal standards has been influenced by the complete trial specifications of some of the OSI protocols and services, it was not possible to derive arguments to objectively choose one of them rather than the other from the experience acquired in writing these specifications.

The existence of two standard formal description techniques for describing the OSI architecture may cause incompatibilities between implementations when formal specifications of the same protocol or service are written using different techniques. After a discussion of these problems, this paper proposes a formal method which can be used to compare LOTOS and Estelle descriptions of the same object in order to detect possible incompatibilities. We consider *temporal logic* as a tool which can specify a (concurrent) system at a higher abstraction level with respect to formal description techniques based on the specification of the possible behaviours. In our method, temporal logic is used to express abstract properties formally derived from the FDT description of a system. Section 4 discusses how to derive temporal logic formulae describing properties of the system from LOTOS and Estelle descriptions. In section 5 we finally show how to use the derived properties to assess the compatibility of a LOTOS and an Estelle description of the same object, discussing the merits and drawbacks of two variants of the method proposed. To show our method a simple example is provided in section 6. Section 7 briefly discusses the possibilities of automating the method proposed.

2. Problems in having two standards

LOTOS and Estelle have been developed simultaneously and so far neither is privileged within the ISO committee. Consequently, in ISO, formal descriptions of communication protocols (at the working draft status) of the same layer (for instance, the transport layer protocol, class 0), can co-exist and there is no guarantee that, when two protocols are implemented on the basis of two different formal descriptions, processes which use these two protocols can communicate correctly.

Generally, the presence of two different formal descriptions of the same object raises the problem of their equivalence, or whether indeed they really do specify the same object. We are not concerned with proving the equivalence between LOTOS and Estelle descriptions (from now on called L and E, respectively) but a weaker property: their *compatibility*. Informally, L and E will be compatible if two processes using two protocols, one implemented from L and the other from E, can communicate successfully.

Assuming that we have a LOTOS and an Estelle description of a given system. Several factors can concur so that the descriptions effectively specify two different objects:

- differences between languages;
- distinct interpretations of the informal description;

To demonstrate this, let us give a simple example.

As we have already stated, communication between LOTOS processes is synchronous, whereas Estelle communication between modules is asynchronous with a boundless queue. This interaction scheme does not permit the "backpressure" control flow to be modeled directly [6]. Backpressure is an implicit control flow mechanism of messages from a user (or client) to a provider (or server), which specifies that the user is blocked from sending other requests to the provider when the communication channel queue between them is full. In this mechanism, there is no explicit communication of control information between the two partners. The presence of a boundless queue between two communicating modules imposed by Estelle means that the "full queue" status is never reached. For instance, if the session protocol wants to send the communication request to the transport protocol, it will perform:

$$\text{output T\_ch.CR(process}_A\text{, process}_B\text{,...)}$$

where T_ch is the channel which links these two protocols. The message will be enqueued in the correct queue without checking the queue status. Referring to the Estelle formal description of the session and transport protocols, every implementation must have a boundless queue between these protocols, even if the implementation language has mechanisms for synchronous communication and for asynchronous communication with bounded queues.

In LOTOS, it is easier to specify backpressure flow control because a user can only send a request to the provider when the latter is ready to interact at the same gate. Consequently, if the provider is busy meeting other requests, it may delay the service request simply by not interacting at the gate in question.

Differences between LOTOS and Estelle descriptions may also result from different interpretations of informal descriptions by the relative specifiers. In fact as the reference description is informal it may be ambiguous and thus two different specificators (one using LOTOS and the other Estelle) may interpret it differently.

The above discussion clearly evidences the need for a method to compare LOTOS and Estelle descriptions. In fact, protocols implemented from two different descriptions may lose their compatibility, thus invalidating both the OSI standard and the formal descriptions themselves.

As LOTOS and Estelle are two different FDTs, descriptions in these languages can be compared in two ways:

1. convert the LOTOS description to an Estelle one and then compare the two Estelle descriptions, or vice versa;
2. convert both LOTOS and Estelle descriptions to other descriptions (in the same language) and compare the two descriptions in this new language.

In order to avoid preferring one FDT to the other we have chosen the second procedure. To cater for the possible different implementation details caused by the differences in the two languages, the third description should be more abstract. For this reason we have chosen to convert both LOTOS and Estelle descriptions to *axiomatic specifications* of the object being described by them. An axiomatic specification is a set of properties, or *axioms*, which qualify implicitly the behaviour of the object. Unlike FDT descriptions (said *behavioural* *descriptions*), as it is at a higher abstraction level, an axiomatic description of a system, stating all its mandatory properties, will not explicitly specify all possible system behaviours.

Temporal logic has been recognized as the most suitable formalism for axiomatically expressing properties of concurrent systems (the class of systems that can be specified by formalisms such as LOTOS and Estelle). This is the reason for choosing this logic for our purposes.

In brief, the method we propose consists in comparing the properties of formal descriptions, properties expressed in a language based on temporal logic and derived formally from the descriptions, and considering them equivalent if the two axiomatic descriptions produced are equivalent.

3. Temporal logic and axiomatic descriptions

Temporal logic provides operators for specifying and reasoning about concurrent programs. It is an extension of Predicate Calculus, using extra operators to reason about future states during the execution of a program. For this purpose, the two temporal operators we will use are: [] (*always*) and ◇ (*eventually*). Informally, in linear time logic [7], if P is an assertion , the meaning of "[]P" is that P is true both in the present state and in every future one; "◇P" means that P is true now or will be true sometimes in the future.

We say that a program satisfies a temporal logic assertion if every possible execution of the program satisfies the assertion. Therefore, we may specify a program by a (finite) set of assertions (each denoting a property) which each program execution must satisfy (axiomatic description). Any other (formal) description which has the same properties (and only those) will be *equivalent* to the temporal logic one. The set of properties characterizing the program behaviour is usually divided into two subsets:

- *safety properties*, which state that something bad never happens, i.e. that the program can never enter into an undesirable state (partial correctness, absence of deadlock, etc.);
- *liveness properties,* which state that something good will eventually happen, i.e. that the program will eventually enter into a desirable state (termination, fairness, etc.).

Verifying that a (formal) description A is an implementation of B is simple enough when we use an axiomatic approach. If $\{A_1, A_2, \dots, A_n\}$ and $\{B_1, B_2, \dots, B_m\}$ are the properties specifying respectively A and B then we say that A is a (correct) implementation of B (or is a lower level description of the system described by B) and we write A«B iff:

$$A_1 \wedge A_2 \wedge \dots \wedge A_n \Rightarrow B_1 \wedge B_2 \wedge \dots \wedge B_m \qquad (1)$$

Intuitively, the reason for this definition is that if A is an implementation of B then every possible behaviour of A must at least have the properties of B; A may also have other properties due to its lower abstraction level. We will use this definition to verify compatibility between a LOTOS and an Estelle description of a communication protocol.

4. Deriving properties from LOTOS and Estelle descriptions

Since we are going to compare LOTOS and Estelle formal descriptions through their respective axiomatic specifications (each consisting of a set of properties) we must define a formal method to derive properties from them.

Safety properties, stating that a property P is always true, are usually expressed through *invariants* (i.e. a predicate that is true in every state of the execution). A formal method for deriving an invariant from a (sequential) program has been proposed by Floyd [8] and is summarized below:

1. establish a set of control points $\{pc_i \,/\, i > 0\}$ of the program;
2. write in every control point an *annotation* (i.e. a predicate);
3. verify the truth of every triplet $\{A_i\}C\{A_j\}$ (consistency check): a triplet $\{A_i\}C\{A_j\}$ is true when control is at pc_i and A_i (the annotation relative to pc_i) is true, and the execution of C leaves control at pc_j where A_j (the annotation relative to pc_j) is true; C is the part of the program included between the two control points pc_i and pc_j.

Therefore, the invariant I for the program is:

$$I = \wedge_i\ (at(pc_i) \Rightarrow A_i)$$

In order to check whether an annotation is consistent, we need an *axiomatic semantics* for the programming language in which the program is written. Such a semantics consists of a set of axioms and inference rules which can be used to infer precisely and formally the effects of each programming language statement on the program state. An axiomatic semantics for a simple sequential programming language was first proposed by Hoare [9]. For example the following Hoare's assignment axiom scheme allows us to derive the truth of the triplet $\{A_i\}C\{A_j\}$ when C is the assignment statement:

Assignment axiom scheme

the triplet $\{P_o\}\ x{:=}expr\ \{P\}$ is valid

where P_o is obtained from P by substituting *expr* for all occurrences of *x*.

Using Hoare's axiomatic semantics one can prove only the consistency of sequential annotated programs. In concurrent programs, because of interactions between concurrent processes (shared variables, message passing, etc.), we must ensure that, in addition to the consistency of the annotation of every single sequential process, any action in one process will invalidate predicates in another (*non-interference check*). Moreover axioms and inference rules must be given for every concurrent statement and construct [10].

Here, for brevity, we will only give the axioms for LOTOS observable actions and Estelle output statements. In a LOTOS interaction *n* processes usually may take part (in the following, for simplicity, we will consider only two processes; the extension to n processes is not particularly difficult); in the same interaction, each of these processes may accept or offer a message. Let us consider two annotated processes, B_1 and B_2, composed by the parallel operator "|g|", where g is a gate and the triples $\{P_1\}g!m\{Q_1\}$ and $\{P_2\}g?v{:}t\{Q_2\}$ are respectively part of B_1 and B_2; if "g!m" and "g?v:t" can cause an interaction, then the interaction itself can be considered equivalent at the assignment statement "*v:=m*". Consequently assertions P_1, Q_1, P_2 and Q_2 must satisfy the truth of the triplet:

$$\{P_1 \wedge P_2\}\ v{:=}m\ \{Q_1 \wedge Q_2\}$$

which can be derived from Hoare's axiom. The axioms and inference rules for other LOTOS constructs are given in [11].

As stated in Section 2, Estelle uses an asynchronous (with unbounded queues) communication mechanism between modules. Consequently, the statement "*output Ip.expr*" is equivalent to enqueuing the value "*expr*" in the queue "*Ip*". Thus the annotation {P}output Ip.expr{Q} is consistent if and only if the triplet

$$\{P\}\ \langle Ip\rangle := \langle Ip,expr\rangle\ \{Q\}$$

is valid, where <Ip> is the queue associated to the Ip interaction point and <Ip, expr> denotes the enqueuing of *expr* at <Ip>. Actually the Estelle output statement axiom is slightly more complex. More detailed axioms and inference rules of Estelle statements are given in [11].

Liveness properties guarantee that in at least one state during the execution of the program something desirable will happen. Therefore, in order to verify liveness properties, we must prove that every state can be reached from the initial state. The method used to derive liveness properties is the same as that described above as far as points 1 and 2 are concerned, whereas point 3 should be substituted by the following:

3'. verify the *liveness* of every triplet $\{A_i\}C\{A_j\}$: a triplet $\{A_i\}C\{A_j\}$ is live if, when executing C from a state in which A_i is true, the execution terminates in a state in which A_j is true, i.e.

$$A_i \Rightarrow \Diamond A_j$$

To verify the latter point, "*live*" axioms must be given for every language statement to prove its termination (*live semantics*) and, therefore, the eventual reachability of a state in which something desirable happens.

The live axiom for the assignment statement "*x:=expr*", if P and Q are two assertions which are true before and after its execution, respectively, is:

$$P \Rightarrow \Diamond Q \qquad (2)$$

i.e., if we are in a state in which P is true then we will reach a state in which Q is true. In other words, the assignment statement "*x:=expr*" will terminate. Particular difficulties may arise when we want to analyze a concurrent program, since the presence of synchronizing primitives may cause a process to remain blocked forever [10]. Here we will only give live axioms for the LOTOS action "g!m" and the Estelle output statement; other live axioms for these FDTs are given in [11].

In LOTOS the action "*g!m*" is only executed when both the process in which it appears and the environment are ready to perform the same action: only in this case can "*g!m*" terminate. Consequently, the live axiom of the above action must consider the non-termination possibility, i.e. the possibility that the environment never offers action "*g<m>*" (*g<m>* is the semantic notation of action *g!m*). If predicate *Ready(g<m>)* is true when the environment is ready to offer action "*g<m>*" and P and Q are two assertions which are true immediately before and immediately after the execution of "*g!m*" respectively, then the live axiom for this action is:

$$P \wedge Ready(g<m>) \Rightarrow \Diamond Q$$

As communication between Estelle modules is asynchronous with unbounded queues, the live axiom of the Estelle output statement is simply expressed by (2), where P and Q are two assertions which are true before and after the execution of this statement, respectively: this means that its liveness does not depend on the external environment.

5. A method to compare LOTOS and Estelle descriptions

The method consists of two phases: the first is the derivation of properties from LOTOS and Estelle protocol descriptions, as shown above; the second is the use of such properties to prove the compatibility of LOTOS and Estelle.

Informally, if two protocols are to result compatible, they must be equivalent with respect to a certain set of properties. Formally we define the *T-equivalence* relation (written: $\approx_T$) between axiomatic descriptions, where T is a set of properties, in the following way:

1: $Pl \approx_T Pl$, i.e. Pl is T-equivalent with itself for every set T of properties;

2: $Pl \approx_T Pe$ iff $Pl « T \wedge Pe « T$ or

$$(Pl_1 \wedge Pl_2 \wedge ... \wedge Pl_n \Rightarrow P_1 \wedge P_2 \wedge ... \wedge P_m) \wedge (Pe_1 \wedge Pe_2 \wedge ... \wedge Pe_k \Rightarrow P_1 \wedge P_2 \wedge ... \wedge P_m) \qquad (3)$$

where $\{Pl_1, Pl_2, ..., Pl_n\}$ and $\{Pe_1, Pe_2, ..., Pe_k\}$ are respectively the sets of properties of axiomatic descriptions Pl and Pe, and $\{P_1, P_2, ..., P_m\} = T$. That is, two set of properties are T-equivalent when both are implementations of T. Note that this relation of T-equivalence is an equivalence relation with the usual properties of reflexivity, symmetricity and transitivity.

Consequently, we define a LOTOS protocol description L to be *compatible* with an Estelle protocol description E if the two axiomatic descriptions formally derived from them are T-equivalent, where T is the *sufficient* set of properties to guarantee that L and E *communicate correctly.* As it is based on the T-equivalence notion, the compatibility relation also has the same properties as this relation. Moreover, the compatibility relation is inherited by implementations; every *correct* implementation of a protocol specification P, having at least the same properties as P, results compatible with P and, consequently, with every other protocol specification Q' compatible with P and with every correct implementation derived from Q'.

The definition of compatibility given above is based on the informal notion of a *sufficient* set of properties to guarantee communication. According to the way that this notion is formalized and set T is built, we can define two variants of our method: the "comparative method" and the "deductive method".

5.1. The comparative method

The first way to formalize the definition of compatibility between two protocol descriptions is to state a priori a third description of the communication protocol, in a language based on temporal logic. As this description consists of a set of properties T, the idea behind this method is to compare the LOTOS and Estelle descriptions against the set T (this is the why the method is called "comparative"), i.e. to verify their T-equivalence.

Summarizing, the comparative method consists in formally deriving the sets Pl and Pe of properties from the specifications and then verifying the implication relations of (3) with a third description given in temporal logic.

Note that in (3) the presence in L and E of different properties (which do not, however, belong to the set $\{P_1, P_2, ..., P_m\}$) does not invalidate their compatibility. In this case, we can consider L and E as different implementations of the same higher level specification T. The verification of the inclusion relation amounts to a proof that both implementations are correct. In addition, the presence of an a priori temporal logic description allows the properties derivation process to be driven from the formal descriptions; one will try to derive from the LOTOS and Estelle specifications only those properties which are present in the temporal logic description.

The main advantage of this method is to guarantee compatibility between formal descriptions on the basis of a "sufficient" set of properties that each description must have. The main drawback is that it imposes the use of another description technique (the temporal logic based language) and, thus, it requires that a system already described in LOTOS and in Estelle must also be described in a third language. Consequently, this method can hardly be accepted within a context in which LOTOS and Estelle descriptions already constitute official specifications.

5.2. The deductive method

The deductive method makes it possible to avoid the "a priori" presence of another formal description in temporal logic. This method consists in deriving a set of properties respectively from LOTOS and from Estelle descriptions (thus the name "deductive" given to the method) and comparing these two sets, which we will call Pl and Pe. Let us call P the *maximal* (with respect to the ordering «), possibly empty, set such that $Pl \approx_P Pe$, then P may not completely specify the system described by L and E. In general, the descriptions may still be completely different with just a few common derived properties, arising, for example, from similar choices made at a lower abstraction level. In the comparative method, the temporal logic description is the reference specification for the communication protocol. In the deductive method there is no reference specification to qualify a sufficient set of properties which L and E must satisfy to communicate correctly. Consequently L and E may not be compatible.

Moreover we have no guidelines in the properties derivation process: i.e. we do not know what properties the protocol *must* have. The informal description may give some assistance either in the property derivation process or in verifying if P (the set of properties which are common at L and E) may be considered as the axiomatic description of the communication protocol specified by L and E. Obviously this is informal reasoning: it is impossible to have the absolute certainty that L and E are compatible. The guarantee that can be given, using this method, is that if the two protocols implemented from the LOTOS and Estelle formal descriptions are found to be incompatible then the incompatibility must not be sought in the properties belonging to P. The impossibility of guaranteeing compatibility is the main drawback of this method, anyway it is useful for a formal comparison when is not desired to refer to a third description in temporal logic.

6. A simple example

In this section we give a (very) simple example which illustrates the proposed technique. We will prove that a LOTOS process and an Estelle module are compatible with respect to the property:

$$[](\text{after\_Proc} \Rightarrow H=<a,b>) \qquad (4)$$

This property can be used to characterize all those processes which, when and if they will terminate, have interacted with the external environment first by the action 'a' and then by 'b'. The interactions should be intended in the sense of LOTOS communication mechanism, i.e. both the process in question and the external environment perform the same action.

The process P is defined, in a LOTOS-like language, by the following behaviour expression:

a; b; stop

The module Q, in a simplified Estelle-like language (in which every module has just an interaction point), is defined by the following body:

```
body                        {of Q}
state s1, s2, stop          {three states for Q}
initialize to s1;           {first state is s1}
 trans from s1 to s2
  when a                    {wait until the environment sends a message a}
   begin output a end       {I agree to perform an 'a'}
       from s2 to stop

  when b                    {wait until the environment sends a message b}
   begin output b end       {I agree to perform an 'b'}
end
```

Note how the LOTOS interaction is simulated through two Estelle asynchronous communications.

The formula (4) expresses a safety property, consequently, in order to prove that P and Q satisfy this property we apply the Floyd method discussed in section 4. The annotation of the process P is:

$$\{H_P=\emptyset\}\ a\ \{H_P=<a>\}\ ;\ b\ \{H_P=<a,b>\};\ stop\ \{H_P=<a,b>\}$$

where the list variable H_P denotes the list of actions performed together by the process and the external environment from the beginning of the execution of P. We omit, for brevity, the consistency checks, which are anyway obvious in this case, but should be done making reference to an axiomatic semantics of LOTOS. The invariant I_P derived from the above annotation is:

$$\begin{aligned} I_P = \ & (at(P) \Rightarrow H_P=\emptyset) \wedge \\ & \wedge\ (at(b;stop) \Rightarrow H_P= \langle a \rangle) \wedge \\ & \wedge\ (after(P) \Rightarrow H_P= \langle a,b \rangle) \end{aligned}$$

(4) is one of the members of the conjunction in I_P, hence it is verified that:

$$[]I_P \Rightarrow [](after(P) \Rightarrow H_P=\langle a,b \rangle)$$

thus the process P has the property expressed by the formula (4).

The annotation for the module Q is:

```
{$H_Q=\emptyset$}
body
state $s_1$, $s_2$, stop
initialize to $s_1$;  {$H_Q^{in} = H_Q^{out}=\emptyset$}
 trans from $s_1$ to $s_2$
  when a
   begin {$H_Q^{in} =\langle a \rangle \wedge H_Q^{out}=\emptyset$} output a end {$H_Q^{in} = H_Q^{out}=\langle a \rangle \wedge H_Q =\langle a \rangle$}
       from $s_2$ to stop
  when b
   begin {$H_Q^{in} =\langle a,b \rangle \wedge H_Q^{out}= \langle a \rangle$} output b end {$H_Q^{in} = H_Q^{out}=\langle a,b \rangle \wedge H_Q =\langle a,b \rangle$}
end
{$H_Q =\langle a,b \rangle$}
```

where the list variables H_Q^{in} and H_Q^{out} express the communications that the module Q has respectively received and sent from the beginning of its execution. The list variable H_Q has an item 'a' if the process Q has received a message 'a' from the environment and has subsequently sent the same message. Again we omit the consistency checks, even if they are a bit less obvious then before. The invariant I_Q that we may derive from this annotation is:

$$\begin{aligned} I_Q = \ & (at(Q) \Rightarrow H_Q=\emptyset \wedge H_Q^{in} = H_Q^{out}=\emptyset) \wedge \\ & \wedge\ (at(s_1) \wedge Exist(a) \Rightarrow H_Q=\emptyset \wedge H_Q^{in} =\langle a \rangle \wedge H_Q^{out}=\emptyset) \wedge \\ & \wedge\ (at(s_2) \wedge Exist(b) \Rightarrow H_Q= \langle a \rangle \wedge H_Q^{in} =\langle a,b \rangle \wedge H_Q^{out}= \langle a \rangle) \wedge \\ & \wedge\ (at(stop) \Rightarrow H_Q^{in} = H_Q^{out}=\langle a,b \rangle \wedge H_Q =\langle a,b \rangle) \wedge \\ & \wedge\ (after(Q) \Rightarrow H_Q =\langle a,b \rangle) \end{aligned}$$

where 'Exist(x)' is true if the queue associated to the module is not empty.

Again, (4) is one of the members of the conjunction in I_P, hence it is verified that:

$$[]I_Q \Rightarrow [](after(Q) \Rightarrow H_Q=\langle a,b \rangle) .$$

Thus Q has the property expressed by the formula (4), and P and Q are compatible with respect to the property (4).

7. Automation possibilities

Our method can be applied to two generic formal descriptions to prove their equivalence (more precisely, their compatibility) provided we have the axiomatic and live semantics of the respective languages. If this method is to be applied to objects of the complexity of the OSI communication protocols we need to consider whether it can be automated.

The fact that the method is structured in steps, each exploiting a different technique, suggests that a similar architecture can be implemented for a semiautomatic system to support the method in the variants of Sections 5.1 and 5.2. However, some of the steps are not algorithmic: both in Floyd's method for deriving an invariant and in that used to prove the reachability of a state that verifies a property, the association of an annotation to every control point is not algorithmic. Finding the most suitable assertion for every control point in order to derive a particular property from a program implies a complete knowledge of the program itself. Furthermore, the equivalence problem between first order temporal logic formulae is a semidecidible problem, i.e. an algorithm may not terminate a query asking whether two formulae are equal. This means that the semidecidibility of proving two formal descriptions equivalent also persists when checking weaker equivalences, such as the T-equivalence defined above and compatibility between protocols.

We will now analyze a possible architecture for the different steps of the system and for each of them we will examine the tools already available. The input to the system is a pair of LOTOS and Estelle descriptions for the same object (L and E respectively).

STEP 1. This step establishes a set of control points for the specifications L and E. Control point identification is easily made automatic: we can take as the control points, the points immediately before and after every language construct. More precisely, in LOTOS, control points are placed before and after each action. A syntactic analyzer can identify the beginning and end of every construct. This tool is already available both for LOTOS (SCLOTOS, LOTOS Syntax Checker, produced by the University of Madrid) and for Estelle (developed at INRIA) [12].

STEP 2. Derivation of two sets of properties $\{Pl_1, Pl_2, ..., Pl_n\}$ and $\{Pe_1, Pe_2, ..., Pe_k\}$ from L and E, respectively. According to the Section 4, this step consists of two points which are repeated for every property we want to derive from L and E:

• The association of a suitable assertion to every control point. As already stated, this is non-automatic. We can reduce this point to find invariants which express safety properties. A method for this is proposed in [13]. This method works using a set of assertions which is gradually refined to produce the invariant for the program in examination. The first set of *candidate* assertions forming an invariant is given by the programmer (or verifier) and is originated from an informal description of possible input values, from specification of output values and from comments which are present in the text. Note that these assertions may not be correct (thus the attribute "candidate" given to them). This method will eliminate the inconsistent one. Likewise proper assertions must be written to derive liveness properties. We know of no methods which have been developed for this purpose.

• The consistency and liveness check. As far as consistency (in the invariant derivation) and statement termination (for deriving a state reachability) are concerned, verifying tools can be developed using axioms and inference rules from the axiomatic and live semantics of the language so that, taking the triplet $\{A_i\}C\{A_j\}$ as input, they can decide consistency or liveness. See [14] for a discussion of a system which can check the consistency of Pascal annotated programs.

STEP 3. In this step the sets $\{Pl_1, Pl_2, ..., Pl_n\}$ and $\{Pe_1, Pe_2, ..., Pe_k\}$ of properties derived, from L and E respectively, are compared (with a third description in temporal logic for the comparative method, or another description in the deductive one). Effectively, the comparison consists in verifying the logical equivalence of pairs of properties in the sets. Since the properties are expressed in first order temporal logic formulae, the equivalence problem is tackled using tools and methods of first order logic, (see [15] for a survey of existing theorem provers).

Finally, we want to mention a completely different approach which may be used to apply the comparative method to descriptions with a finite number of states: this is CESAR [16]. This automatic tool, developed in the ESPRIT/SEDOS project, makes it possible to verify that a (LOTOS or Estelle) description A is an implementation of a temporal logic description B (i.e. A«B). For this purpose, description A is translated with exhaustive simulation in a functionally equivalent transition system. The principle used for verifying in CESAR is the following:

> *a description A is an implementation of a temporal logic description* $B = \{B_1, B_2, ..., B_m\}$ *if for every state s of the corresponding transition system every axiom* B_i *is true.*

Therefore, safety and liveness properties are checked by evaluating assertions B_1, B_2, ..., B_m directly on transition system states without using the methods discussed in this work (annotations, consistency or reachability checks, etc.).

Actually CESAR accepts descriptions in Hoare's CSP as input, therefore LOTOS and Estelle descriptions must be translated into descriptions in this language. Ad-hoc translators are been developed for this purpose.

7. Conclusions

In this paper, we have discussed the problems which are raised when we have two descriptions of the same system in different formal languages, examining a particular standardization environment (ISO/TC 97/SC 21). The risk of having incompatibilities between protocols implemented from different descriptions is tackled by comparing Estelle and LOTOS descriptions against a more abstract specification, i.e. a temporal logic one. This comparison is made using a method which integrates several formal techniques and which can be partially automated.

We believe that the method which we propose is applicable to objects of the complexity of an OSI protocol or service only if an automated support tool is provided. Two critical points, when automating the two phases of the method we have outlined, are not algorithmic:

1. the derivation of an invariant in Floyd's method and the proof of the reachability of a state that verifies a property consist in associating an annotation to every control point: finding the most suitable assertion for every control point in order to derive a particular property from a program implies a complete knowledge of the program itself;
2. the implication problem between first order temporal logic formulae is in general a semidecidible problem, i.e. an algorithm may not terminate a query asking whether a formula implies another.

This is enough to show that the support system cannot be completely automatic, but should be seen as a set of tools which aid the user in his decisions.

Considerable work is still needed for the definition and development of tools to be able to effectively use some of the techniques involved and for the integration of such tools in an environment which would support our comparison method.

References

[1] Zimmerman, H., OSI Reference Model: The ISO Model of Architecture for Open Systems Interconnection, IEEE Trans. on Comm. Vol. COM-28, No. 4 (1980).

[2] ISO 8073, Information Processing Systems - Open Systems Interconnection - Connection Oriented Transport Protocol Specification, (1986).

[3] ISO , LOTOS, a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, Second Draft Proposed Standard 8807 (1986).

[4] Milner, R., A Calculus of Communicating Systems, Lecture Notes in Computer Science No. 92 (Springer Verlag, New York, 1980).

[5] ISO, Estelle: A Formal Description Technique Based on an Extended State Transition Model, Second Draft Proposed Standard 9074 (1986).

[6] Vissers, C.A., Scollo, G., Formal Specifications in OSI, IBM Europe Seminar Networking in Open Systems, Oberlech, Austria, August 18-22, 1986.

[7] Lamport, L., "Sometimes" is sometimes "Not Never", a Tutorial on the Temporal Logic of Programs, in Proceedings of the 7th ACM Symposium on Principles of Programming Languages , Las Vegas (1980).

[8] Floyd, R.W., Assigning Meaning to Programs, Proceedings Symposium on Applied Mathematics 19, Providence R.I. (1967).

[9] Hoare, C.A.R., An Axiomatic Basis for Computer Programming, Communications of the ACM, Vol. 12, No. 10, (1969).

[10] Lamport, L., Schneider, F.B., Formal Foundation for Specification and Verification, in: Paul, M., Siegert, H.J. (eds.), Distributed Systems, Lecture Notes in Computer Science vol. 190, (Springer-Verlag, New York, 1985).

[11] Laneve, C., Un Metodo per il Confronto di Specifiche Formali di Protocolli di Comunicazione , Tesi di Laurea, Universita` di Pisa (1987).

[12] Diaz, M., Vissers, C., Budkowski, S., Estelle and LOTOS Software Environments for the Design of Open Distribuited Systems, in: ESPRIT '87, Achievements and Impact (North-Holland, Amsterdam, 1987).

[13] Good, D.I., London, R.L., Bledsoe W.W., An Interactive Program Verificator System, IEEE Trans. on Software Enginering. Vol. SE-1, No. 1 (1975).

[14] Manna, Z., Waldinger, R., Studies in Automatic Programming Logic (North-Holland, Amsterdam, 1977).

[15] Lindsay, P.A., Moore, R.C., Ritchie, B., Review of Existing Theorem Provers, University of Manchester, Technical Report UMCS-87-8-2 (1987).

[16] Fernandez, J.C., Richier, J.L., Voiron , J., Verification of Protocol Specification Using the CESAR System, in: Protocol specification, testing and verification 5 (North-Holland, Amsterdam, 1985).