KFKI-1983-98

J. HARANGOZÓ
P. ECSEDI-TÓTH
P. TŐKE

NEW RESULTS ON COMPUTER COMMUNICATION II. PROTOCOL MODELS

Hungarian Academy of Sciences

CENTRAL RESEARCH INSTITUTE FOR PHYSICS

BUDAPEST

2017

KFKI-1983-98

NEW RESULTS ON COMPUTER COMMUNICATION
II. PROTOCOL MODELS

J. HARANGOZÓ, P. ECSEDI-TÓTH*, P. TŐKE**

Department of Process Control, Technical University,
Budapest, Hungary

*Hungarian Academy of Sciences
Research Group for Automata Theory
Szeged, Hungary

**Department of Numerical Methods and Computer Science
Eötvös Loránd University, Budapest, Hungary

HU ISSN 0368 5330
ISBN 963 372 145 8

CONTENTS

STATE OF THE ART OF PROTOCOL DESIGN IN THE EARLY EIGHTIES

J. HARANGOZÓ

Department of Process Control, Technical University,
Budapest, Hungary

ABSTRACT

The paper analyses the process of design of computer network protocols, introduces the most recently elaborated formal specification methods, evaluates their capabilities from the viewpoint of formal verification. It raises some problems to be solved in the future.

АННОТАЦИЯ

В статье анализируется процесс проектирования протоколов ЭВМ, вводятся новейшие формальные методы спецификации протоколов и дается их оценка с точки зрения требований формальной спецификации. Приводится несколько проблем, требующих решения.

KIVONAT

A cikk elemzi a számitógéphálózat-protokollok tervezési folyamatát, bevezeti a legujabb formális protokollspecifikálási módszereket és értékeli azokat a formális specifikáció kivánalmainak szempontjából. Felvet néhányat a jövőben megoldandó problémák közül.

INTRODUCTION

Formal specification and verification of protocols, i.e. protocol design by formal means, have seen significant development in recent years. As a consequence of the wide ranging activity under the guidance of the International Standardization Organization (ISO) a reference model of information processing systems has been developed which is the basis for defining and specifying the architecture of computer networks. This international cooperation has resulted in the following new results:

(1) formal means are increasingly used to define services and to specify protocols

(2) new software supporting means have been developed to facilitate the application of these methods of specification and definition

(3) encouraging expreiments are in progress to apply the means of program verification in protocol verification

(4) a process has started which attempts to construct correct protocols with mathematical logics.

If one examines the direction of developments in formal designing methods two tendencies can be observed. One of them follows the old "classical" idea according to which formal specification and verification have to be carried out together to obtain a fully correct protocol satisfying every requirement. The aim is understandable and clear but several obstacles stand in the way of its realization, first of all from the viewpoint of verification. The completely correct verification of a complex protocol is still an unsolved problem though there are encouraging experiments in the verification of simple protocols.

The other new trend follows from taking the present realities into account. The primary aim here being to reach better lucidity, readability in the formal specification of protocols and their easy applicability during implementation. The verification is carried out by not really strict mathematical means but rather by simple logical consequences.

The research connected with the elaboration of the ISO reference model has also influenced significantly research into local area networks. The architectural and protocol principles, methods worked out for the general computer networks can be used for the local area networks too.

DESIGNING WITH FORMAL MEANS

The multilevel hierarchical structure of the ISO reference model provides the design of every level separately. Using the bottom-up construction method the design of the architecture is carried out from the physical to the application level. The basic principle is that while using the services of the lower levels the actual level performs its own data transmission tasks as well as providing, services to the above level. The "black box" concept has two consequences from the viewpoint of designing of the examined level:

(1) the services for the given level must be defined by the lower level without the knowledge of any element of the given level about the internal structure or operation of the lower level

(2) the rules of interaction between the entities of the given level must be specified, i.e. the protocols. To specify these protocols a knowledge of the services provided by the respective level to the upper level is indispensable. These services are of determining character with regard to the protocols of the given level. The interaction between the entities of the given level is carried out to realize some functional task in order to serve the entities of the upper level.

From the viewpoint of the architectural description of a given level it is completely satisfactory to define the used services and to specify the protocols, but this is not enough from the viewpoint of designing the respective level because a knowledge of the requirements of this level is indispensable since for this to be done the definition of services provided for the upper level is required.

Thus protocol designing is the activity requiring knowledge of the whole reference model including the overall definition of services used by the different functional levels and that of the provided services, and the description of entities of the functional levels, and that of the rules of interaction between them, i.e. the specification of the protocol of the given level. To prepare a protocol that really satisfies the characteristics described in the specification the ready plans have to be verified as a final phase of designing. Consequently the process of protocol design consists of the following phases:

(1) Informal description of the required and provided services and the protocol.

Either our ideas are written or not, the designer's first thoughts about the system are informal. In order to understand the expectations and requirements of the system to be designed a global concept has to be developed that is independent of the formal means the designer intends to use.

(2) Choice of formal means to be used for designing.
The designer chooses the formal means suitable for specifying, solving and controlling the task. This is the most critical phase of protocol design using formal means. It is necessary to choose or elaborate such mathematical means that are simple and clear and that help the difinition, specification and verification or testing as well as the implementation. The simpler the means the less the likelihood of mistakes during realization and the more probable it is that the implemented version of the plans reflects the original ideas of the designer.

(3) Formal definition of services, formal specification of protocols.
In this phase a formal image, an abstract description, a model of the system to be designed is developed. It is the phase requiring the most work, the most thorough grounding, the most ingenuity.
To construct the abstract description the ISO reference model serves as a basis. During the creation of one level the description of the required and proved services contains the information exchange between the model its environment while the protocol specification contains the entities of the level and the rules and the processes of information exchange between them. The close link between the chosen formal means and the constructed model becomes apparent in this phase. The capability of the model as a result of the formal description is greatly determined by the chosen means. A very clear model can be realized with the means based on a state transition machine but the operation to be carried out in certain states and the description of data structures connected with them are not satisfactory. The latter problem can be solved for modelling by using a programming language, at the same time the natural character appreciated at modelling based on a state transition machine is lost. It is for this reason that the correct choice of formal means or the elaboration of suitable means is of fundamental significance from the viewpoint of the utility of the model.

(4) Checking of ready plans.
The last phase of designing is the checking of ready plans, i.e. to make sure whether the thoughts, solution in the plans really reflect the original ideas of the designer or whether the system really uses the services determined in the definition and ensures the defined services for it. The only solution that eliminates every fault is verification. It is this method that is able to prove whether the designed protocol really possesses the given characteristics included in the formal specification. Full or partial verification in possible depending on whether the examination is carried out for every feature in the specification, or whether for the control of only a few important or provable ones. Several theoretical and practical difficulties can occur in connection with verification that make the examination of the full system impossible in the case of complicated protocols.

The other method controlling the ready plans is testing. This method is suitable for finding the most common faults but a system found to be faultless after testing cannot be safely regarded as faultless. In spite of this its practical significance is great as it is perhaps the control means that can be used the easiest and the most efficiently.

In the following we introduce the most recently elaborated solutions in the field of protocol design using formal means and it is these that are expected to be used extensively in the future.

DEFINITION, SPECIFICATION

1. Formal Definition Technique (FDT)

The working group of ISO TC97/SC16/WG1 founded an ad hoc subgroup to work out some formal description means; this resulted in the subgroup publishing its elaboration of a specification language known as the Formal Definition Technique (FDT) [ISO81].

This is suitable for describing the protocols and services using an extended finite state machine model (FSM) and the PASCAL programming language.

The system to be described contains modules (entities) connected to each other. Each module is an extended finite state automaton. A module is in permanent interaction with its environment. The extended finite state automaton differentiates between these so-called elementary events depending on whether the initiative of the event is the environment (inputs) of the model or the model itself (outputs). An elementary event (input) initiated by the environment causes a state transition in the model whose consequence may be another elementary event (output) initiated by the model.

Information exchange between two modules is realized by queuing, i.e. the output information of one module gets into a waiting queue before it would appear as the input information of the other module.

To fix the sequence of the elementary events the model has internal state space that determines the possible state transitions of the model and thus its connection with its environment at every moment. The designer has to give the state space of all transitions and the set of possible transition. With complicated protocols the definition of an automaton becomes difficult as the number of internal states significantly increases. That is why a new definition method using a programming language instead of the usual finite automaton was elaborated. The difinition of the state space of a model is carried out by a set of variable. A possible state of this space can be characterized by the values of these variables. For example:

```
VAR STATE:(IDLE,WAIT_FOR_CC,WAIT_FOR_T_ACCEPT,DATA_TRANSFER);
    .
    .
    .
```

The definition of possible state transitions is carried out by the specification of state transition types. Every state transition type is characterized by an enabling condition (Boolean expression) and an operation. The value of thus Boolean expression depends on the variable defining the states of the module and the input elementary event. The operation is carried out as a part of the state transition, it can change the value of the variables and can specify the output of the elementary event.
The elements necessary to describe a state transition are:
(1) enabling condition:
- actual state (after *FROM* clause)
- input elementary event (after *WHEN* clause)
- further enabling condition (after *PROVIDED* clause)
- priority of a state transition (after *PRIORITY* clause)

(2) operation belonging to a state transition:
- marking the next state (after *TO* clause)
- effect of operation, generation of output event
  (block after *BEGIN*).

The finite automation of there states can be described in the following way:

```
(*TRANSITION*)
FROM A
  WHEN AP.REQ1
    PROVIDED C1
      TO B
      BEGIN
        ACTION1;
        AP.IND1
      END;
    PROVIDED C2
      BEGIN
        ACTION2;
        AP.IND2
      END;
WHEN AP.REQ2
  TO C
  BEGIN
    ACTION3;
    AP.IND3
  END;
```

The model itself is non-deterministic as in a given state (at a given moment) during a given elementary input event several different state transitions are possible but only one transition is carried out. But this carrying out is not determined by the specification of the model itself.

To describe the relation between a module and its environment it is necessary to introduce the concept of "channel".The services provided by the lower level can be used through access points. Every point corresponds to a channel that without the description of entities can be perfectly characterized by listing the primitives (elementary information units) flowing through them. The language makes possible the introduction of data type *INTERACTION* that can be defined by the designer to describe the channel within which the primitives flowing out and it can be listed together with its parameters.
For example:

```
INTERACTION
  TS_ACCESS_POINT(TS_USER,TS_PROVIDER)
  BY TS_USER:
    T_CONNECT_REQ(...);
    T_ACCEPT_REQ(...);
    T_DISCONNECT_REQ(..);
  BY TS_PROVIDER:
    T_CONNECT_IND(..);
    T_ACCEPT_IND(..);
       etc.
```

The introduction of a new data type module makes possible the listing of all the channels used by the module (ENTITY), i.e. it can give the units (PRIMITIVES) of information exchange between the module and its environment and the rules of its sequences. In addition, the global sequence restriction (affecting every module that determine the sequence of information units going through the different channels of the module can be given too (TRANSITIONS). In the case of service specification it means that these restrictions define the relation of the interaction at the end points of the connection (output to input or vice versa). In the case of protocol specification the restriction specify the sequence in which the different data units can be sent to the lower level.

Consequently, the service and protocol specification is a PASCAL language description that contains:

(1) the channel and the primitives belonging to them (*INTERACTION*)

(2) the specification of one or more modules with conventional declaration elements (label, constant, type, variable, procedure, function) and declaration elements necessary to describe the entity (major state declaration, state set definition, initialization, transition).

A description of the Transport Protocol and the descriptions of the Virtual File Server and Virtual File Access Protocol are to be found in [LEV82] and [BOCH82A, BOCH82B, BOCH82C].

2. Formal Specification Technique (FST)

Within the framework of the development of the ISO reference model a formal specification technique was elaborated in 1981 utilizing the financial support of the National Bureau of Standards (NBS) [BLU81].

The formal description is based on an FSM supplemented by variable where the state transition of the automation is marked by program segments. The description examines one level of the ISO reference model in its environment. A given level is regarded as one lying between two neighbouring levels; in addition, every level is surrounded by the operating system. The examined level consists of modules (entities) that communicate with each other according to some protocol while using the services of the lower level. Every module that can be considered as an automaton generally has three interfaces: one towards the lower level, one towards the upper level and one towards the oparating system. A well defined set of events can be mapped to every interface that corresponds to the set of service primitives. A state transition of the automaton is carried out after the realization of an interface event (input) and a fulfilment of other internal conditions. During a state transition the variables of the automaton get new values and the automaton can initiate interface events (output). The automaton has predetermined initial and final states. Instead of the description with a state transition table or diagram the representation of a state transition of an automaton is realized through substitution rules (production) using the relation between finite state automata and regular languages. Its greatest advantage is that the description is near to the computer and it can easily be processed.

To support formalism a partial subset of PASCAL language is used supplemented by some special constructions. The description of a state transition and that of semantic characteristics belonging to it are demonstrated by the following example:

```
<DEST> → <ORIGIN> [FROM A: SERVICE.REQUEST]
                  ((P1>EXPR1) AND (P2=EXPR2))
          BEGIN
            (*PROGRAM SEGMENT TO EXPRESS SEMANTICS
              BELONGING TO STATE TRANSITION*)
            VAR1:=[FROM A:P3]+1;
            IF VAR2=[FROM A:P2] THEN
               [TO A: SERVICE.RESPONSE (P1:=VAR1)]
          END;
```

That is the automaton gets into the DEST state if it was in the ORIGIN state and a service request primitive arrives from the interface marked a the parameters P1, P2 of which satisfy the relation in brackets (enabling conditions). Under the effect of the a state transition, variable VAR1 takes the value of primitive parameter P3 incremented by 1 - in this case it is an indifferent parameter - and if the value of P2 is equal to the value of variable VAR2 then an output interface event comes to interface a (SERVICE. REQUEST primitive) the parameter P1 of which takes the value of variable VAR1.

Besides the regular PASCAL declaration elements the formal specification contains special declaration elements: primitives, predicates, interfaces and states declarations. Apart from this the specification contains a description of the protocol data units of the bit level described by a linving language.

The declaration of primitives is performed in the same way as the declaration of *FUNCTION* and *PROCEDURE* in PASCAL. In order to describe the effect of primitives an explanation in living language is to be attached to the formal description.

The predicates are defined after the clause *PREDICATE* as a procedure. The body can contain only Boolean expressions. In its effect a *PREDICATE* can be regarded as a function that gives back its Boolean value but the *PREDICATE* does not allow side effects. The definition of service primitives contains the name of the service primitives and the parameters attached to them.

The definition of a special data type *MACHINE* can also be found in every specification that is in fact a record whose fields are local variables belonging to the state transitions of the automaton.

3. Specification Language

To specify distributed systems within computer networks a Specification Language SPEX was developed [SCHWA81].

The basis of this language is a non-deterministic state transition model of the system to be specified that has many special characteristics concerning the specification.

A system can be regarded as the set of connected nodes that can be a station or a transfer medium. The definition of a given level is created by the interaction of nodes. The sample sequences in the interaction characterize the type of node. In general, a system can consist of different nodes of special behaviour. In order to characterize a system the behaviour of every type of node is to be written (it will be the "NODE BEHAVIOUR" of specification). Further on, the set of possible elements of every type of node has to be given and the way in which they are connected (it will be the "TOPOLOGY" part of the specification) and the characteristics of the interactions between them (this will be the "CHARACTERISTICS" part of the specification).

A node is some kind of entity that has internal variables and external interface variables. These variables may arbitrarily be complex data types. The node reacts to definite events. If such an event takes place some state variables and some interface variables change their values.

State variables can be reached only locally within the nodes. Interface variables are divided into two parts in every node: the group that will be exported to other systems and the one that will be used within the given system for connection with other nodes. Moreover every interface has a direction too marking whether the data flow into or out of the node.

The actual behaviour of the node is described by giving its reaction to events determined in advance. A precondition belongs to every well-known event that is in fact a predicate containing the state and interface variables of the given node. As this precondition becomes true the event belonging to it takes place.

The behaviour of the node is given by the new value of all its variables taken after the fulfilment of all the possible events. All changes belonging to an event take place simultaneously, at one time i.e. these events are regarded as elementary (atomic) events.

The initial state is also necessary to be given to the whole description of the behaviour of the node. It means that the initial value of the variables is to be given while creating the system. All of them have to be specified for the node types in the system.

The behaviour of the whole system is defined by all the valid sequences of the events. Such a valid sequence where the system starts from the initial state then enabled events follow each other can be of endless length. If it is of finite lenght a not-enabled event causes the final state of the system.

After giving all the node types the connection mode of the nodes has to be specified too. It can be done by the interface variables that serve the connection with other nodes. They are in fact divided variables among the affected nodes. The "TOPOLOGY" part thus defines the connection of the interface variables of the nodes to the interface variables of other nodes.

The "CHARACTERISTICS" part of the specification deals with two characteristics of the protocol, viz. with "supposed" and "declared" characteristics. The declared characteristics are to be proved by the one performing the specification and it serves for the subsequent control of the accuracy of specification. On proving these characteristics it increases and proves the credit of the specifying person and shows that the specification is based on understanding of the system. Supposed characteristics are operations defined by non-programming language-means, that are given by the output-input relations between the arguments and the obtained values.

The specification of the Connection Establishment Brotocol of the ARPA network was carried out using the SPEX language in [SCHWA81]. Some elements of SPEX were used in [KOV82].

4. Temporal Logics

Recently, the formal specification, and verification of computer programs and protocols were attempted by mathematical logics - within this temporal logics. The research started from the assumption that the modal logics of Kripke is able to describe the events after one another and the conditions of their happening. The protocols that can be regarded as a set of sequential events can probably also be defined by this form of logics [SCHWA81, HAIL80].

The logical expression of the requirements of the protocol (protocol characteristics) are described in the form of axioms in the temporal logics. The basic operators in these expressions are: operator "diamond" (possible) and operator "box" (necessary). In addition, there are more complicated operators built from these primitive operators (e.g. until, until-after, etc.). Predicates at, in, etc. in the expressions mark the control points.

The description regards the protocol as a multiprocessor system running on only one processor system and generates logical expressions of the expected characteristics for this model. Thus the description has to specify the process constituting the system, the input-output and control points of processes.

An experiment to describe a data link level protocol using temporal logics can be seen in [TOTH82]

VERIFICATION

Verification methods are usually related to specification methods. Regarding the development tendency a dual movement can be observed in this field: one of these is the traditional specification in accordance with the known principles and whose strict formal verification contains every detail; the other one is the specification followed by partial formal or informal verification.

FDT and FST specification methods help the implementation on the protocol. Because the basis of both methods is an extended final state automaton their verification can be carried out by reachability analysis. However, extension leads to several problem which is why no formal examination containing every details has been elaborated for any of them. In one application of FDT [BOCH82A], the simple logical consequence method or informal means are used to prove the correctness of the protocol. With regard to FST it is known that there exists no formal verification system.

The requirements of strict verification emerges during the application of SPEX and temporal logics. SPEX has such means by using the AFFIRM system worked out to prove the correctness of programs. AFFIRM is an experimental system that makes possible the algebraic specification and verification on the characteristics of data types defined by the user. The basis of the system is a natural deduction theorem proving algorithm that proves in an interactive way the characteristics of data types that are started in the form of predicate calculi. PASCAL programs that contain the data type defined by the user can be verified by the method of inductive substitution. AFFIRM is able to verify short programs only. In accordance with this the protocol specification and verification in SPEX was successful only for simple protocols.

Similar demands can be seen in temporal logical design too but the practical applications of the method still requires further research work.

At the moment FDT and FST are the two methods that are nearest to reality from the viewpoint of their practical realization. Their lucidity and their easy reading significantly ease the preparation of the right specification followed by the application of the specification in implementation. As the practical examples show, they have been efficiently used to specify the Transport and Virtual File Transfer Protocol of an ISO reference model [BLU81, BUR81, BOCH82A, BOCH82B, BOCH82C].

PROBLEMS

In spite of the undisputed development in protocol designing several problems remain unsolved.

There is fundamental disagreement in the syntactic and semantic definition of service and the functional capabilities of protocols. This means that it is highly complicated to prepare protocols for both the designers and users and to reach their identical interpretation.

The convertability of formal descriptions remains unsolved. It is difficult to decide whether a specification is really the one we want and whether it is equilvalent to the wirtten one. The evaluation of certain descriptions is difficult, i.e. it is difficult to decide which of the several descriptions of the same protocol is the better choice. It is difficult ot decide wherher the different specifications really specify equivalent protocols.

These problems undoubtedly hamper important developments in applications and the long-term utilization and will determine the direction of further research.

REFERENCES

ISO81	A FDT Based on an Extended State Transition Model. ISOTC97/SC16/WG1 Working draft, Boston, Dec. 1981.
LEV82	Leveile, A., Bochmann, G.V.: Formal Specification of a Transport Protocol. Working draft, 1982.
BOCH82A	Bochmann, G.V., Henckel, L.P., Zeletin, R.: Formalized Specification and Analysis of a Virtual File System. General Description. HMI Report, No. HMI-B 367, Feb. 1982.
BOCH82B	Bochmann, G.V.: Specification of a Virtual File Server. HMI Report, No. HMI-B 367, Feb. 1982.
BOCH82C	Bochmann, G.V.: Specification of a Virtual File Access Protocol. HMI Report, No. HMI-B 367, Feb. 1982.
BLU81	Blumer, T.P., Tenney, R.L.: A Formal Specification Technique and Implementation Method for Protocols. NBS Report, No. ICST/HLNP-81-15, July. 1981.
SCHWA81	Schwabe, D.: Formal Specification and Verification of a Connection-Establishment Protocol. USR Report, No. ISR/RR-81-91, Apr. 1981.
KOV82	Kovacs, L.: Formal Specification and Verification of Computer Network Protocols. Doctoral dissertation, Dept. of Process Control, Technical University, Budapest, 1982 (in Hungarian)
BUR81	Burrus, J. et al.: Specification of the Transport Protocol. NBS Report, No. ICST/HLNP-81-1, 1981.
HALL80	Hailpern, B., Owicki, S.: Verifying Network Protocols Using Temporal Logics. NBS Trends and Application Symp. May 29, 1980, pp. 18-28
TOTH82	Toth, P.: Protocol Verification Using Temporal Logics. (in preparation)

TOWARDS A GENERAL THEORY OF PROTOCOLS

P. ECSEDI-TÓTH

Hungarian Academy of Sciences
Research Group for Automata Theory
Szeged, Hungary

ABSTRACT

The author shows that a computer communication protocol can be modelled - i.a. - by an appropriate mathematical logical language. The modal logic, as appropriate basis of model is introduced.

АННОТАЦИЯ

Авторы показывают, что протоколы сети ЭВМ представляют собой серию предложений, излагаемых на соответствующем языке математической логики. Вводится модальная логика, как соответствующая математическая логика, которая может быть основой такой языковой модели.

KIVONAT

A szerző megmutatja, hogy a számitógéphálózat protokollok megfelelő matematikai logikai nyelven fogalmazott mondatok sorozataként tekinthetők. Bevezeti a modális logikát, mint megfelelő matematikai logikát, amely egy ilyen nyelvi modell alapja lehet.

CHAPTER I. LOGICAL APPROACH - AN INTRODUCTION

1. TRADITIONAL VERSUS LOGICAL CONCEPTS OF PROTOCOLS

According to the standard definition, a protocol is a set of rules goverining the communication between components in a distributed computer network; these rules ensure reliable transmission through unreliable channels. This definition, however, tells nothing about what a protocol is from a mathematical point of view; i.e. about the mathematical model of a protocol. Knowing only the traditional definition, it is impossible for one to choose any well-defined mathematical objects which are the models of protocols. To promote the possibility of choice, additional assumptions must be made. One of these assumptions, undoubtedly valid for any protocol, is that the rules have to be compiled in a language. Usually three types of languages are chosen: natural languages (e.g. spoken English [39,40], computer languages (e.g. PL/I [7,8][14], concurrent PASCAL [5], FAPL [64], etc.) and mathematical ones. The use of natural languages arises because of their relatively easy comprehensibility and partly because of the lack of more adequate tools. With regard to computer languages, these are recognized as being fairly well-known, and their use considerably supports implemantation. Nevertheless, these two types of languages are not mathematical objects on their own and their exact semantics can be given (if this is indeed possible) in a completely implicit way (i.e. by using other mathematical languages), hence they are inadequate for defining protocols from a semantic point of view. It this remains for one to coose a mathematical language, notwithstanding that not all mathematical languages suffice.

A finer analysis on which to base this choice is the following argument. The main reason for searching for an exact mathematical concept of a protocol is not merely to learn its semantics (i.e. its specification), but it is also to know the essential properties of the protocol at issue not given explicitly in the specification; for instance, whether its behaviour is defined in all possible (correct or incorrect) situations. In particular, one wishes to derive certain properties from other ones; i.e. the language chosen must provide appropriate tools, e.g. a formal inference system, to deal with such derivations. A formal mathematical language augmented with an inference system is called a logic (cf. Section II). Hence our main assumption is:

A protocol is a set of sentences in an appropriate mathematical logical language.

Beyond the fact that this assumption is important for developing a coherent theory of protocols, it is in line with the traditional concept (see Chapter II); the new definition is nothing but a more formal and more exact reformulation of the old one.

2. OVERVIEW OF THE GENERAL STRUCTURE OF LOGICS

To clarify the very notion of a logic we survey here some of the fundamental concepts of a logic in general (and in particular of two-valued logics) - with which most verificators are presumed to be familiar. The aim of this section is to fix the terminology used in the rest of the paper.

By a logic, we mean a pair (L,C) where L is a logical language (i.e. a formal language with semantics) and C is an inference system, the calculus of the logic.

The finer structure of language L can be given as a quadruplet L=(F,M,I,T), where F is a set, the set of formulae; M is a class, the class of models; T is an algebra, the algebra of truth-values; and I is the interpretation, i.e. I is a mapping from FxM into T. (Note that M is not a set but a proper class, thus I is not a function in the set-theoretical sense.) Actually, the members of class M are exactly what we wish to describe and study by the formal tools of logic; they can be defined directly by using set-theoretical means (cf. Chapter II. for some particular definitions). The members of F are generated by a formal grammar from a set of symbols fixed in advance. This set of symbols must contain some signs which identify the things in the members of M (these signs are usually named "non-logical symbols"), moreover some additional signs, too (the "logical symbols"). In fact, formulae serve as assertions about the models in M. The interpretation I establishes a connection between models and formulae by telling to what degree of truth a particular formula is valid in a certain model. The algebra T of truth-values is generally a kind of lattice (for details, see [22]). In most situations T is a two-membered Boolean algebra; and the logics obtained by this choice of T are called two-valued logics. Here we shall restrict ourselves to two-valued logics. (Although the usage of many-valued logics would result in a finer analysis of protocols, the theory would be more complex and less comprehensible.) Since two-membered Boolean algebra is unique (up to isomorphism) we shall not introduce it explicitly in our notations; i.e. we shall write L=(F,M,I) understanding that the set of truth--values contains the elements TRUE and FALSE, only, enriched by the well--known logical operations: $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\rightarrow$ (if...then...), etc.

In this particular case, the interpretation I can be considered simply as a relation between F and M: $I \subset F\times M$. If $(\varphi,A)\in I$, then, allowing some temporary ambiguity in terminology (eliminated later, see Ch.III.), φ is said to be true in A, otherwise it is false in A ($\varphi\in F$, $A\in M$); in other words, I tells if some formula φ states a true or false fact about A. We shall use the more common symbol $\models$ instead of I writing it in an the infix way: $A\models\varphi$ stands for $(\varphi,A)\in I$. The set of all formulae which are true in every model will be called the set of universally true formulae, denoted by U.

Calculus C of the logic is a formula-manipulating algorithm in character and can be defined in several different ways. The so called enumerating calculi are given in the standard recursion theoretical way; they are designed to enumerate some or all elements in the set of universally true formulae. If a calculus C enumerates nothing other than elements of U, then it is called correct. If C is able to enumerate the whole set U, then it is called complete. According to these definitions, a logic is correct (or sound), and complete if its calculus is such. A more commonly used type of calculus is defined by distinguishing some (usually not all) universally true formulae, the set of axions, and some formula-transformation rules, the so called inference rules. This kind of definition of calculus presupposes that we are able to decide whether or not an inference rule is applicable to a particular formula. If each inference rule possesses the property that the result of the transformation is universally true provided its input is such then the calculus is called correct. If the whole set U can be generated by applying the inference rules arbitrarily but only finitely many times to axioms and to the formulae obtained in previous steps then we say that the calculus is complete. Similar definitions apply to logics.

It should be mentioned that correctness of calculi is very important: if a calculus is not correct then it is totally unsuitable for any purpose. On the other hand, although highly desirable, completeness of calculi is not essential: if a calculus is not complete then certain (true) assertions cannot be proved. This situation is obviously not the best but can be inproved by using other, more appropriate (correct) calculi.

Some particular calculi and their properties are treated in Chapter IV.

3. LOGICAL CONCEPT OF A PROTOCOL

Let $L=\langle F,M,\models\rangle$ be a (two-valued) mathematical logical language and let (L,C) be a logic over L. Then, by our main assumption (in Section 1) a protocol P is a subset of F: $P\subset F$. The class M of models is restricted by protocol P: only those elements of M are considered in which every element of P is true.

$$M_P = \{A \mid A\in M \text{ and } A\models P\}$$

where $A\models P$ stands for $(\forall\varphi\in P)\ (A\models\varphi)$.

A complete specification of protocol P is then achieved: P contains the formulae corresponding to the set of rules (in the traditional sense) and M_p defines the semantics (i.e. the possible realizations) of P in an explicit and abstract way. Different properties of protocol P, such as freedom from deadlocks (liveness) and partial correctness (safety) can be expressed again by formulae of L, i.e. a property φ of P is an element of F. The verification of φ means that one deduces φ from P using the inference system C of the logic.

To our knowledge, no papers dealing with the verification of protocols have even mentioned the role played by the inference system of the underlying logic; in fact, they all use implicitly only one inference rule: the rule of mathematical induction. It will be explained below that this lonely rule, albeit it is one of the most important rules, cannot serve as a satisfactory tool for protocol verification nor for understanding the very nature of protocols. In closing this section, we wish to emphasize the main point of our approach literally, too:

> <ins>The only adequate tool for the specification and verification of protocols</ins> (as well as of any other involved dynamic phenomena/ <ins>is logic</ins>; i.e. the use of a mathematical logical language endowed with an inference system.

4. LINKS WITH OTHER APPROACHES

To fill the gap between the traditional inexact definition of a protocol and the mathematical accuracy required for its verification, researchers admit different mathematical objects as models of it. The aims of this section are
(i) to analyse these choices from a logical point of view,
(ii) to investigate whether they can, at least in principle, be inserted into the overall picture outlined above.

Usually, the following objects are accepted as models of protocols: finite state machines (or more precisely, composition of a few Mealy-type automata [6,17-18,61,71], graphs (such as Petri-nets [52], timed Petri-nets [52], evaluation graphs [55], UCLA-graphs [55,73-74] etc.), computer programs [5,9-12,34], formal languages [33,35-37,63,68-69], and so on.

All of these concepts are well-suited to the logical approach. For example, if elements of M, i.e. the models, are automata or graphs, then the definition of a particular protocol means simply, that we give a concrete element in M (i.e. one particular automaton or graph) and idenfiy the protocol at hand with this element. Notice, however, that this identification is not legitimate from a strict mathematical point of view. There can be other elements in M which are equivalent to the one chosen - as far as the properties of the protocol in question are concerned - but which differ in other respects; these equivalent elements of M are realizations of the same protocol, and ignorance of them can cause essential errors. This undersiced phenomenon is occasioned by the fact that, except some very simple examples, protocols can be dexcribed by infinite tools, only. (Recall that a simple function, such as addition over natural numbers, is an infinite object.) From the verificational point of view, however, an automaton or a graph must be finite. Choosing a concrete finite or infinite element of M and identifying the protocol at issue with it is inadequate because if the chosen element is finite, then essential features of the protocol remain unmodelled while if it is infinite, then the verification cannot (by direct checking of the possibilities) be carried out.

To overcome this difficulty, one must identify the protocol not with one single model but the class of all equivalent models and treat this infinite class by other finitary tools, provided by logics. (Exactly the same situation occurred in the theory of verification of sequential programs: the choice of a single model to represent all possible runs of a program resulted in properties of incompleteness [1-4][29-30][58].)

Other approaches that identify protocols with sets of sentences over a computer language or a formal language (in the sense of Chomsky) are again well-sented to the logical approach. In these cases, a protocol is a subset of the set of formulae of the underlying language. They do not bother with realizations (i.e. with elements of M), hence the difficulties mentioned above are avoided: nevertheless, a complete semantic specification of a protocol is missing if realizations (models in the sense of logic) are not involved.

Summing up, all known methods for specifying and verifying protocols realize only a part of the specification provided by the logical approach.

At the same time, however, logics have a finitary tool, the inference system, by which the infinitary logical specification of a protocol can appropriately be handled (provided that the logic is complete and correct). As we have already mentioned, inference systems are implicit in the literature on protocol verification. Of course, if one verifies something arbitrarily, then one uses an inference system even if it is hidelen in arguments. In fact, one inference rule, the rule of mathematical induction, can be abstracted from the literature. This rule is applied in the form that if some properties are true in all initial situations and they are inherited by transitions of situations then these properties are true in all situations. By this lonely rule, however, only invariance properties can be proved; others, such as some eventuality ones, so vital for the correc functioning of protocols, cannot. At the same time, careless use of this rule can lead to difficulties, namely incompleteness.

5. CHOICE OF THE UNDERLYING LOGIC

The problem of finding the appropriate language and the logic which are expressive enough for formalizing protocols and their properties but are still tractable as a mathematical object is of secondary importance from the protocol implementer's and user's point of view. Even so, it is still a challenging theoretical task. Actually, any logic reducible in the sense of Lindström to the classical first order logic will do; such logic differ "only" in the way of use and cost of application. This study investigates in detail the impacts of several possible choices of language L. In fact, besides the traditional first order logic, we shall deal with different kinds of dynamic logics, modal and temporal logics and logics of actions. These logics share many features when employed to specify and verify protocols but they differ also in some important respects. The similarities and differences are analysed in Chapter II.

6. ADDITIONAL PRINCIPLES ON THE NATURE OF PROTOCOLS

Protocols, in comparison with simple (i.e. sequential) programs, may exhibit extremely involved behaviour. Central to the discussions on protocols are the concept of a state and that of an action. We recall here the basic notions and explain informally some of their properties relevant to obtaining a concise theory of protocols. (For more about actions see [20-21],[72].) To justify the terminology, we mention that a state can be imagined as one particular variant of the "world" to be described and an action may result in a change of states. In any state, several actions can be performed: these are called "enabled" or "permitted" relative to the state. Some of the enabled actions may be parallely executable or mutually interdependent or exclusive; also there can be obligatory or prohibited actions enabled relative to a particular state. Performances of actions may cause side-effects in several distinct states, they may last for more than one instant of time, and this time of execution may vary depending on actions and states.

The overall picture is then rather complex and gives rise to many intricate problems; some of these will be touched on while others, not of principal interest to the theory of protocols, will be completely neglected.

Bearing in mind the aim of this paper we can make some essential assumptions which will have considerable effects on the tractability of the whole theory. Let us start by assuming that the Law of Excluded Miracle is valid. That is, we shall always suppose that any change in states is caused by some actions from the set of actions given in advance. This law seems to be counterintuitive for modelling complex dynamic phenomena such as complex programs which communicate with each other through unreliable channels since physical or logical errors can cause "miracles". At a more general level, however, the possible errors can also be considered as actions, hence admitting the Law is legitimate and helps to keep the theory extensive. Similar approaches were adopted in [15] and [37].

Another important assumption is the validity of the Law of Relative Permanency. By this we mean that the universe of discourse is relatively permanent which, in turn, means that no action may cause birth or death of individuals in this universe.

Finally, we shall assume that acting agents are hidden; that is, performances of actions are controlled from "outside" by some acting agents such as human beings, computers or other. One of the simplest ways of realizing this assumption is to suppose that all actions permitted relative to some state are triggered at the moment they become enabled. This will be supposed throughout the paper unless the contrary is explitly expressed.

7. RAMIFICATIONS WITHIN THE LOGICAL APPROACH

If one adopts the logical approach for protocol specification and verification, further refinements can be made depending on what aspects of the functioning of protocols are considered the most important for modelling. If one looks at a protocol layer as a black box which gives replies to effects arriving from upper layers (i.e. from the user of its own side) or from lower layers (i.e. from the opposite stations) then one describes the functioning of that layer from a component-oriented point of view. Admitting this kind of description, we do not deal with such questions as what happens in the communication channels, i.e. how many messages are on route, etc. (cf. Fig. I-1).

Another possible standpoint is when one concentrates on the communication itself neglecting what happens in the different components. This way of reasoning will be called communication-oriented description (cf. Fig. I-2).

Both kinds of reasoning have several advantages and drawbacks and, in general, they represent an "ideal" or "purely theoretical" method. Real protocols can adequately be described only by some merging of the two approaches.

In this paper both kinds of descriptions as well as their mergings will be investigated. It is mentioned, in closing, that these two kinds of descriptions are distinguishable in the literature on protocol verification: e.g. Danthine and Bremer [17], Merlin [52], Bochman [7],[9] and others adopt the component-oriented description of protocols; Harangozó [35-37], Teng and Liu [68-69], Hoare [43] and Ecsedi-Tóth [19] describe protocols in a communication-oriented way.

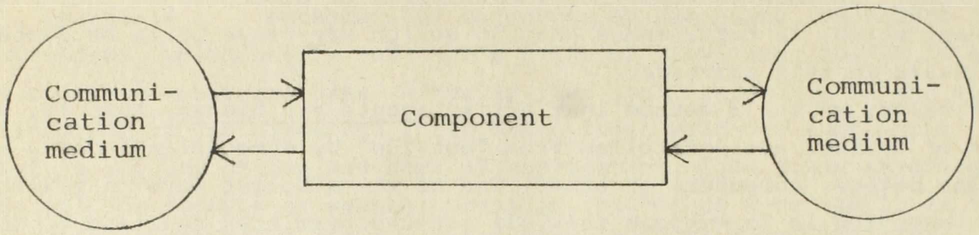


Fig. I-1

Component-oriented description

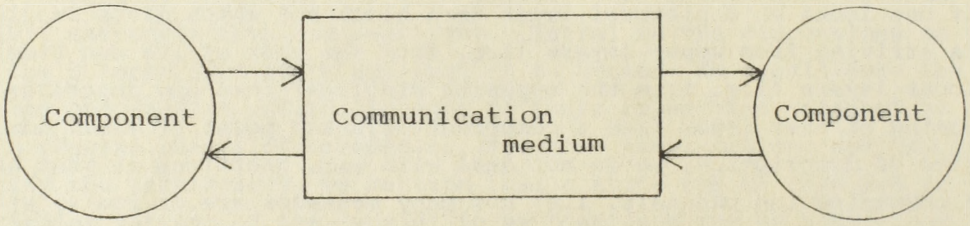


Fig. I-2

Communication-oriented description

CHAPTER II. MODELS OF PROTOCOLS

The concept of models plays a central role in developing logics since they represent the mathematical counterpart of the phenomena to be studied. Here we introduce two different kinds of models. We are mainly concerned with nonclassical models; classical ones will be used as auxiliary tools.

1. CLASSICAL AND NONCLASSICAL MODELS

Our meaning of a classical model is a nonempty set, the universe of the model, with some distinguished elements (constants) in it and with functions and relations defined on the universe. No assumption is made on the number of constants, functions and relations (there may be none or infinitely many of them) but we tacitly assume that each function or relation can have only a finite number of arguments. Almost all objects investigated in mathematics are classical models. For example, every group, ring, Boolean algebra, universal algebra (and thus, in particular, every Mealy-type automaton, graph), vector space, metric space, topological space, projective geometry and so on, will give an instance of classical models. The classical model consisting of the set {0,1,...} with 01 as constant and with the well-known functions and relations: +, ., $\leq$, etc. usually plays an important role in applications of logic.

For modelling dynamic phenomena which evolve in time, classical models can be used but with considerable difficulty because they lack the intuitive counterparts of time moments and the flow of time. Fortunately, in more general types of models, all important aspects of time considerations can be modelled in a natural way. These models are known as nonclassical models. Generally speaking, a nonclassical model can be constructed from a classical one by substituting the individual elements in the universe of the latter by other classical models, i.e. a nonclassical model is a classical one the individuals of which have an "inner structure", cf. Fig. II-1 and Fig. II-2.

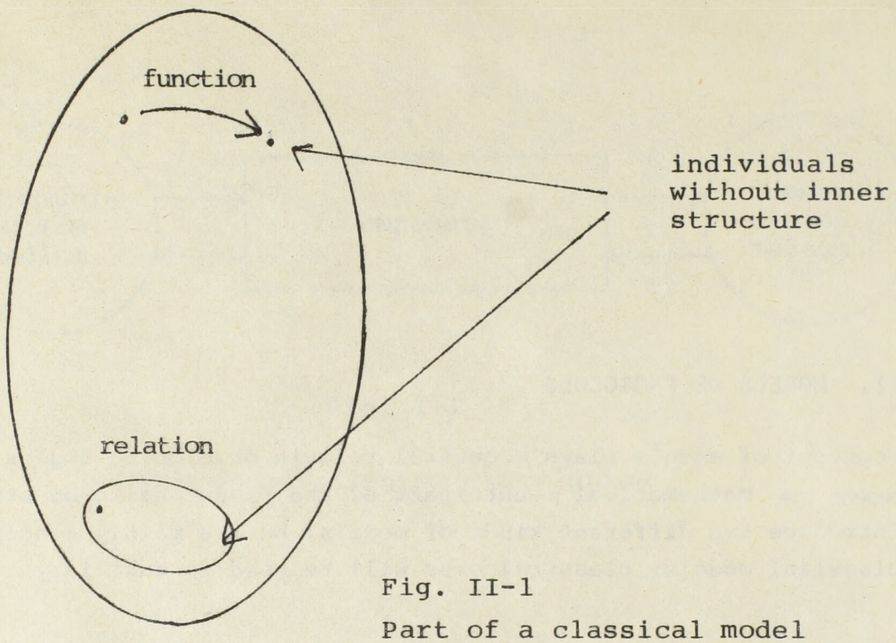


Fig. II-1

Part of a classical model

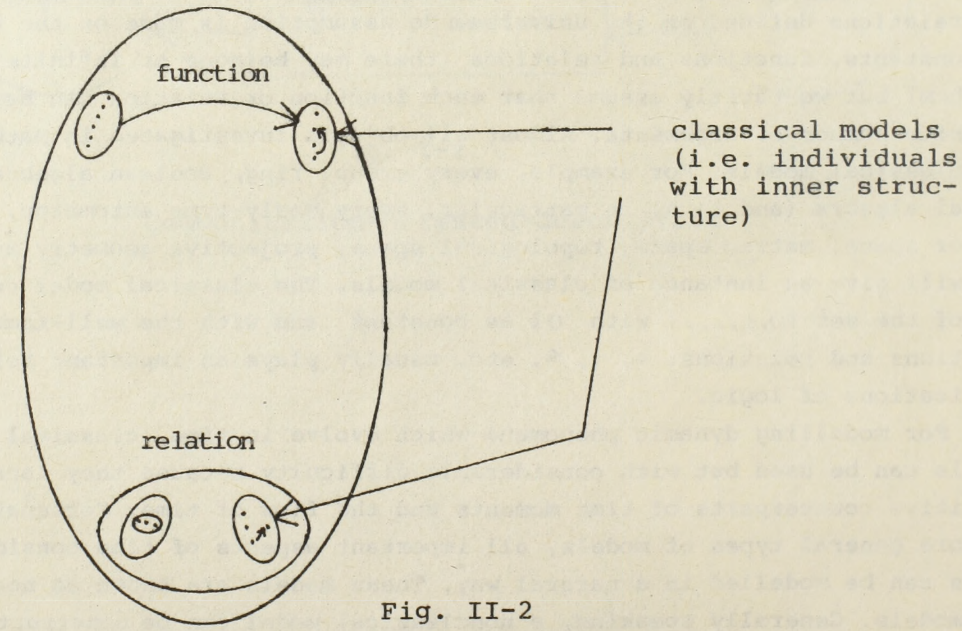


Fig. II-2

Part of a nonclassical model

The full power of this concept is not be used in this paper; instead we introduce a particular case, the notion of Kripke models, which is much more tractable, possesses a well-developed theory and still has power enough for our aims.

By a Kripke model we mean a pair (Q,R), where Q is a set of classical models such that all classical models in Q have the same uninverse (cf. Ch.I,6) and R is a binary relation Q, that is, $R \subseteq Q \times Q$. Sometimes we shall use distinguished elements in Q but no functions or relations except R are defined on Q is this paper. Members of Q are called states and R is denoted as the accessibility relation. One can imagine that states are "snapshots" on the phenomenon in question whereas R describes the "flow of time". Several additional assumptions can be made on the accessibility relation R; thus, several different Kripke models can be used. Some examples are: "R is reflexive", "R is transitive" or both (i.e. "R is a preordering"); "R is dichotomous, (i.e. for all $q_1,q_2 \in Q$, either $(q_1,q_2) \in R$ or $(q_2,q_1) \in R$); "R is discrete". It is believed that the examples given in the next three sections illustrate well the usefulness of Kripke models in specifying and verifying protocols.

2. EXAMPLES: AUTOMATA AND PETRI-NETS

To motivate the definitions above and to give a feeling of the connection between the concept of Kripke models and that of other kinds of mathematical objects used for modelling dynamic phenomena (including models for protocols), we shall reformulate here in terms of classical and Kripke models two well-known examples: automata and Petri-nets. Other examples are given in the following two sections.

By the standard definition (cf. [ ]), a Mealy-type automaton is a quintuple

$$\mathcal{A} = (A,X,Y,\delta,\lambda)$$

where A,X, and Y are nonempty sets; $\delta: A \times X \to A$ and $\lambda: A \times X \to Y$. Elements of A,X and Y are usually named (internal) states, inputs and outputs, respectively. δ is called a "transition function", λ is the "output function". If the sets A,X,Y and hence the function δ and λ too, are finite, then is a finite automaton (also called a Finite State Machine, FSM).

To each Mealy-type automaton , a classical model can be constructed as follows. Let the universe of the classical model be A and consider the elements of XxY as (partial) unary functions on A defined by δ and λ in the following way: $(x,y)(\sigma)=\sigma'$ iff $\delta(\sigma,x)=\sigma'$ and $\lambda(\sigma,x)=y$. What we obtain is clearly a classical model (an algebra), the so called "transition diagram" of $\mathcal{A}$. A Kripke model for $\mathcal{A}$ can be constructed by considering the elements of X as actions: each $x \in X$ can be looked at as a function $x:A \times Y \to A \times Y$ defined by $x(\sigma,y)=(\sigma',z)$ iff $\delta(\sigma,x)=\sigma'$ and $\lambda(\sigma,x)=z$. We set $Q=A \times Y$ and $R=X$. Then clearly (Q,R) is a Kripke model which represents $\mathcal{A}$. Another more trivial

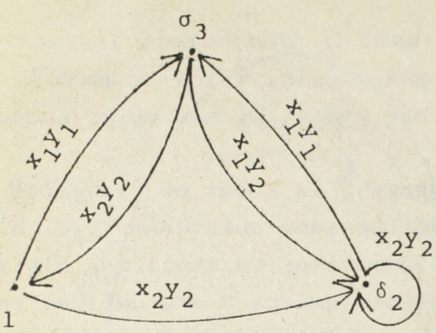


Fig. II-2

$A = \{\sigma_1,\sigma_2,\sigma_3\}$

$X = \{x_1,x_2\}$

$Y = \{y_1,y_2\}$

δ	σ_1	σ_2	σ_3
x_1	σ_3	σ_3	σ_2
x_2	σ_2	σ_2	σ_1

λ	σ_1	σ_2	σ_3
x_1	y_1	y_1	y_2
x_2	y_1	y_2	y_2

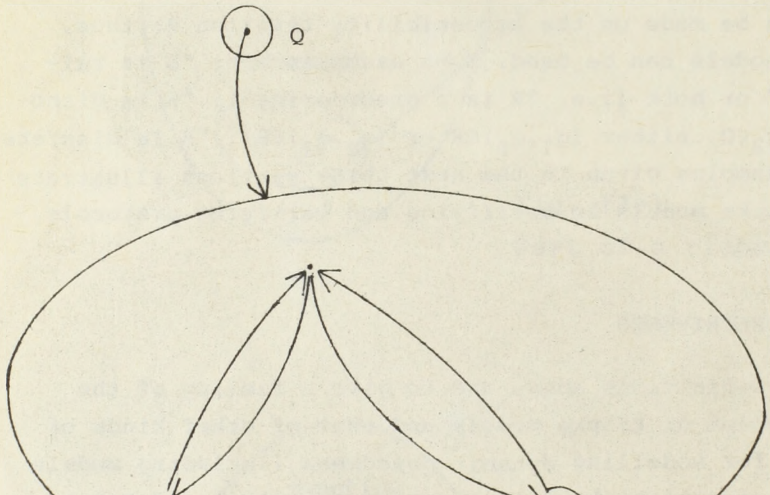


Fig. II-3

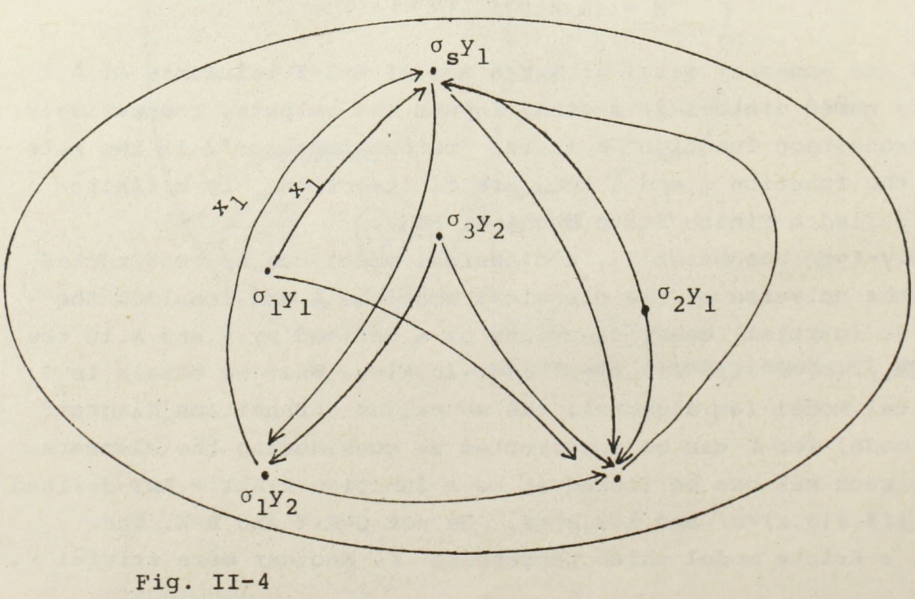


Fig. II-4

Kripke model can be obtained from the classical one constructed above by taking that classical model as the single element of Q and putting $R=\emptyset$. These three models are illustrated by Figs. II-2,3,4 below .
Note that several other automaton models of protocols can be described in the same way. Examples are: variable structure automaton interlocutor link-machine. Each of these concepts is a slight generalization of a Mealy-type automaton and hence it is quite easy to construct both classical and Kripke models which represent them.

Another possibility of modelling a protocol is by Petri-nets. By a (weighted) Petri-net an ordered quadruplet $(\mathbb{P}, \mathbb{T}, \alpha, \beta)$ is meant, where $\mathbb{P}, \mathbb{T}$ are nonempty sets, the set of places and that of transitions, respectively, α is the "forward incidence function":, α: $\mathbb{P} \times \mathbb{T} \to \omega$; β is the "backward incidence function", β: $\mathbb{P} \times \mathbb{T} \to \omega$; where $\omega = \{0,1,\dots\}$ is the set of weights. For all $\tau \in \mathbb{T}$ and $\pi \in \mathbb{P}$, we define the following sets:

$$\begin{aligned} \tau^{\bullet} &= \{\pi \in \mathbb{P} \mid \beta(\pi,\tau) \neq 0\} \\ {}^{\bullet}\tau &= \{\pi \in \mathbb{P} \mid \alpha(\pi,\tau) \neq 0\} \\ \pi^{\bullet} &= \{\tau \in \mathbb{T} \mid \alpha(\pi,\tau) \neq 0\} \\ {}^{\bullet}\pi &= \{\tau \in \mathbb{T} \mid \beta(\pi,\tau) \neq 0\} \end{aligned}$$

A marking m of the Petri-net is a mapping $m: \mathbb{P} \to \omega$. The set of markings will be denoted by M . A transition $\tau \in \mathbb{T}$ is enabled for a marking m iff

$$(\forall \pi \in {}^{\bullet}\tau)(\alpha(\pi,\tau) \leq m(\pi)).$$

Let M_{τ} be the set of markings for which the transition τ is enabled; similarly T_m denotes the set of transitions enabled for m. By the firing of the transition τ we shall mean a function f_{τ}: $M_{\tau} \to M$ defined as follows:

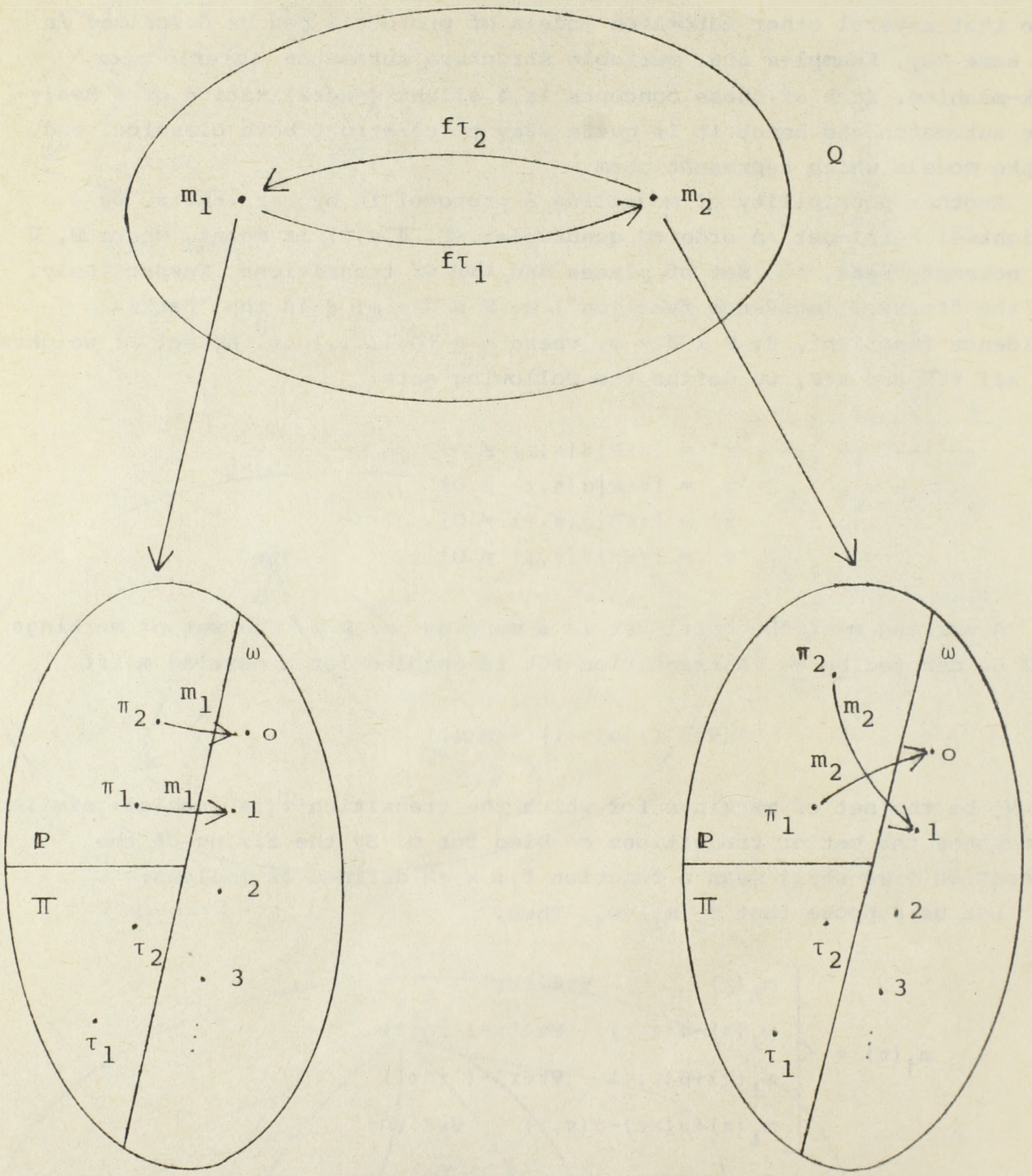
Let us suppose that $f_{\tau}(m_i)=m_j$. Then,

$$m_j(\pi) = \begin{cases} m_i(\pi) & \forall \pi \notin {}^{\bullet}\tau \cup \tau^{\bullet} \\ m_i(\pi)-\alpha(\pi,\tau) & \forall \pi \in {}^{\bullet}\tau-({}^{\bullet}\tau \cap \tau^{\bullet}) \\ m_i(\pi)+\beta(\pi,\tau) & \forall \pi \in \tau^{\bullet}-({}^{\bullet}\tau \cap \tau^{\bullet}) \\ m_i(\pi)+\beta(\pi\tau)-\alpha(\pi,\tau) & \forall \pi \in {}^{\bullet}\tau \cap \tau^{\bullet} \end{cases}$$

By a firing of a Petri-net $(\mathbb{P}, \mathbb{T}, \alpha, \beta)$ we mean a function f: $M \to M$ such that if $f(m_i) = m_j$, then for all $\pi \in \mathbb{P}$

$$m_j(\pi) = \sum_{\tau \in T_{m_i}} f_{\tau}(m_i)(\pi)$$

where f_{τ} is a firing of τ, chosen arbitrarily but well before the summation, for all $\tau \in T_{m_i}$. The set of firings of a Petri-net will be denoted by .



where α and β are defined by the tables:

α	τ_1	τ_2
π_1	1	O
π_2	O	1

β	τ_1	τ_2
π_1	O	1
π_2	1	O

Fig. II-5

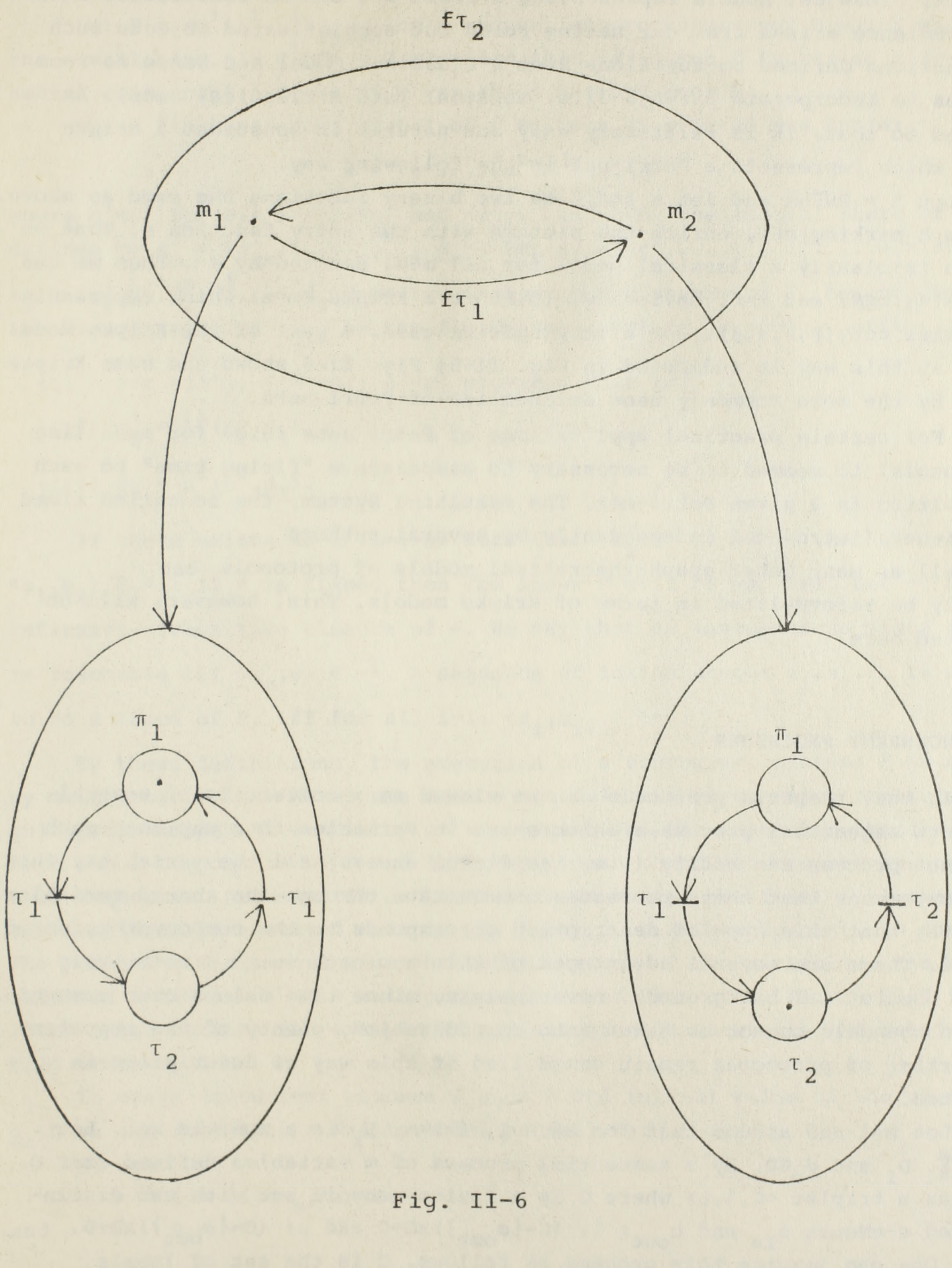


Fig. II-6

The example of Petri-nets demonstrates well the power of Kripke models. In fact, classical models representing a Petri-net can be constructed in an inconvenience arises from our having ruled out sophisticated objects such as functions defined on functions from a classical model and hence no room remains to incorporate firings (i.e. actions) into a classical model. As opposed to that, it is relatively easy and natural to construct a Kripke model which represents a Petri-net in the following way.

Let $A = P \cup \Pi \cup \omega$ and let α and β be two binary functions the same as above. For each marking $m \in M$, enrich the picture with the unary function m. What we obtain is clearly a classical model for all $m \in M$, denoted by A. Then we can set $Q=\{A_m \mid m \in M\}$ and $R=F$. Obviously, (Q,R) is a Kripke model which represents the Petri net $(\mathbb{P},\Pi,\alpha,\beta)$. For a very special case, a part of the Kripke model given in this way is indicated in Fig. II-5; Fig. II-6 shows the same Kripke model by the more commonly used delineation of Petri nets.

For certain practical applications of Petri nets (e.g. for modelling protocols) it seemed to be necessary to associate a "firing time" to each transition in a given Petri-net. The resulting system, the so called timed Petri-net (introduced independantly by several authors as well as many other graph-theoretical models of protocols can easily be reformulated in terms of Kripke models. This, however, will be omitted here.

3. CONCURRENT PROCESSES

In many respects protocols can be viewed as a collection of several distinct sequential processes which share in variables in a way that each distinct process can modify (i.e. has direct access) all the variables. This, in turn, means that these processes communicate through the shared variables. Observe, that this type of description corresponds to the component-oriented method. There are several advantages of this approach (e.g. a relatively clear theoretical background), nevertheless, since time delays over communication channels cannot be taken into consideration, plenty of the important properties of protocols remain unmodelled if this way of description is accepted.

Let m=1 and assume that for each i, $i \leq i \leq m$, D_i is a nonvoid set. Let $D = \bigtimes_{i=1}^{m} D_i$ and $\underline{d}_o \in D$. By a sequential process of m variables defined over D, we mean a triplet $\langle C,\lambda,\mu \rangle$ where C is a finite nonvoid set with two distinguished elements c_{in} and c_{out}; $\lambda: (C-\{c_{out}\}) \times D \to C$ and $\mu: (C-\{c_{out}\}) \times D \to D$.

One can imagine this process as follows. C is the set of labels (control points) of parts of the process, c_{in} is the entry point, c_{out} is the exit point. The name of λ is "the next statement" function; it returns a label, the label of that part which is to be executed next. The function μ is called "data transformation" function. For each i $(1 \leq i \leq m)$, the set D_i is considered as the range of the i-th variable of the process, while $\underline{d}_o$ represents the initial values of the variables.

Let $P = \langle C^i, \lambda^i, \mu^i \rangle_{1 \leq i \leq n}$ be n sequential processes over D which share in the m variables. If $C^i \cap C^j = \emptyset$ provided that $i \neq j$, then P is called a (n-components) concurrent process of m variables defined over D.

By an "instantaneous state" of P we mean an ordered m+n tuple:

$$s = \langle c^1, \ldots, c^n, d^1, \ldots, d^m \rangle$$

where $c^i \in C^i$ for all i $(1=i=n)$ and $\langle d^1, \ldots, d^n \rangle \in D$. The initial state of P is defined by $s_o = \langle c^1_{in}, \ldots, c^n_{in}, \underline{d}_o \rangle$. Let $s_1 = \langle c^1_1, \ldots, c^n_1, d^1_1, \ldots, d^m_1 \rangle$ and $s_2 = \langle c^1_2, \ldots c^n_2, d^1_2, \ldots, d^1_2, \rangle$ be two instantaneous states of P. Let us define the binary relation $\rightarrow_{i,P}$ by the following items: $\langle s_1, s_2 \rangle \in \rightarrow_{i,P}$ iff

(i) for all j, $(1=j=n)$, $c^j_1 = c^j_2$ provided that $i \neq j$

(ii) $c^i_2 = \lambda^i(c^i_i, \underline{d}_1)$

(iii) $\underline{d}_2 = \mu^i(c^i_1, \underline{d}_1)$.

If there exists an i $(1=i=n)$ such that $\langle s_1, s_2 \rangle \in \rightarrow_{i,P}$, then we write $\langle s_1, s_2 \rangle \in \rightarrow_P$. If P is fixed, then the index P will be omitted. Let $\rightarrow^*$ be the reflexive, transitive closure of $\rightarrow$. We say that an instananeous state s of P is reachable iff $\langle s_o, s \rangle \in \rightarrow^*$. A sequence of instantaneous $s_1, s_2 \ldots$ is said to be a trace of P, iff for all $i=1$, $\langle s_i, s_{i+1} \rangle \in \rightarrow$.

By these definitions, the execution of a concurrent process P is modelled by multiprogramming. We assume that a scheduler is given which controls this multiprogramming: at each step of the execution, the scheduler chooses one from the n components of P and lets the chosen component execute one part of that sequential process. In an instantaneous state, the first n elements give the labels to be executed next in the first,..., n-th component, in due course. The remaining m elements give the actual values of the variables. The binary relation $\rightarrow$ simulates this multiprogrammed execution of P. Figure II-7 illustrates these notions (an example of an instanteneous state is indicated by a bold line).

To every concurrent process P over D and initial value of the variables $\underline{d}_o \in D$ we can associate a Kripke model as follows:

We set

$$Q = \{s \mid \langle s_o, s \rangle \in \rightarrow^*\}$$

and

$$R = \rightarrow^*.$$

Observe that R is a preordering in this model. A part of this Kripke model is shown in Fig. II-8.

Note also, that there exists a distinguished element in the Kripke model, namely $0 \in Q$, defined as the intial instantaneous state so.

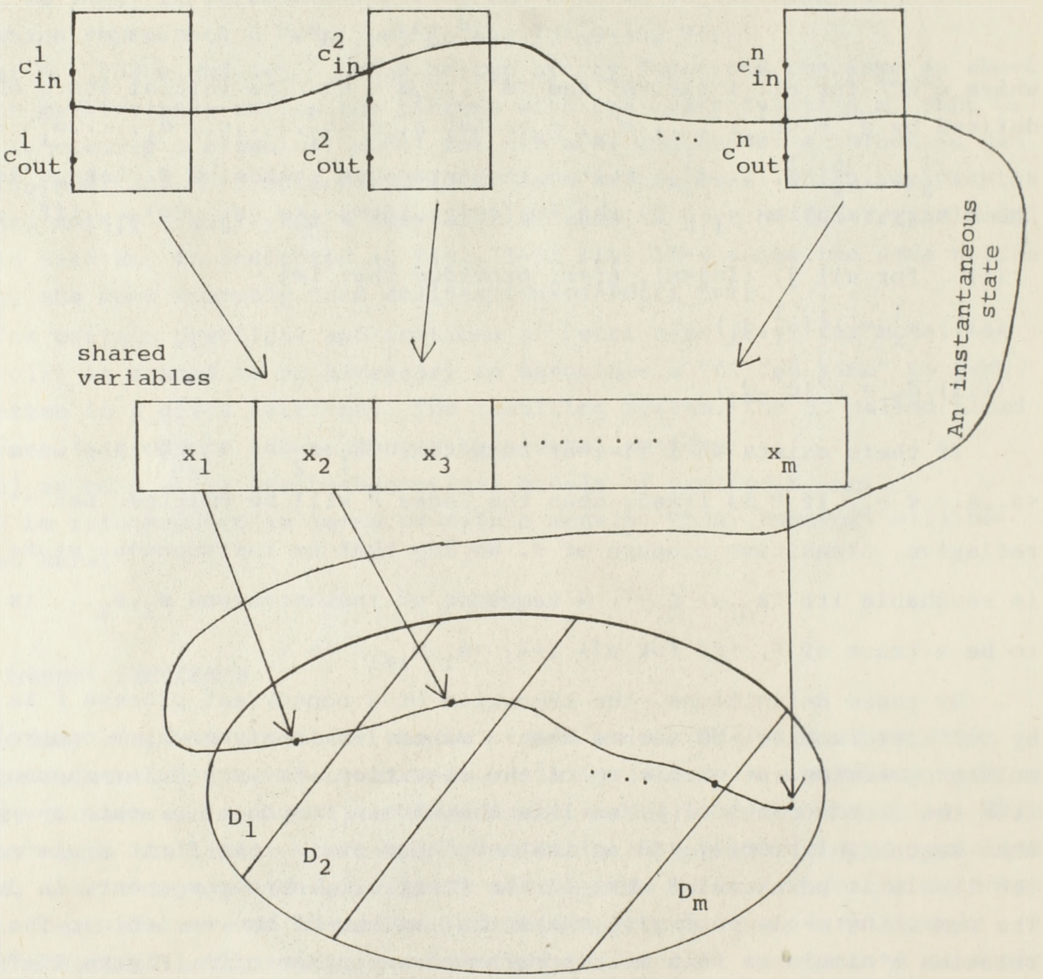

c^1_{in}
c^1_{out}
c^2_{in}
c^2_{out}
c^n_{in}
c^n_{out}
...
An instantaneous state
shared variables
x_1
x_2
x_3
....
x_m
...
D_1
D_2
D_m

Fig. II-7

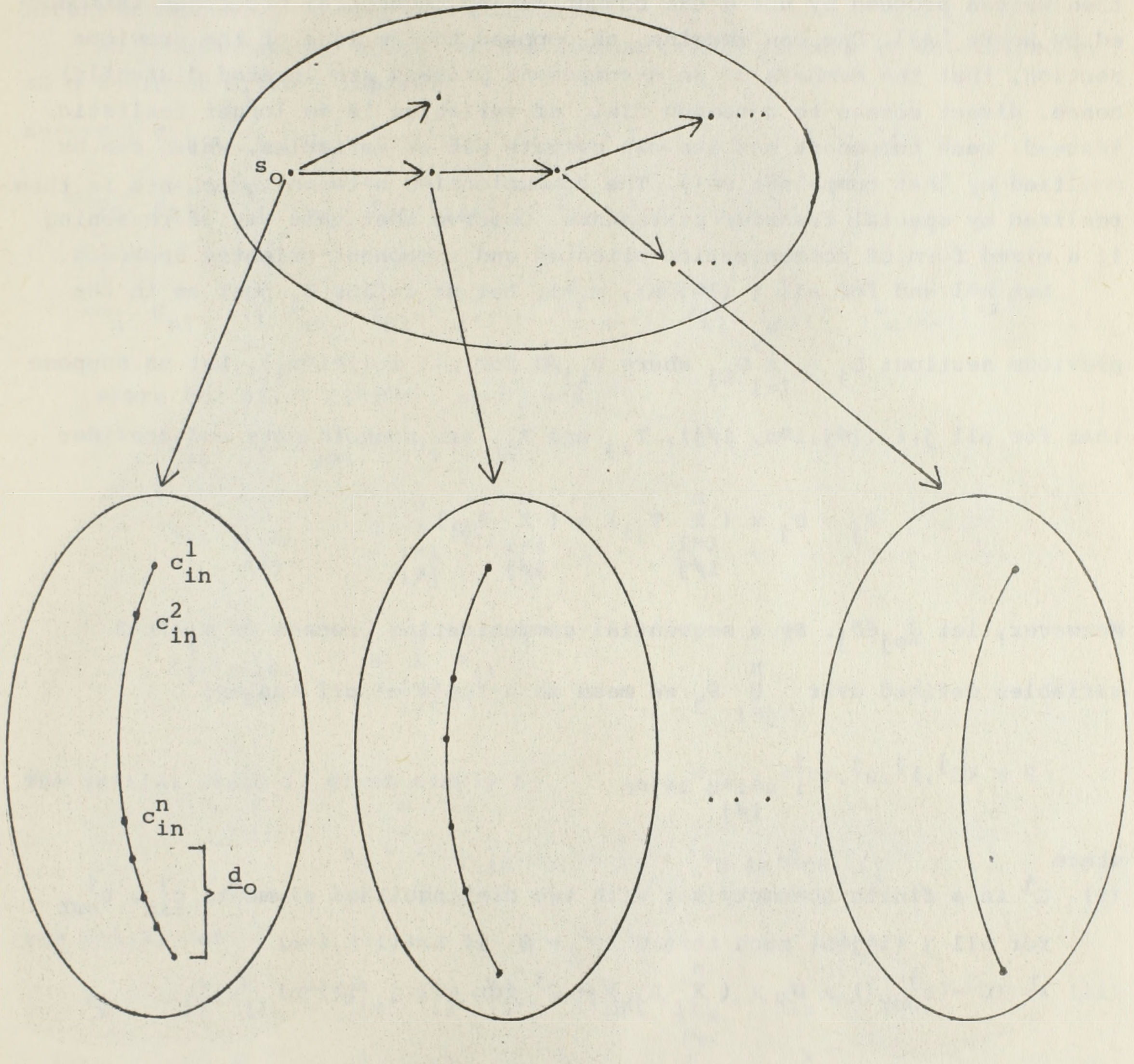


Fig. II-8

4. COMMUNICATING SEQUENTIAL PROCESSES

If we wish to concentrate upon the communication realized by protocols, then we can proceed by using the communicating sequential processes introduced by Hoare [43]. One can imagine, as opposed to the idea of the previous section, that the members of an n-component process are located distantly; hence, direct access to a common field of variables is no longer realistic. Instead, each component has its own private set of variables, which can be modified by that component only. The communication between components is then realized by special transfer statements. Observe that this way of reasoning is a mixed form of communication-oriented and component-oriented approach.

Let $n \geq 1$ and for all j $(1 \leq j \leq n)$, $m_j \geq 1$. Let us define D_j just as in the previous section: $D_j = \mathop{X}\limits_{i=1}^{m_j} D_{ij}$ where $D_{ij} \neq \emptyset$ for all i $(1 \leq i \leq m_j)$. Let us suppose that for all j,i $(1 \leq j, i \leq n,\ i \neq j)$, T_{ij} and R_{ji} are nonvoid sets and consider

$$\mathcal{D}_j = D_j \times \Big(\mathop{X}\limits_{\substack{i=1\\ i\neq j}}^{n} T_{ji} \Big) \times \Big(\mathop{X}\limits_{\substack{i=1\\ i\neq j}}^{n} R_{ji} \Big).$$

Moreover, let $\underline{d}_{oj} \in \mathcal{D}_j$. By a sequential communicating process of m_j+2n-2 variables defined over $\bigcup\limits_{j=1}^{n} \mathcal{D}_j$ we mean an n-tuple of n+2 tuples:

$$P = \langle C^j, \lambda^j, \mu^j, \langle \nu_i^j \rangle_{\substack{s \leq i \leq n\\ i \neq j}} \rangle_{1 \leq j \leq n}$$

where

(i) C^j is a finite nonempty set with two distinguished elements c_{in}^j, c_{out}^j for all j $(1 \leq j \leq n)$ such that $C^k \cap C^\ell = \emptyset$ if $k \neq \ell\ (1=k, \ell=n)$

(ii) λ^j: $(C^j - \{c_{out}^j\}) \times D_j \times \Big(\mathop{X}\limits_{\substack{k=1\\ k\neq j}}^{n} R_{jk} \Big) \to C^j$ for all j $(1 \leq j \leq n)$

(iii) μ^j: $(C^j - \{c_{out}^j\}) \times D_j \times \Big(\mathop{X}\limits_{\substack{k=1\\ k\neq j}}^{n} R_{jk} \Big) \to D_j \times \Big(\mathop{X}\limits_{\substack{k=1\\ k\neq j}}^{n} (T_{jk}) \Big)$ for all j $(1 \leq j \leq n)$

(iv) ν_i^j: $(C^j - \{c_{out}^j\}) \times T_{ji} \to R_{ij}$ for all i and j $(1 \leq j \leq n,\ 1 \leq i \leq n,\ i \neq j)$

The informal content of these clauses can be explained as follows. The role played by set C^j is the same as in the previous section. D_j is the range of private variables of the j-th component; T_{ji} is the range of the distinguished variable x_{ji} (in the j-th component) the value of which is to be transmitted to the value of the variable y_{ij} of the i-th component; and similarly, R_{ji} is the range of the variable y_{ji} (in the j-th component) which receives its value from the i-th component. The mappings λ^j and μ^j determine the computation in the j-th component in the same way as in the concurrent case. Also, $\underline{d}_{oj}$ is the initial value of the private as well as of the communication variables in the j-th component.

An "instantaneous state" of the communicating process is determined by an n-tuple of m_j+2n-1 tuples:

$$\begin{aligned} s= &<<c_1, d_{11},\dots, d_{1m_1}, t_{12},t_{13},\dots, t_{1n}, r_{12}, r_{13},\dots, r_{1n}>,\\ &<c_2, d_{21},\dots, d_{2m_2}, t_{21},t_{23},\dots, t_{2n}, r_{21}, r_{23},\dots, r_{2n}>,\\ &\vdots\\ &<c_n, d_{n1},\dots, d_{nm_n}, t_{n1},t_{n2},\dots, t_{n(n-1)}, r_{n1}, r_{n2},\dots,r_{n(n-1)}>> \end{aligned}$$

where for all j $(1\leq j\leq n)$, $c_j \in C_j$ and

$$<d_{j1},\dots, d_{jmj}> \in D_j$$

$$<t_{ji}>_{\substack{1\leq i\leq n\\ i\neq j}} \in \mathop{X}_{\substack{i\neq 1\\ i\neq j}}^{n} T_{ji}$$

$$<r_{ji}>_{\substack{1\leq i\leq n\\ i\neq j}} \in \mathop{X}_{\substack{i=1\\ i\neq j}}^{n} R_{ji} \quad .$$

The initial state is given simply by

$$s_o = <<c_{1\ in},\underline{d}_{o1}>, \dots, <c_{n\ in},\underline{d}_{on}>>.$$

For k=1,2, let

$$\begin{aligned} s_k = &<<c^k_1,d^k_{11},\dots,d^k_{1m}, t^k_{12},t^k_{13},\dots,t^k_{1n}, r^k_{12},\dots,r^k_{1n}>,\dots,\\ &<c^k_n,d^k_{n1},\dots,d^k_{nm_n}, t^k_{n1},\dots t^k_{n(n-1)}, r^k_{n1},\dots,r^k_{n(n-1)}>> \end{aligned}$$

be two instantaneous states of P. We can define the binary relation $\rightarrow_P$ which simulates the execution of P as follows:

$$<s_1,s_2> \in \rightarrow_P \text{ iff for all j } (1\leq j\leq n)$$

(i) $c^2_j = \lambda^j(c^i_j, d^1_{j1},\dots, d^1_{jm_j}, <r_{ji}>_{\substack{1\leq i\leq n\\ 1\neq j}})$

(ii) $<d^2_{j1},\dots,d^2_{jm_j},<t^2_{ji}>_{\substack{1\leq i\leq n\\ i\neq j}}> = \mu^j(c^1_j,d^1_{j1},\dots,d^1_{jm_j}, <r^1_{ji}>_{\substack{1\leq i\leq n\\ i\neq j}}$

(iii) $r_{ji} = \nu_i^j(c_i^1, t_{ij})$ for all i ($1 \leq i \leq n$, $i \neq j$).

Again, $\rightarrow^*$ is the reflexive, transitive closure of $\rightarrow$ (where the index *P* is omitted provided no confusion can occur). We say that an instantaneous state s is reachable iff $\langle s_o, s \rangle \in \rightarrow^*$. The definition of a trace is similar to that of the concurrent processes.

The execution of a communicating sequential process *P* can be described briefly in the following way. Each component of *P* has its own scheduler, which controls the execution: at every step, the scheduler determines that part of the component which is to be executed next, according to the next statement function λ, and performs the data transformations prescribed by μ. Observe, that both the choice of the next part and that of the data transformation depend strongly on the value of the communicating variables received from other components in the previous steps. Finally, the scheduler transmits to other components the values of the communicating variables obtained in the present step. This way of execution means that different components run relatively independently; the only synchronization among them is realized by the communicating variables.

A Kripke model of the communicating sequential process *P* is easily obtained by setting

$$Q = \{s \mid \langle s_o, s \rangle \in \rightarrow^* \text{ and}$$

$$R = \rightarrow^* \ .$$

By definition, R is again a preordering.

CHAPTER III. LOGICAL LANGUAGES

The aim of this part is to develop mathematical logical languages appropriate for studying Kripke models. We shall start by recalling the most fundamental definitions of classical first order languages. Since familiarity with these definitions is assumed, we shall proceed at a fairly rapid pace. The main concepts of these languages will be used in developing nonclassical languages as auxiliary means.

1. CLASSICAL FIRST ORDER LANGUAGES

By a type d we shall mean a quintuple $(\Omega_R, \Omega_F, \Omega_K, t_R, t_F)$ where $\Omega_R, \Omega_F, \Omega_K$ are pairwise disjoint sets, the sets of relation, function and constant symbols, respectively, and $t_R\colon \Omega_R \to \{1,2,\dots\}$, $t_F\colon \Omega_F \to \{1,2,\dots\}$ are the "arity functions" of Ω_R and Ω_F. Let a type d be fixed. We can give the set of nonlogical symbols for the language to be constructed as $\Omega_R \cup \Omega_F \cup \Omega_K$. The logical symbols are $\wedge, \neg$, $\forall$ and $\equiv$ (for conjunction, negation, universal quantification and equality) and we shall also use some separator symbols:),(,and an infinite set V of variables.

By the set of terms of type d we shall mean the least set $Term_d$ satisfying the following conditions:

(i) $\Omega_K \subset Term_d$ and $V \subset Term_d$

(ii) if $f \in \Omega_F$ such that $t_F(f)=n$ and $\tau_1,\dots,\tau_n \in Term_d$, then $f(\tau_1,\dots,\tau_n) \in Term_d$.

Let

$$Prim_d = \{r(\tau_1,\dots,\tau_n) \mid r \in \Omega_R,\ t_R(r)=n,\ \tau_1,\dots,\tau_n \in Term_d\} \cup \{\tau_1 \equiv \tau_2 \mid \tau_1,\tau_2 \in Term_d\},$$

be the set of prime formulae.

The set of classical first order formulae F_d is the smallest set determined by the following recursion:

(i) $Prim_d \subset F_d$

(ii) if $\varphi,\psi \in F_d$, then $\neg\varphi \in F_d$ and $\varphi\wedge\psi \in F_d$

(iii) if $\varphi \in F_d$, $v \in V$, then $(\forall v)\varphi \in F_d$.

By a classical model of type d, we mean a pair (A,I) such that $A \neq \emptyset$; for all $f \in \Omega_F$, if $t_F(f)=n$, then

$$I(f): A^n \to A;$$

for all $r \in \Omega_R$, if $t_R(r)=n$, then

$$I(r) \subset A^n$$

and finally, for all $k \in \Omega_K$,

$$I(k) \in A.$$

The collection of all classical models of type d will be denoted by M_d^{cl}. Elements of M_c^{cl} will be denoted by capital (script) German (Gothic) letters: $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$, ... and so on.

Let $\mathfrak{A} \in M_d^{cl}$. By an assignment relative to $\mathfrak{A}=(A,I)$ we shall mean a mapping

$$a: V \to A.$$

The set of all assignments relative to $\mathfrak{A}$ will be denoted by $A^{\mathfrak{A}}$.

Let $\mathfrak{A} \in M_d^{cl}$ and $a \in A^{\mathfrak{A}}$. For every $\tau \in \mathrm{Term}_d$ we shall define $\tau^{\mathfrak{A}}[a]$, the "value of the term τ in $\mathfrak{A}$ under assignment a" by the following recurrence:

(i) if $\tau = v \in V$, then $\tau^{\mathfrak{A}}[a] = a(v)$

(ii) if $\tau = k \in \Omega_K$, then $\tau^{\mathfrak{A}}[a] = I(k)$

(iii) if $\tau = f(\tau_1, \ldots, \tau_n)$, then

$$\tau^{\mathfrak{A}}[a] = I(f)(\tau_1^{\mathfrak{A}}[a], \ldots, \tau_n^{\mathfrak{A}}[a]).$$

For every $\varphi \in F_d$, the relation $\mathfrak{A} \models \varphi[a]$, to be read as "$\varphi$ is true in under the assignment a" is defined by recursion, too.

(i) if $\varphi^{\cdot} = r(\tau_1, \ldots, \tau_n) \in \mathrm{Prim}_d$, then

$\mathfrak{A} \models \varphi[a]$ iff $<\tau_1^{\mathfrak{A}}[a], \ldots, \tau_n^{\mathfrak{A}}[a]> \in I(r)$

(ii) if $\varphi = (\tau_1 \equiv \tau_2) \in \mathrm{Prim}_d$, then

$\mathfrak{A} \models \varphi[a]$ iff $\tau_1^{\mathfrak{A}}[a] = \tau_2^{\mathfrak{A}}[a]$

(iii) if $\mathfrak{A} = \neg \psi$, then

$\mathfrak{A} \models \varphi[a]$ iff $\mathfrak{A} \not\models \psi[a]$

(iv) if $\varphi = \psi \wedge \chi$, then

$\mathfrak{A} \models \varphi[a]$ iff both $\mathfrak{A} \models \psi[a]$ and $\mathfrak{A} \models \chi[a]$ hold

(v) if $\varphi = (\forall v)\psi$, then

$\mathfrak{A} \models \varphi[a]$ iff for all $a' \in A^{\mathfrak{A}}$ such that $a(u)=a'(u)$ provided $u \neq v$, $\mathfrak{A} \models \psi[a']$ holds.

If for all $a \in A^{\mathfrak{A}}$, $\mathfrak{A} \models \varphi[a]$, then we say that φ is true in $\mathfrak{A}$ and write simply $\mathfrak{A} \models \varphi$. If for all $\mathfrak{A}$, $\mathfrak{A} \models \varphi$, then φ is said to be universally true, in notation: $\models \varphi$.

We shall use some derived logical symbols, too. These can be defined as follows:

$$\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$$
$$\varphi \rightarrow \psi = \neg\varphi \vee \psi$$
$$\varphi \leftrightarrow \psi = \varphi \rightarrow \psi \wedge \psi \rightarrow \varphi$$
$$\exists v \varphi = \neg\forall v \neg\varphi$$

It is easy to check that these definitions coincide with the intuitive meaning of disjunction, implication, (logical) equivalence and existential quantification.

An occurrence of a variable v in a formula φ is bounded iff it is either immediately after a quantifier or in that subformula of φ which follows a quantifier. If an occurrence of v is not bound then it is free. Note that the same variable v can occur in the same formula φ in both ways; for example, in the formula $\neg(v \equiv f(k)) \wedge (\forall v)(v \equiv k)$, the first occurrence of v is free whereas, the second and third ones are bounded. If no variable occurs freely in φ, then φ is called a sentence. Similarly, if no quantifier appears in φ, then φ is named quantifier-free.

It is clear from the abovesaid, that $\langle F_d, M_d^{cl}, \models\rangle$ is a mathematical logical language for every type d: the classical first order language of type d.

2. REFORMULATION OF CONCURRENT AND COMMUNICATING PROCESSES USING PROGRAMMING -LANGUAGE-LIKE CONSTRUCTS

The aim of this section is twofold: first, to give an application of the first order classical language and second, to redefine concurrent and communicating sequential processes in terms of programming-language-like constructs. These constructs are well-known and commonly used, hence the reformulation - hopefully - helps to make the use of processes more convenient and explcit.

Let $d = \langle \Omega_R, \Omega_F, \Omega_K, t_R, t_F \rangle$ be a fixed type; $Term_d$ is the set of terms of type d and let QFF denote the set of quantifier-free first order formulae. We shall define the set SCP_d, the set of sequential composite processes of type d, by the following recurrence.

(i) The set SC_d of simple commands (of type d) is the least set containing the statements:

(α) skip

(β) for all $v \in V$ and $\tau \in Term_d$

$v \leftarrow \tau$.

The informal meanings of these commands are well-known: skip is the empty--statement, $v \leftarrow \tau$ assigns the value of τ to the variable v.

(ii) The set CC_d of communicative commands (of type d) is the smallest set which contains the following statements. Let Ξ be an arbitrary (maybe empty) set of templates. Elements of Ξ are intended to identify the sort of messages transmitted by the communicative statements. (For example, in the case of HDLC protocols, templates can be: sv for supervisory frames, i for information frames and u for unnumbered frames, i.e. in this particular case $\Xi = \{sv, i, u\}$.) Hence, for all $\tau \in Term_d$, $\xi \in \Xi$, $v \in V$ and $p \in SCP_d$

(α) $p(v)!\xi(\tau) \in CC_d$

(β) $p(\tau)?\xi(v) \in CC_d$

The statement $p(v)!\xi(\tau)$ is to be understood informally as: "transfer the value of τ" (which is of the sortiment given by the template ξ) to the variable v of the process p. Similarly, $p(\tau)?\xi(v)$ is to be meant as "wait for the message τ of template ξ from the process p and receive it in v".

(iii) The set CP_d of composite processes (of type d) is the smallest set which contains $SC_d \cup CC_d$ and all of the following statements given below:

(α) The set GP_d of guarded processes (of type d) is defined in two steps. Let $n \geq 1$, and for all i $(1 \leq i \leq n)$, let $\varphi_i \in QFF_d$ $q_i \in CP_d$ and $p_i \in \{skip\} \cup CC_d$. Then (α1) the set of selective guarded processes SGP_d consists of all statements of the form

$$\cup(\varphi_1, p_1 \to q_1 \square \ldots \square \varphi_n, p_n \to q_n)$$

(α2) the set RGP_d of repetitive guarded processes consists of all statements of the form

$$*(\varphi_1, p_1 \to q_1 \square \ldots \square \varphi_n, p_n \to q_n).$$

The intended meaning of a statement in the form

$$\cup(\varphi_1, p_1 \to q_1 \square \ldots \square \varphi_n, p_n \to q_n)$$

is: "choose nondeterministically an i $(1 \leq i \leq n)$ such that φ_i is satisfied for the actual values of the variables, then execute p_i followed by the execution of q_i". The statement

$$*(\varphi_1, p_1 \to q_1 \square \ldots \square \varphi_n, p_n \to q_n)$$

is to be understood informally as "repeat nondeterministically in all possible ways the execution of p_i followed by q_i for all such i $(1 \leq i \leq n)$ for which φ_i is satisfied by the actual values of the variables until every φ_i will be false". (These explanations of the meanings will be made more precise below.)

(β) The set of sequenced processes (of type d) SP_d consists of all statements of the following form: let C be a finite set (the set of labels) with two distinguished elements $c_{in} \in C$ and $c_{out} \in C$; let $p_1, \ldots, p_n \in CP_d$, then $(c_{in}: p_1, c_1: p_2, \ldots; c_{out}: p_n) \in SP_d$.

The execution of a sequenced process is to be understood informally in the well-known way: let p_1 labelled by c_{in} be executed first followed by p_2 labelled by c_1, and so on; finally let p_n labelled by c_{out} be the last in the sequence of execution.

The set SCP_d is then defined as the smallest set containing SC_d, CC_d, CP_d, SP_c and $GP_d = SGP_d \cup RGP_d$ and closed under the formation rules exhibited above.

Before going further in the definitions of concurrent and communicative sequential processes, we define the set of variables $V(p)$ and the set of communicating variables $CV(p)$ occurring in a process p, by recurrence:

Let $p \in SCP_d$;

if $p = skip$, then $V(p) = CV(p) = \emptyset$

if $p = v \leftarrow \tau$, then $V(p) = \{v\} \cup V(\tau)$

where $V(\tau)$ is the set of variables occurring freely in τ, and $CV(p) = \emptyset$

if $p = q(v)!\xi(\tau)$, then $V(p) = V(\tau)$ and $CV(q) = \{v\}$

if $p = q(\tau)?\xi(v)$, then $V(q) = V(\tau)$ and $CV(p) = \{v\}$

if $p = (c_1:p_1;\ c_2:p_2)$, then $V(p) = V(p_1) \cup V(p_2)$ and $CV(p) = CV(p_1) \cup CV(p_2)$

if $p = .(\varphi_1, p_1 \rightarrow q_1 \,\Box\, \ldots \,\Box\, \varphi_n, p_n \rightarrow q_n)$ where the dot stands for either

$\cup$ or $*$, then $V(p) = \bigcup_{i=1}^{n} (V(\varphi_i) \cup V(p_i) \cup V(q_i))$ and $CV(p) = \bigcup_{i=1}^{n} C(p_i)$ where

$V(\varphi_i)$ is the set of variables occurring freely in φ_i.

Let $m \geq 1$ and for all i $(1 \leq i \leq m)$ $p_i \in SCP_d$. The m-tuple $[p_1,\ldots,p_m]$ is a concurrent sequential process iff for all i,j $(1 \leq i,j \leq m)$, $V(p_i) = V(p_j)$ and $CV(p_i) = CV(p_j) = \emptyset$. Similarly, $[p_1,\ldots,p_m]$ is a communicating sequential process iff for all i,j $(1 \leq i,j \leq m,\ i \neq j)$, $V(p_i) \cap V(p_j) = \emptyset$ and $CV(p_i) \subset \bigcup_{\substack{k=1 \\ k \neq i}}^{m} CV(p_k)$.

To define the semantics of the notions introduced above, let $\mathfrak{A} \in M_d^{cl}$ and the set of assignments relative to $\mathfrak{A}$ be $A^{\mathfrak{A}}$. Since in either case of the process introduced above $V(p) \cup CV(p) \in V$, we can observe that an "instantaneous state" is just an element of $A^{\mathfrak{A}}$ (obviously restricted to the set $V(p) \cup CV(p)$. Similarly, the relation $\rightarrow$ is defined as a binary relation on $A^{\mathfrak{A}}$. This, in turn, means that the Kripke model associated with a concurrent or communicating process is given on the set of assignments relative to the classical model $\mathfrak{A}$, the universe of which contains the ranges of all variables occurring in that process. Hence $\mathfrak{A}$ plays the role of an abstract data type [32][47].

Formally, we have to define the "next statement function", the "data transformation function" and, in case of communicating processes the functions ν_i^j that determine the communication.

The next statement function λ can easily be given - depending on the properties of the "executing agent", since according to the definitions above labelling is used only in sequenced processes, and sequencing is allowed only a finite number of times. In the simplest case, let

$$(c_{in}:p_1;\ c_2:p_2;\ldots;\ c_{out}:p_n) \in SP_d,$$

and assume that $a \in A^{\mathfrak{A}}$ defines the actual values of the variables. Then,

$$\lambda(c_i, a) = c_{i+1} \quad \text{for all } i \quad (1 \leq i \leq n)$$

where c_{in} and c_{out} are denoted by c_1 and c_n, respectively.

The data transformation function μ is defined by recursion. Let us suppose that $a \in A^{\mathfrak{A}}$ gives the actual value of the variables immediately before the execution of process p. Let c be an arbitrary label. If p=skip, then $\mu(c,a)=a$. If $p=v \leftarrow \tau$, then $\mu(c,a)(w)=\tau^{\mathfrak{A}}[a]$ for v=w, otherwise $\mu(c,a)(w)=a(w)$. Let $p=\cup(\varphi_1, p_1 \rightarrow q_1 \; \square \; \ldots \; \square \; \varphi_n, p_n \rightarrow q_n)$. If for all i $(1 \leq i \leq n)$, $\mathfrak{A} \not\models \varphi_i[a]$, then $\mu(c,a)=a$, otherwise let i be chosen nondeterministically in such a way that $\mathfrak{A} \models \varphi_i[a]$ and let us define that $\mu(c,a)=\mu_i(c,a)$ where $\mu_i(c,a)$ represents the data transformation of the process $(p_i;q_i)$.

Let $p=*(\varphi_1, p_1 \rightarrow q_1 \; \square \; \ldots \; \square \; \varphi_n, p_n \rightarrow q_n)$. If for all i $(1 \leq i \leq n)$, $\mathfrak{A} \not\models \varphi[a]$, then $\mu(c,a)=a$, otherwise let i be repeatedly and nondeterministically chosen so that $\mathfrak{A} \models \varphi_i[a]$ and let $\mu(c,a)=\mu_i(c,a)$ be defined where $\mu_i(c,a)$ again represents the transformation of the process $(p_i;q_i)$. Finally, let us suppose that $p=q(v)!\xi(\tau)$ and assume that a and a' give the actual values of the variables in p and q, respectively. Note that p and q have no common variables, hence $a \cup a'$ is a restriction of an element of $A^{\mathfrak{A}}$. Then, let

$$\nu_q^p(c, \tau^{\mathfrak{A}}[a]) = a'(v)$$

and

$$\mu(c,a')(w) = a'(w) \quad \text{for } w \neq v \; .$$

Using similar notations, if $p=q(\tau)?\xi(v)$, then $\nu_p^q \; (c, \tau^{\mathfrak{A}}[a']) = a(v)$ and $\mu(c,a)(w) = a(w)$ for $w \neq v$.

Observe that at each moment of execution, the actual values of the variables can be expressed by classical first order formulae, hence we can formulate the before - after behaviour of processes. However, it is impossible to describe any execution of a process as a whole because classical languages provide no means for handling more than one assignment simultaneously. What is missing is a kind of quantification over assignments. This will be provided by more complex languages in the rest of the chapter.

3. MODAL (FIRST ORDER) LANGUAGES

Historically modal languages were introduced for formalizing such modal sentences as "It is possible that ..." or "It is necessary that ...". Later, Kripke defined the exact semantics for these languages [45-46] using the concept of nonclassical models now named Kripke models. Nowadays, modal languages are the simplest formal tools which can describe Kripke models. Given a Kripke model (Q,R), the basic idea of constructing modal languages is that elements of Q do not appear explicitly in formulae nor does the accessibility relation R.

Let a type d be fixed. The alphabet of modal languages can be obtained from that of the classical ones by considering two new logical symbols: $\Box$ (box) and $\Diamond$ (diamond).

The definition of formulae can be carried over in the same way as that of F_d but two new formation rules can be used:

(i) if φ is a modal formula, then $\Box\varphi$ is a modal formula, too

(ii) if φ is a modal formula, then $\Diamond\varphi$ is a modal formula, too.

The set of modal formulae is the smallest set which contains F_d and is closed under rules (i) and (ii). The set of modal formulae will be denoted by F_d^{mod}.

The collection of Kripke models of type d is denoted by M_d^{Kr};

$$M_d^{Kr} = \{(Q,R) \mid Q \subset M_d^{cl} \text{ such that if } \mathfrak{A} = (A,I_A),\ \mathfrak{B} = (B,I_B) \in Q \text{ then } A = B \text{ and } R \subset Q\times Q\}$$

The assumption that all elements of Q have the same set as a universe allows the definition of assignment to remain unchanged; let the set of assignments relative to Q be denoted by A^Q. The definition of $\mathfrak{A} \models^{mod} \varphi[a]$ for $\mathfrak{A} \in Q$, $a \in A^Q$, $\varphi \in F_d^{mod}$ goes similarly to the classical case except that two new clauses must be added to the recurrènce:

(i) $\mathfrak{A} \models^{mod} \Box\varphi[a]$ iff for all $\mathfrak{B} \in Q$, such that $\langle\mathfrak{A},\mathfrak{B}\rangle \in R$, $\mathfrak{B} \models^{mod} \varphi[a]$ holds;

(ii) $\mathfrak{A} \models^{mod} \Diamond\varphi[a]$ iff for some $\mathfrak{B} \in Q$, such that $\langle\mathfrak{A},\mathfrak{B}\rangle \in R$, $\mathfrak{B} \models^{mod} \varphi[a]$ holds.

If for all $\mathfrak{A} \in Q$, $\mathfrak{A} \models^{mod} \varphi[a]$, then φ is said to be true in the Kripke model (Q,R) under assignment a, in notation: $(Q,R) \models^{mod} \varphi[a]$. If for all $a \in A^Q$, $(Q,R) \models^{mod} \varphi[a]$, then is true in (Q,R). If φ is true in every element of M_d^{Kr}, then it is a universally true (modal) formula.

Obviously, we have defined a mathematical logical language

$$\langle F_d^{mod},\ M_d^{Kr},\ \models^{mod} \rangle$$

this being the first order model language for every type d.

Different properties of the Kripke model (Q,R) can easily be expressed by modal formulae. Some of them are exhibited in the following table; it is to be understood that the property on the left hand side of the vertical holds for (Q,R) iff the formula on the right hand side is true in (Q,R) for all $\varphi, \psi \in F_d^{mod}$.

R is reflexive	$\Box\varphi \rightarrow \varphi$
R is transitive	$\Diamond\Diamond\varphi \rightarrow \Diamond\varphi$
R is dichotomous	$\Box(\Box\varphi \rightarrow \Box\psi) \vee \Box(\Box\psi \rightarrow \Box\varphi)$
R is discrete	$\Box(\Box(\varphi \rightarrow \Box\varphi) \rightarrow \varphi) \rightarrow (\Diamond\Box\varphi \rightarrow \varphi)$

Several properties, useful from the protocol verificational point of view, can be formalized in the modal languages. Some of the most important ones are collected in the table below.

Property	Modal formula
"φ will be true and remains true thereafter"	$\Diamond\Box\varphi$
"φ will be true again and again, infinitely often"	$\Box\Diamond\varphi$
"φ is an inductive assertion, i.e. if it is ever true, then it will remain true thereafter"	$\Box(\varphi\to\Box\varphi)$
"if ψ is ever true, then φ will always be true thereafter	$\psi\to\Box\varphi$
"if ψ is ever true, then φ will sometimes be true thereafter	$\psi\to\Diamond\varphi$

4. TEMPORAL (FIRST ORDER) LANGUAGES

The functioning of protocols is strongly connected with time considerations. Albeit modal languages are appropriate to describe many properties of protocols, others, more explicitly referring to the flow of time, are inexpressible by modal formulae only. An example can be:

"φ will have been true, when ψ becomes true".

To deal with such sentences, one can introduce other modal operators in addition to or instead of the box and diamond symbols. One possibility is to use the "until" operator. This operator is somewhat stronger than $\Box$ and $\Diamond$. Historically, these languages were introduced in order to formalize the tenses of English verbs, hence they are named tense - or temporal languages [24-27][30-31][44][50-51][60]. The basic idea of constructing such languages is similar to that of the modal ones, i.e. neither the elements of Q nor R will appear explicitly in temporal formulae.

Let a type d be fixed. The alphabet of temporal languages are obtained from symbols of classical languages by adjoining a new logical symbol $\rhd$ to them. The recursive definition of classical formulae is augmented by one new formation rule: if φ and ψ are (temporal) formulae, then $\varphi\rhd\psi$ (reads as "φ until ψ") is again a temporal formula.

The set of temporal formulae of type d will be denoted by F_d^{tem}. Let $(Q,R)\in M_d^{Kr}$, $a\in A^Q$ and $\mathfrak{a}\in Q$. The relation $\models^{tem}$ is defined by the usual recursion as in the classical case by adding the following clause, as well:

$\mathfrak{a}\models^{tem}\varphi\rhd\psi[a]$ iff for all $\mathfrak{b}\in Q$ such that $<\mathfrak{a},\mathfrak{b}>\in R$, we have that if for all $\mathfrak{c}\in Q$, for which both $<\mathfrak{A},\mathfrak{L}>\in R$ and $<\mathfrak{L},\mathfrak{B}>\in R$ hold, then $\mathfrak{L}\models^{tem} \neg\varphi[a]$ entails $\mathfrak{B}\models^{tem}\varphi[a]$.

Clearly, $<F_d^{tem}, M_d^{Kr}, \models^{tem}>$ is a mathematical logical language, a temporal first order language.

The operator $\rhd$ is really stronger than $\Box$ and $\Diamond$ since we can define:

$$\Box\varphi \text{ iff } \psi\rhd(\psi\wedge\neg\psi)$$

$$\text{and} \quad \Box\varphi \text{ iff } \neg\Box\neg\varphi \text{ for arbitrary } \psi.$$

Hence, F_d^{mod} is definable in F_d^{tem}, which means, in turn, that everything expressible in the modal languages is expressible by temporal formulae.

We can restrict our considerations to a special class of Kripke models, namely to those where R is a discrete dichotomous ordering. This restriction, however, causes some minor changes to the description of processes by ruling out the basic nondeterminancy built into guarded commands. Nevertheless, when nondeterminancy is only of secondary interest, then the restriction can be very useful.

Let $M_d^{ord} \subset M_d^{Kr}$ such that R is a discrete dichotomous ordering. In this particular case, the meaning of $\varphi\rhd\psi$ can be given in a somewhat simpler way than in the unrestricted one, as follows.

Let $\mathfrak{A}\in M_d^{ord}$ and $a\in A^Q$ $\mathfrak{A}\models\varphi\rhd\psi[a]$ iff there exist $\mathfrak{B}_1,\mathfrak{B}_2,\ldots,\mathfrak{B}_n\in Q$ such that for all i $(1\leq i\leq n)$, $<\mathfrak{B}_i,\mathfrak{B}_{i+1}>\in R$ and $\mathfrak{B}_i \models\varphi$, moreover $\mathfrak{B}_n \models\psi$.

Considering only elements of M_d^{ord}, it is possible to introduce another modal operator called the "next" operator. Let $O\psi$ stand for $(\varphi \neg\varphi)\rhd\psi$ where φ is arbitrary. Intuitively speaking, $O\psi$ states that ψ will be true in the next state.

In closing, we give some properties of protocols, expressible by temporal formulae in the following table.

Property	Temporal formula
"if φ is true, then it remains true until Ψ becomes true"	$\varphi\rightarrow(\varphi\rhd\Psi)$
" by the time φ is true, ψ will have been true"	$\Diamond\varphi\rightarrow(\neg\varphi\rhd\psi)$

5. SITUATION-DESCRIPTIVE (FIRST-ORDER) LANGUAGE

In the construction of modal and temporal languages we stipulated that no elements of Q and R can appear explicitly in formulae and we overcome the difficulty of identifying states by allowing quantification over Q. (Observe that according to their definitions, $\Box$ and $\Diamond$ play the role of universal and existential quantification on states, respectively.) This indirect and incomplete specification of states is sometimes unsatisfactory, i.e. when describing protocols it is often necessary to identify a particular state explicitly. Hence we have to allow some kind of appearance of elements of A_Q in constructing formulae. If this is the case, then the languages obtained are called situation-descriptive languages [21][58-59][66]. We shall not deal with such languages in general, but give only one very special case.

Let L be a finite set of labels. Elements of L will be used as extra-logical variables. Let F_d^{tem} be the set of temporal formulae constructed earlier in this chapter, see Section 4. The set F_d^{sit}, the set of situation-descriptive formulae, is defined by

$$F_d^{sit} = F_d^{tem} \cup \{\text{at } l \mid l \in L\} \cup \{\text{after } l \mid l \in L\}.$$

The models of the language are again Kripke models. By an assignment relative to the Kripke model (Q,R) we mean a mapping $a: V \cup L \to Q \cup A$ where V is the set of (ordinary) variables and A is the (common) universe of elements of Q, such that

$$a: V \to A \quad \text{and} \quad a: L \to Q.$$

The set of assignments relative to (Q,R) is denoted by $A^{\mathfrak{A}}$. The relation $\models^{sit}$ can be defined by recurrence for elements of F_d^{tem} just as $\models^{tem}$ was. For the new formulae, the definitions are given as follows.

Let $\mathfrak{A} \in Q$, $a \in A^Q$; then

$\mathfrak{A} \models^{sit}$ at l iff $a(l) = \mathfrak{A}$ and

$\mathfrak{A} \models^{sit}$ after l iff $\langle a(l), \mathfrak{A}\rangle \in R$.

Then, obviously, $\langle F_d^{sit}, M_d^{Kr}, \models^{sit}\rangle$ is a mathematical logical language for all type d, the situation-descriptive language.

To illustrate the expressive power of situation-descriptive languages let us consider a two-component concurrent sequential process P. Let us suppose that in each of the two components. there exists a critical section labelled by l_1 in the first and by l_2 in the second component. If these critical sections are not allowed to be executed simultaneously, i.e. the two components are mutually exclusive with respect to the labels l_1 and l_2, then this situation is described by the formula

$$\Box \neg(\text{at } l_1 \wedge \text{at } l_2)$$

Observe that for communicating sequential processes the mutual exclusivity is meaningless.

We say that a concurrent or communicating process P is deadlocked or, equivalently, is in a deadlock state, if no component can step any further; e.g. in the concurrent case, every component of the process is waiting for the fulfilment of such conditions which are falsified by the actual value of the variables whereas in the communicating case, every component is waiting for the arrival of messages from other components. Let us consider first, a concurrent process with two components and assume that the first statement l_1: if φ_1 then l_2 else l_1 occurs in the first component, the statement l_3: if φ_2 then l_3 else l_3 in the second one. (Note that the traditional programming construct if...then...else... is easily expressible by selective guarded commands.) It may well happen under execution of P, that the next parts of the two components to be executed are the statements labelled by l_1 and l_3, respectively, but neither φ_1 nor φ_2 is true under the actual values of variables, i.e. *P* arrives at a deadlock state. We can avoid this undesired situation if we require that the following situation--descriptive formula is true in the Kripke model associated with *P*:

$$\Box((\text{at } l_1 \wedge \text{at } l_3) \rightarrow (\varphi_1 V \varphi_2))$$

Let us suppose now that *P* is a communicating process with two components p_1 and p_2:

$$l_1\text{:}p_1(x)?\xi(y)$$

occurs in p_2,

$$l_2\text{: } p_2(u)?\xi(v)$$

occurs in p_1. Again, it can happen that both components wish to execute the statements labelled by l_1 in the second and by l_2 in the other component. If this is the case, then *P* is in deadlock state .since both components are waiting for the arrival of messages from each other. This undesired situation can be ruled out by requiring that the formula

$$\Box \neg(\text{at } l_1 \wedge \text{at } l_2)$$

holds. Note that this formula is similar to the formula expressing mutual exclusiveness but, in this case, it rather prevents *P* from being deadlocked.

In connection with critical sections in components of a concurrent process *P* is the so called liveness property. Let us suppose, that the labels of the critical sections are l_1 and l_2, respectively in the first and second component of a (two-component) concurrent process *P*, and that the formula

$$\varphi_o \rightarrow \Box \neg(\text{at } l_1 \wedge \text{at } l_2)$$

expressing the mutual exclusiveness of the two components with respect to the labels l_1 and l_2 is true in the Kripke model associated with *P*. It can happen, however, that this formula is satisfied in such a manner that, for example, the second component never enters into its critical section labelled by l_2, that is, at l_2 will never be true (hence at $l_1 \wedge$ at l_2 will always be false, which, in turn, means that $\Box \neg(\text{at } l_1 \wedge \text{at } l_2)$ will always be true). This objectionable situation is avoided as follows. Let us suppose that the critical section labelled by l_2 appears in the form

$$(l_o\text{: } p_1;\ l_2\text{: } p_2)$$

in the second component. It is clear that if l_o: p_1 is actually under execution, then (by definition of sequenced processes) the second component wishes to execute l_2: p_2 next. Because of mutual exclusivity, l_2:p_2 can be executed eventually and not necessarily in the next step. But if we require that the formula

$$\text{at } l_o \rightarrow \Diamond \text{ at } l_2$$

should hold, then l_2:p_2 will eventually be executed. This property is called a kind of liveness of the second component at l_2. Note that this notion of liveness has no sense with communicating processes.

In the case of concurrent processes one component can be blocked by a malicious scheduler, too; i.e. if the scheduler never allows that component to take a step. This malignancy can be excluded by requireing that for every such part l:p of the component whose part is critical but defines no loop, the formula

$$\text{at } l \rightarrow \Diamond(\neg \text{ at } l)$$

is true in the Kripke model associated with the process at hand. If l:p defines a loop, e.g. the form of p is

$$\text{if } \varphi \text{ then } l_1 \text{ else } l$$

then the formula above will not force the scheduler to be benevolent, since it may well happen that φ will never be true which implies also that

$$\Diamond(\neg \text{ at } l)$$

will be false forever. Nevertheless, it is possible that φ is true at a particular moment of execution but just then the scheduler lets another component continue. The only thing that can be required is for φ to be true and remain true thereafter, whereupon the scheduler eventually chooses that component. This is formalized by the following situation-descriptive formula:

$$\text{at } l \wedge \Box\varphi \rightarrow \Diamond(\neg \text{ at } l).$$

A much stronger property of the scheduling is that if φ is true infinitely often, then the part labelled by l will eventually be executed. This is given by the formula:

$$\text{at } l \wedge \Box\Diamond\varphi \rightarrow \Diamond(\neg \text{ at } l).$$

In the case of communicating processes, every component is controlled by its own scheduler, which is, of course, benevolent.

These properties and their formalization by situation descriptive formulae are summarized in the following table.

Property	Situation-descriptive formula
"mutual exclusiveness with respect to labels l_1 and l_2"	$\Box\neg(\text{at } l_1 \wedge \text{at } l_2$
"no deadlock can occur at l_1 and l_3" (concurrent case)	$\Box((\text{at } l_1 \wedge \text{at } l_3) \rightarrow$ $\rightarrow (\varphi_1 \vee \varphi_2)$
"no deadlock can occur at l_1 and l_3" (communicating case)	$\Box\neg(\text{at } l_1 \wedge \text{at } l_2)$
"liveness"	$\text{at } l_o \rightarrow \Diamond \text{ at } l_2$

"the scheduling is bevevolent"	$at\ 1 \rightarrow \Diamond(\neg\ at\ 1)$ (no loop) $at\ 1 \wedge \Box\varphi \rightarrow \Diamond(\neg\ at\ 1)$ (loop)
"the scheduling is strongly benevolent"	$at\ 1 \wedge \Box\Diamond\varphi \rightarrow \Diamond(\neg\ at\ 1)$

6. DYNAMIC (FIRST ORDER) LANGUAGES

So far, we have introduced and analysed plenty of languages. All these languages are appropriate for describing Kripke models (notwithstanding, with different expressive power). The idea behind them was that to each run of a process, a Kripke model was associated. This means that we can describe any single particular execution of a process by these formulae, but it is impossible to deal with all possible executions at the same time. Clearly, by providing inference systems for these languages, we are able to prove several properties of every single run, but any property of the whole collection of runs of a process is completely intractable. For example, partial correctness of a process is to be understood as an assertion that all finite runs (i.e. all runs which terminate) end in a prescribed state. Kripke models, however, have power enough to contain all runs of a process, hence, by constructing appropriate languages the features of the collection of all runs can be investigated. Then, obviously, processes (the parts of the main process with which Kripke models are associated) must appear explicitly in formulae to be defined. Allowing this, we obtain another kind of languages, the so called dynamic languages [48][53-54][56-57][59][65].

For defining dynamic formulae, let us suppose that a type d is fixed, and consider P, the set of variables for processes and V, the set of ordinary individual variables. For all $p \in P$, let $\boxed{p}$ and $\langle p \rangle$ be two new logical symbols.

We can construct the set of dynamic formulae F_d^{dyn} by stipulating $F_d^{cl} \subset F_d^{dyn}$ and that F_d^{dyn} is the smallest set closed under the following two rules: if $\varphi \in F_d^{dyn}$ and $p \in P$,

then (i) $\boxed{p}\ \varphi \in F_d^{dyn}$

(ii) $\langle p \rangle\ \varphi \in F_d^{dyn}$.

According to their intended meaning one possible reading of these clauses can be: "for all terminating runs of p, φ will necessarily be true in the ending states" and "for some terminating run of p, φ will eventually be true in some state".

The models of these languages can simply be given by defining an accessibility relation $R_{p,\underline{d}_o}$ for each $p \in P$ and for every input data vector $\underline{d}_o$

in exactly the same way as was done in Chapter II, Sections 3 and 4.
We can assume that

$$R_{p_o,\underline{d}_o} \cap R_{p_o,\underline{d}_1} = \emptyset$$

and

$$R_{p_o,\underline{d}_o} \cap R_{p_1,\underline{d}_o} = \emptyset$$

for all p_o, p_1, $\underline{d}_o$, $\underline{d}_1$. (This assumption, however, is counterintuitive and is adopted for the sake of convenience. On the other hand, by admitting this assumption we do not restrict generality, since using simple technical tools, we can always distinguish between these relations.) It follows that

$$R_p = \bigcup_{\underline{d}_o} R_{p,\underline{d}_o} \quad \text{and hence}$$

$$R = \bigcup_p R_p$$

are binary relations on the set of states. Thus, (Q,R) is a Kripke model where R has a finer structure) i.e. dynamic languages again describe Kripke models.

Let $A^Q = \{a \mid a\colon V \to A\}$ be the set of (ordinary) assignments. Every $a \in A^Q$ can be extended to the set $V \cup P$ by defining

$$\underline{a}(p)(\underline{d}_o) = R_{p,\underline{d}_o} .$$

Let $A^{(Q,R)} = \{\underline{a} \mid a \in A^Q\}$. The relation $\models^{dyn}$ can be given in the standard way for classical formulae while for the new formulae we proceed by recurrence. Let

$$\underline{a} \in A^{(Q,R)} ,$$

then,

(i) $(Q,R) \models \boxed{p}\ \varphi[\underline{a}]$ iff for all data vectors $\underline{d}$, $(Q,R_{p,\underline{d}}) \models \Box\, \varphi[a]$

(ii) $(Q,R) \models \langle p \rangle\ \varphi[\underline{a}]$ iff for some data vectors $\underline{d}$, $(Q,R_{p,\underline{d}}) \models \Diamond\, \varphi[a]$.

Note that by these definitions $\boxed{p}$ and $\langle p \rangle$ differ somewhat from the similar definitions in [48][53-54][56-57]; moreover that $\neg\boxed{p}\neg\varphi$ is stronger than $\langle p \rangle\varphi$ which again makes some deviations in comparison with the usual definitions of the modal operators $\Box$ and $\Diamond$.

Clearly, we obtain a mathematical logical language $\langle F_d^{dyn}, M_d^{Kr}, \models^{dyn}\rangle$, the dynamic first order language. Observe that by introducing the "until" operator $\boxed{p\rangle}$ instead of $\boxed{p}$ and $\langle p \rangle$, we could define another language, the temporal dynamic language

$$\langle F_d^{temdyn}, M_d^{Kr}, \models^{temdyn}\rangle$$

along the line of reasoning of Chapter III, Section 4.

Some useful properties expressible by dynamic formulae are given in the table below.

Property	Dynamic formula
"p is partially correct with respect to the input condition φ and output condition ψ"	$\varphi \to \boxed{p}\ \psi$
"there exists a run of p, in the final state of which ψ is true"	$\neg \boxed{p} \neg \psi$
"there exists a run of p with input condition φ, in the final state of which ψ is true"	$\varphi \to \neg \boxed{p} \neg \psi$

7. ACTION LANGUAGES

When we allow the appearance of symbols for both state and process explicitly in formulae, we obtain a language which is very much stronger than any of the languages introduced so far. This is the so called language for the logic of actions, [20-21][38][41-42][66]. Here we shall restrict ourselves to the definition of two particular cases.

The first is simply obtained by putting together the definitions of F_d^{sit} and F_d^{dyn}. In this particular case, states appear only in the form of at l and after l; p processes occur in the form of modal operators $\boxed{p}$ and $\langle p \rangle$ (or $\boxed{p}\!\rangle$). These languages will be denoted by

$$\langle F_d^{lam}, M_d^{Kr}, \models^{lam}\rangle$$

and

$$\langle F_d^{lat}, M_d^{Kr}, \models^{lat}\rangle .$$

The alphabet of another kind of action language is obtained by adjoining a new logical symbol Δ (delta) to that of the classical one, and taking L, the set of labels to identify states and P, the set of actions, to identify processes. Let

$$F_d^{la\Delta} = F_d^{cl} \cup \{\text{at } l \mid l\in L\} \cup \{\Delta_{lp}\varphi \mid l\in L,\ p\in P,\ \varphi\in F_d^{cl}\}$$

The intended meaning of the new formulae of the form $\Delta_{lp}\varphi$ is: "if the action p is performed at state l, φ will be true in the resulting state", i.e. Δ_{lp} is a generalization of the "next" operation. The semantics of formulae Δ_{lp} is defined more exactly as follows.

Models of this language are again Kripke models in the form (Q,R). For each $p \in P$, let

$$\delta_p \subset \{f \mid f: Q \to Q\}$$

and let R be defined as $R = \bigcup_p \delta_p$. Clearly, R is a binary relation, hence (Q,R) is indeed a Kripke model. Then

$$(Q,R) \models \Delta_{1p}\ \varphi[a] \quad \text{iff} \quad \delta_p(a(1)) \models \varphi[a]$$

where a is an assignment in the sense of Section 6 above.

Of course, we can start with F_d^{mod} or F_d^{tem} instead of F_d^{cl} when defining formulae. Then we have three new languages

$$<F_d^{la\Delta}, M_d^{Kr}, \models^{la\Delta}>$$

$$<F_d^{lam\Delta}, M_d^{Kr}, \models^{lam\Delta}>$$

$$<F_d^{lat\Delta}, M_d^{Kr}, \models^{lat\Delta}> .$$

The overall picture of the languages introduced in this chapter is given by the following diagram; arrows represent set theoretic inclusion.

$$\begin{tikzcd}
 & & \langle F_d^{lat}, M_d^{Kr}, \models^{lat} \rangle \arrow[drr] & & \\
\langle F_d^{temdyn}, M_d^{Kr}, \models^{temdyn} \rangle \arrow[urr] & & \langle F_d^{lam}, M_d^{Kr}, \models^{lam} \rangle \arrow[u] \arrow[ddrr] & & \langle F_d^{lat\Delta}, M_d^{Kr}, \models^{lat\Delta} \rangle \\
 & & & \langle F_d^{sit}, M_d^{Kr}, \models^{sit} \rangle \arrow[uul] \arrow[ul] \arrow[ddddr] & \\
\langle F_d^{dyn}, M_d^{Kr}, \models^{dyn} \rangle \arrow[uu] \arrow[uurr] & & & & \langle F_d^{lam\Delta}, M_d^{Kr}, \models^{lam\Delta} \rangle \\
 & & & \langle F_d^{tem}, M_d^{Kr}, \models^{tem} \rangle \arrow[uu] \arrow[uuur] & \\
 & \langle F_d^{tem}, M_d^{ord}, \models^{ord} \rangle \arrow[urr] & & & \\
 & & & \langle F_d^{mod}, M_d^{Kr}, \models^{mod} \rangle \arrow[uu] \arrow[uuulll] \arrow[uuur] & \langle F_d^{la\Delta}, M_d^{Kr}, \models^{la\Delta} \rangle \\
 & & \langle F_d^{cl}, M_d^{cl}, \models^{cl} \rangle \arrow[ur] \arrow[urr] & &
\end{tikzcd}$$

CHAPTER IV. CALCULI

The aim of the present chapter is to complete the definitions of several logics by providing calculi to languages introduced in the previous one.

1. CLASSICAL LOGIC

Classical logic has many different complete and correct calculi. For example, Manna [50] gives an interesting calculus, the resolution calculus (for further details, see [16] [49] [62]). Another calculus, the natural (or Kanger-Gentzen) calculus is investigated in detail by Takeuti [67]. Both types of calculi are designed in such a way that the search for formal proofs (which is generally extremely difficult) is governed by simple rules, hence the construction of proofs even for computers is relatively easy and mechanical. (This is true, at least in principle. In practice, hovever, the search for proofs of nontrivial assertions is frequently far beyond the capacity of the largest computers - even by these well-governed inference systems.)

A third type of calculi, called Hilbert-style calculus is not so well--designed but it is very much easier to comprehend than the first two. This calculus is axiomatic, and can be given as follows:

<u>Axioms</u> For all $\varphi,\psi,\chi \in F_d^{cl}$, the following schemata are axioms:

HA1: $\varphi \to (\psi \to \varphi)$

HA2: $\neg\neg\varphi \to \varphi$

HA3: $(\neg\varphi \to \neg\varphi) \to (\psi \to \varphi)$

HA4: $(\varphi \to (\chi \to \psi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))$

HA5: $(\forall v)(\varphi \to \psi) \to (\varphi \to (\forall v)\psi)$ provided v does not occur freely in φ

HA6: $(\forall v)\varphi \to \psi$, where ψ is obtained from φ by substituting an arbitrary term t into every free occurrence of v in φ provided each (free) variable of t remains free after the substitution.

Inference rules

HR1: ψ is inferred from φ and $\varphi \rightarrow \psi$ (Rule of Detachment)

HR2: $\forall v\varphi$ is inferred from φ (Rule of Generalization)

Note that in the presence of equality, three additional axioms (or more precisely, axiom-schemata), for expressing that equality is a congruence relation, are needed. A famous and fairly deep theorem, Gödel's Completeness Theorem, states that this inference system is complete and correct.

If one wishes to prove a theorem φ, then one can proceed in two basic ways, viz to start with the axioms and, using inference rules, generate newer and newer formulae (from the axioms and from the formulae generated previously) until φ is generated, that is to use a forward or top-down way of reasoning. The drawbacks of this top-down approach, thanks to the huge number os possible steps at the very beginning of generation and thereafter, are obvious.

The other approach is a backward or a bottom-up way of reasoning, when one starts from φ to be proved, and looks for formulae from which it can be inferred and then to continue this searching in the same way to the formulae found in the previous steps until axioms are arrived. Nevertheless, this bottom-up way of reasoning is not well organized thanks to the Rule of Detachment: if ψ is to be proved by this Rule, an infinite set of candidates for φ exists and no general rule can be given to help the choice from this set. The Kanger-Gentzen calculus rules out this natural but inconvenient inference rule.

The Kanger-Gentzen calculus is usually given in terms of sequents. Let $\sum$ and Γ be two finite (maybe empty sets of sentences. By a sequent we mean the triplet $\sum \rightarrow \Gamma$. It can be interpreted as follows:

(i) Let $\sum = \{\varphi_1,\dots,\varphi_n\}$, $\Gamma = \{\psi_1,\dots,\psi_m\}$; then $\sum \rightarrow \Gamma$ is to be meant as

$\varphi_1\wedge\dots\wedge\varphi_n \rightarrow \psi_1\vee\dots\vee\psi_m$.

(ii) By convention, $\wedge\emptyset$ = true, $\vee\emptyset$ = false; according to this convention if $\Gamma = \emptyset$, then $\sum \rightarrow \Gamma = \neg(\varphi_1\wedge\dots\wedge\varphi_n)$; if $\sum = \emptyset$ then $\sum \rightarrow \Gamma = \psi_1\vee\dots\vee\psi_n$ and if $\sum = \Gamma = \emptyset$, then $\sum \rightarrow \Gamma$ is universally false.

Axioms For all $\varphi \in F_d^{cl}$, the sequent $\{\varphi\} \rightarrow \{\varphi\}$ is an axiom.

Inference rules ($\sum,\varphi = \varphi,\sum = \{\varphi\}\cup\sum$ everywhere)

KGR1 (Weakening) left: $\varphi,\Gamma \rightarrow \sum$ is inferred from $\Gamma \rightarrow \sum$

right: $\Gamma \rightarrow \sum,\varphi$ is inferred from $\Gamma \rightarrow \sum$

KGR2 (Factoring) left: $\varphi,\Gamma \rightarrow \sum$ is inferred from $\varphi,\varphi,\Gamma \rightarrow \sum$

right: $\Gamma \rightarrow \sum,\varphi$ is inferred from $\Gamma \rightarrow \sum,\varphi,\varphi$

KGR3 (Merging) left: $\Gamma_1, \varphi, \psi, \Gamma_2 \to \Sigma$ is inferred from $\Gamma_1, \psi, \varphi, \Gamma_2 \to \Sigma$

right: $\Gamma \to \Sigma_1, \varphi, \psi, \Sigma_2$ is inferred from $\Gamma \to \Sigma_1, \psi, \varphi, \Sigma_2$

KGR4 (Negation) left: $\neg\varphi, \Gamma \to \Sigma$ is inferred from $\Gamma \to \Sigma, \varphi$

right: $\Gamma \to \Sigma, \varphi$ is inferred from $\varphi, \Gamma \to \Sigma$

KGR5 (Conjunction) left: $\varphi\wedge\psi, \Gamma \to \Sigma$ is inferred from $\varphi, \Gamma \to \Sigma$ or from $\psi, \Gamma \to \Sigma$

ritht: $\Gamma \to \Sigma, \varphi\wedge\psi$ is inferred from $\Gamma \to \Sigma, \varphi$ and $\Gamma \to \Sigma, \psi$

KGR6 (Quantification) left: $(\forall v)\varphi(v), \Gamma \to \Sigma$ is inferred from $\varphi(t), \Gamma \to \Sigma$

right: $\Gamma \to \Sigma, (\forall v)\varphi(v)$ is inferred from $\Gamma \to \Sigma, \varphi(u)$

where t is an arbitrary term, u does not occur in Γ and Σ.

This inference system is again complete and correct (possibly with additional axioms for equality), as proved by Gentzen and Kanger. Observe that in constructing a proof for φ in a bottom-up way we can restrict the searching to the set of subformulae of φ, i.e. to those formulae of which φ is built up. Since the set of subformulae schemata of arbitrary φ is finite, this restriction considerably delimits the candidates of formulae by which φ can be proved.

The presentation of the resolution is lengthy, hence it is omitted here.

2. MODAL LOGIC

Calculi for modal logics are investigated in several textbooks [24][27][44]; other aspects are developed in [13]. There ara many different modal calculi, e.g. Hughes-Cresswell [44] give 30, Gabbay [24] gives another 25. We shall present only three from hughes-Creswell, the so called S4, S4.3.1 and S5 [44].

Calculus S4 is in fact the simplest by which processes can be investigated.

Axioms Let $\varphi \rightsquigarrow \psi$ stand for $\neg\Diamond(\varphi\wedge\neg\psi)$ and $\varphi \leftrightsquigarrow \psi$) for $(\varphi\rightsquigarrow\psi)\wedge(\psi\rightsquigarrow\varphi)$

S4 A1 $\varphi\wedge\psi \rightsquigarrow \psi\wedge\varphi$

S4 A2 $\varphi\wedge\psi \rightsquigarrow \varphi$

S4 A3 $\varphi \rightsquigarrow (\varphi\wedge\varphi)$

S4 A4 $((\varphi\wedge\psi)\wedge\chi) \rightsquigarrow (\varphi\wedge(\psi\wedge\chi))$

S4 A5 $((\varphi\rightsquigarrow\psi)\wedge(\psi\rightsquigarrow\chi))\rightsquigarrow(\varphi\rightsquigarrow\chi)$

S4 A6 $(\varphi\wedge(\varphi\rightsquigarrow\psi))\rightsquigarrow\psi$

S4 A7 $\Box\varphi\rightsquigarrow\Box\Box\varphi$

Inference rules

S4 R1 From φ and $\psi \leftrightsquigarrow \chi$, infer ξ, where ξ differs from φ only in having χ in some of the places where φ has ψ.

S4 R2 From φ and ψ, infer $\varphi\wedge\psi$.

S4 R3 From φ and $\varphi\rightsquigarrow\psi$, infer ψ.

Calculus S4.3.1 gives axioms and inference rules for treating the next operator.

<u>Axioms</u> ($\rightsquigarrow$ stands as for S4)

S4.3.1 A1 $((\varphi \rightsquigarrow \Box\varphi) \rightarrow \varphi) \rightarrow (\Diamond\Box\varphi \rightarrow \varphi)$

S4.3.1 A2 $\Box(\Box\varphi \rightarrow \Box\psi) \vee \Box(\Box\psi \rightarrow \Box\varphi)$

S4.3.1 A3 $\Box\varphi \rightarrow \Box\Box\varphi$

S4.3.1 A4 $\Box\varphi \rightarrow \varphi$

S4.3.1 A5 $\Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)$

S4.3.1 A6 φ where φ is a universally valid classical first order sentence

<u>Inference rules</u>

S4.3.1 R1 From φ, infer $\Box\varphi$ (Rule of Necessiation)

Calculus S5 is, in a certain sense, the strongest modal calculus.

<u>Axioms</u>

S5 A1 Axioms for S4

S5 A2 $\varphi \rightsquigarrow \Box\Diamond\varphi$

<u>Inference rules</u> The same as for S4.

It is proved in [13] that S4 augmented with the classical inference system is a correct and complete calculus for such modal languages where R, the accessibility relation of Kripke models, is a preordering, i.e. R is reflexive and transitive; and S5 (with the classical inference system, of course) is complete and correct for such languages where R is an ordering, i.e. R is reflexive, transitive and symmetric. In Hughes-Cresswell [44], S4.3.1, again with the classical axioms and rules, was proved to be complete and correct with respect to those languages where R is a discrete, linear preordering.

3. TEMPORAL LOGICS

Calculi of temporal logics are investigated in [60]; other aspects in [21]. Rescher and Urquhart treat 22 different calculi for temporal logics [60]. We shall give here only one of them - obtainable from the inference system S4.3.1. This is defined as follows.

<u>Axioms</u>

T4.3.1 A1 $\Box\varphi \rightarrow \varphi$

T4.3.1 A2 $\Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)$

T4.3.1 A3 $\Diamond\Diamond\varphi \rightarrow \Diamond\varphi$

T4.3.1 A4 $\Box(\Box(\varphi \rightarrow \Box\varphi) \rightarrow \varphi) \rightarrow (\Diamond\Box\varphi \rightarrow \varphi)$

T4.3.1 A5 $\bigcirc(\neg\varphi) \longleftrightarrow \neg\bigcirc\varphi$

T4.3.1	A6	$O(\varphi \rightarrow \psi) \rightarrow (O\varphi \rightarrow O\psi)$
T4.3.1	A7	$\Box\varphi \rightarrow O\varphi$
T4.3.1	A8	$\Box\varphi \rightarrow O\Box\varphi$
T4.3.1	A9	$\Box(\varphi \rightarrow O\varphi) \rightarrow (\varphi \rightarrow \Box\varphi)$
T4.3.1	A10	$(O\varphi \rightarrow \varphi) \triangleright \varphi \rightarrow O\varphi$ (induction axiom)

<u>Inference rules</u>

T4.3.1	R1	If φ is a classical universally true sentence, then, infer φ.
T4.31.	R2	From φ and $\varphi \rightarrow \psi$, infer ψ
T4.3.1	R2	From φ, infer $\Box\varphi$.

It can be proved that this calculus is correct and complete for the language $<F_d^{tem}, M_d^{ord}, \models^{tem}>$.

4. ON CALCULI FOR OTHER LANGUAGES

Other languages, introduced in the last three sections of Ch.III. are more problematic from the viewpoint of their calculi.

For situation-descriptive languages, as defined in Chapter III., it can be proved indirectly that complete and correct calculi exist. This can be done by a lengthy elimination of situation-descriptive formulae. Albeit such calculi exist, no explicit and direct one, for example in axiomatic form, is known to us at the moment.

Dynamic languages were proved to have complete and correct calculi for particular models, for the so called arithmetical universes [23][53-54]. For more general Kripke models, it is known that these languages have no complete calculi.

A similar remark applies to action languages in the form $<F_d^{lat}, M_d^{Kr}, \models^{lat}>$; languages with Δ as a primitive have complete and correct calculi as was shown in Hayes, Ecsedi-Tóth [21][38]. For these languages, however, again only the existence of such inference system was demonstrated and no particular system is yet known.

REFERENCES

[1] Andréka, H., Németi, I.: A Characterization of Floyd-provable Programs, Preprint of Math. Inst. HAS (1978)

[2] Andréka, H., Németi, I.: Classical Many-Sorted Model-Theory to Turn Negative Results on Program Schemes to Positive, Preprint (1979)

[3] Andréka, H., Németi, I., Sain, I.: Completeness Problems in Verification of Programs and Program Schemes, Preprint (1980)

[4] Andréka, H., Németi, I., Sain, I.: A Complete Logic for Reasoning about Programs via Nonstandard Model Theory, I. II., Theoretical Computer Sci., Vol. 17. No. 2 and 3. (1982)

[5] Bochmann, G.V.: Logical Verification and Implementation of Protocols, Proc. 4th Data Communication Symp., Quebec 1975, pp. 7.15-7.20

[6] Bochmann, G.V.: A General Transition Model for Protocols and Communication Services, IEEE Trans. on Comm., Vol. COM-28, No.4. April 1980, pp. 643-650

[7] Bochmann, G.V.: Architecture of Distributed Computer Systems (Lecture Notes in Comp. Sci. Vol. 77.) Springer-Verlag, New York, Berlin, (1979)

[8] Bochmann, G.V., Chung, R.J.: A Formalized description of HDLC Classes of Procedures, IEEE National Telecommunications Conference, (1977)

[9] Bochmann, G.V., Gecsei, J.: A unified Model for the Specification and Verification of Protocols, Proc. of IFIP' 77, North Holland Publ. Co. Amsterdam, New York (1977) pp. 229-234

[10] Bochmann, G.V., Joachim, T.: Development and Structure of an X.25 Implementation IEEE Trans. on Software Eng., Vol. SE-5, (Sept. 1979) pp. 429-439.

[11] Bochmann, G.V., Merlin, P.: On the construction of Communication Protocols, Proc. Int. Conf. on Computer Communication, 1980, pp. 371-378.

[12] Bochmann, G.V., Sunshine, C.: Formal Methods in Communication Protocol Design, IEEE Trans. on Comm. Vol. COM-28, No.4. April 1980, pp. 624-631.

[13] Bowen, K.: Model Theory for Modal Logic Kripke Models for Modal Predicate Calculi, (Synthese Library Vol. 127) Dordrecht Boston (1979)

[14] Brand, D., Joyner, W.H. Jr.: Verification of Protocols Using Symbolic Execution, Computer Networks, Vol.2., Oct.(1978), pp. 351-360.

[15] Brooks, M.: Determining Correctness by Testing, SRI AI Dept. of Comp. Sci. Memo AIM 336.

[16] Chang, C.C., Lee, R.C.: Symbolic Logic and Mechanical Theorem Proving, Academic Press New York, London (1973)

[17] Danthine, A., Bremer, J.: Modelling and Verification of End-to-End Transport Protocols, Computer Networks, Vol.2. (Oct. 1978) pp. 381-395.

[18] Danthine, A.: Protocol Representation with Finite State Models, IEEE, Trans. on Comm. Vol. COM-28, No.4 (April 1980) pp. 632-643.

[19] Ecsedi-Tóth, P.: Logical Basis for Spcification and Verification of Protocols, Proc. Math. Aspects of System Sci., (Salgótarján, 1982).

[20] Ecsedi-Tóth, P.: Intensional logic of actions, CL&CL XII, (1978) pp. 31-45

[21] Ecsedi-Tóth, P.: On a General Theory of Dynamic Phenomena, Preprint (1980)

[22] Ecsedi-Tóth, P., Turi, L.: Completeness and Interpolation Theorems in Lattice-Valued Logics, to appear.

[23] Fischer, M.J., Ladner, R.E.: Propositional Mocal Logic of Programs, Proc. 9th Ann. ACM. Symp. on Theory of Computing, Boulder (May 1977) pp. 286-294

[24] Gabbay, D.: Investigations in Modal and Tense Logics with Applications to Problems in Philosophy and Linguistics Synthese Library, Vol. 92 , Reidel Publ. Co., Dordrecht-Boston (1976).

[25] Gabbay, D.:(Axiomatizations of logics of programs, manuscript (1977)

[26] Gabbay, D., Pnuelli, A., Shelah, S., Stavi, J.: On the temporal analysis of fairness, Proc. 7th ACM Symp. on Principles of Programming Languages (Jan. 1980) pp. 163-173

[27] Gallin, D.: Intensional and Higher-Order Modal Logics, North-Holland--American Elsevier, New York, Amsterdam (1975)

[28] Gécseg, F. Peák, I.: Algebraic Theory of Automata, Akadémiai Kiadó (1972) Budapest

[29] Gergely, T., Szőts, M.: On the incompleteness of proving partial correctness, Acta Cyb. Tom.4. Fasc 1 (1978) pp. 45-57.

[30] Gergely, T., Ury, L.: Time Models for Programming Logics, Proc. Maghematical Logic in Comp. Sci. (Colloq. Math. Soc. J. Bolyai Vol. 26) North Holland, Amsterdam, New York (1981)

[31] Gergely, T., Ury, L.(A Theory of Interactive Programming, Acta Informatica, Vol. 17, Fasc. 1. (1982) pp. 1-20.

[32] Guttag, J.: Notes on Type Abstraction (Version 2) IEEE Trans. on Software Eng. Vol. SE-6 No.1. (Jan. 1980) pp. 13-23.

[33] Hailpern, B., Owicki, S.: Verifying Network Protocols Using Temporal Logic, NBS Trends and Application Symp. (May 1980) pp. 18-28.

[34] Hajek, J.: Automatically Verified Data Transfer Protocols, Proc. of the Fourth Int. Comp. Communications, (Sept. 1978) Kyoto pp. 749-756

[35] Harangozó, J.: Protocol Definition with Formal Grammars, Proc. Symp. Comp. Netw. Protocols, Liege (1978) pp. F6-1-10.

[36] Harangozó, J.: Formal Description of the Protocol Hierarchy, Proc. EUROCOMP' 78, London (1978)

[37] Harangozó, J.: Formal Description of Network Protocols, Dissertation (1978). Techn. High School Budapest.

[38] Hayes, P.: A Logic of Actions, Machine Intelligence 6, Edinburgh Univ. Press (1971)

[39] High Level Data Link Control Procedures. Proposed Draft International Standard on Elements of Procedures. ISO/TC/SG-6, N-1005 (1975)

[40] High Level Data Link Control Procedures. Frame Stuctures. ISO DIS-3309, 2 (1975)

[41] Hewitt, C.: How to Use What You Know, Advance Papers of The Fourth ICAI. Tbilisi, (USSR) (1975)

[42] Hewitt, C., Baker, H. Jr.: Actors and Continuous Functionals, MIT(LCS) TR-194 (1977)

[43] Hoare, C.A.R.: Communicating Sequential Processes, Comm. ACM. Vol. 21 (1978) pp. 666-677.

[44] Hughes, G.E., Cresswell, M.J.: An Introduction to Mocal Logic, Methuen, London (1974)

[45] Kripke, S.: Semantical analysis of modal logic I, normal propositional calculi, Zetischrift für Math. Log. Vol.9. (1963) pp. 67-96.

[46] Kripke, S.: Semantical analysis of modal logic II, non-normal modal propositional calculi, The Theory of Models (ed. J.W., Addison, L. Henkin, A. Tarski) North-Holland, Amsterdam (1965) pp. 206-220.

[47] Liskow, B.H., Zilles, S.N.: Specification Techniques for Data Abstraction, IEEE Trans. on Software Eng. Vol. SE-1, No.1. (March 1975) pp. 7-19.

[48] Lipton, R.: Reduction: A Method of Proving Properties of Parallel Programs, Comm. ACM, 18, 12 (Dec. 1975) pp. 717-721

[49] Loveland, D.W.: Automated Theorem Proving: A Logical Basis (Fund. Studies in Comp. Sci. Vol. 6) North-Holland (1978)

[50] Manna, Z.: Introduction to Mathematical Theory of Computation, McGraw Hill Publ. Co. New York (1974)

[51] Manna, Z., Pnueli, A.: The modal logic of programs, Proc. 6th ICALP (Lecture Notes in Comp. Sci. Vol. 74) Springer-Verlag, New York, Berlin (1979) pp. 385-409

[52] Merlin, P.M.: A Methodology for the Deign and Implementation of Communication Protocols, IEEE Trans. Commun., Vol. COM-24 (June 1976) pp. 614-616.

[53] Meyer, A.R.: Equivalence of DL, DL^+, and ADL for Regular Programs with Arrey Assignments, Internal Report, MIT (Aug. 1977).

[54] Parkh, R.: A Completeness Result for PDL, Proc. MFCS'78 (Lecture Notes in Comp. Sci. Vol. 64.) Springer-Verlag, New York, Berlin (1978)

[55] Postel, J.B.: A Graph Model Analysis of Computer Communication Protocols, Ph.D., Dissertation, UCLA, Eng-7410, (1974) p. 184

[56] Pratt, V.R.: Semantical Considerations on Floyd-Hoåre Logic, Proc. 17th Ann. IEEE Symp. on Found. of Comp. Sci. (1976) pp. 109-121.

[57] Pratt, V.R.: Models of Programming Logics, Proc. 20th Ann. IEEE. Symp. on Found. of Comp. Sci. (1979)

[58] Rasiowa, H.: Completeness in Classical Logic of Complex Algorithms, manuscript

[59] Rasiowa, H.: Algorithmic Logic, Multiple-valued, Extensions, Studia Logica XXXVIII (1979) 4, p. 317

[60] Rescher, N., Urquhart, A.: Temporal Logic, Springer-Verlat, Wien, New York, (1971)

[61] Rudin, H., West, C.H., Zafiropulo, P.: Automated Protocol Validation: One Chain of Development, Computer Networks, Vol. 2. (1978) pp. 373-380.

[62] Sandford, D.M.: Using Sophisticated Models in Resolution Theorem Proving, (Lecture Notes in Comp. Sci. Vol. 90) Springer-Verlag (1980) Berlin, Heidelberg, New York.

[63] Schindler, S.: Algebraic and Model Specification Techniques, Proc. Hawai Int. Conf. on System Sci, (Jan. 1980) p.13.

[64] Schultz, G.D., Rose, D.B., West, C.H., Gray, J.P.: Executable Description and Validation of SNA, IEEE Trans. on Comm. Vol. COM-28, No.4. (April 1980) pp. 661-677.

[65] Segerberg, K.: A Completeness Theorem in the Modal Logic of Programs, Notices Amer. Math. Soc. 24 (1977) A-552.

[66] Štepankova O., Havel, I.: Some Results Concerning the Situation Calculus, Proc. of MFCS'73 (1973, High Tatras) Math. Inst. Slovak Acad. Sci. (1973) pp. 321-326.

[67] Takeuti, G.: Proof Theory, (Studies in Logic Vol.81.) North-Holland / American Elsevier, Amsterdam, New York (1975)

[68] Teng, A.Y., Liu, M.T.: A Formal Approach to the Desigh and Implementation of Communication Protocols, Proc. COMPSAC 1978

[69] Teng, A.Y., Liu, M.T.: A Formal Model for Automatic Implementation and Logical Validation of Network Communication Protocols, NBS Computer Networking Symp. 1978, pp. 114-123

[70] Wand, M.: Induction, Recursion and Programming, North-Holland, Amsterdam, New York (1980).

[71] West, C.H.: General Technique for Communications Protocol Validation, IBM J. Res. Develop. Vol. 22, No.4. (July 1978) pp. 393-404

[72] von Wright, G.H.: An essay in deontic logic and the general theory of action, (Acta Philo. Fennica Vol. 21.) North Holland, Amsterdam, New York (1968)

[73] Zafiropulo, P.: Protocol Validation by Duologuw-Matrix Analysis. IEEE Trans. on Comm. Vol. COM-26, No.8. (Aug. 1978) pp. 1187-1194.

[74] Zafiropulo, P., West, C.H., Rudin, H., Cowan, D.D., Brand, D.: Towards Analyzing and Synthesizing Protocols, IEEE Trans. on Comm. No. 4. (Apr. 1980) pp. 651-661.

ALGORITHM FOR INVESTIGATION OF FORMALLY DEFINED COMPUTER NETWORK PROTOCOL CONFORMITY

P. TŐKE

Department of Numerical Methods and Computer Science
Eötvös Loránd University, Budapest, Hungary

ABSTRACT

Possible and realizable algorihms for the conformity examination of computer communication networks based on Digital Network Architecture (DNA) are dealt with. Such examination is possible only if the rules governing the communication of the entities of the layers of a given network architecture are defined by some formal methods. In the present paper algorithms are proposed for conformity examination when the communication rules or protocols are defined by finite algebraic automata and formal grammars. The possibility of applying the developed methods is demonstrated in the DDCMP governing the communication in the data link control layer of the DNA.

АННОТАЦИЯ

Условием надежного применения алгоритмов, служащих для исследования конформности сетей ЭВМ на базе DECNET, является определение совокупности свойств слоев сети данной архитектуры. Автором предлагается алгоритм для исследования конформности в таких случаях, когда правила коммуникации определяются средствами конечного автомата или формальной грамматики. Пригодность метода автор демонстрирует на протоколе DDCMP на процедуре управления коммуникацией данных.

KIVONAT

DECNET-bázisu számitógéphálózatok konformitásvizsgálatára szolgáló algoritmusok megbizható alkalmazásának előfeltétele, hogy az adott hálózatarhitektura rétegeinek entitásai formális módszerekkel legyenek meghatározva. A szerző konformitásvizsgálati algoritmusra tesz javaslatot olyan esetekre, amelyekben a kommunikációs szabályokat véges automata, vagy formális nyelvtan eszközeivel definiálják. A módszer alkalmasságát a szerző DDCMP protokollon (a DNA adatkapcsolatát vezérlő eljáráson) demonstrálja.

I. INTRODUCTION

In the realization of a computer communication network the unambiguous definition of the communication protocols is very important. On the other hand for the error free operation of a network based on a given network architecture it is necessary that the elements of the realized network operate in conformity with the corresponding functional units of the given network architecture.

The present computer network architectures have a layered structure. This means that the functions of the given architecture are classified into hierarchically ordered classes such that the layers are independent of each other. The functions of a layer are realized using the services presented by the lower layers in the hierarchy. In a given layer a function is the result of specific communication of the entities according to the protocols given by the communication architecture. The communication of the entities in a given layer is called peer communication between the peer entities. For the realization of the functions of a given layer, the services of the lower layers accessed through the service access point in the layer's boundary are needed. The access to the services of the lower layers are governed by interfaces.

The definition of a given network architecture comprises the definition of the hierarchical layering of the functions, the specification of the inner structure of the layers, the definition of the interconnection between the layers and the description of the rules of peer communications and interfaces.

A communication architecture contains no information for its realization by software and hardware elements but it is questionable whether a given realization does conform to the network architecture prescribed. On the other hand if we are to add a new element to a network we must check whether this element has the communication capability identical to that given in the communication architecture.

In both cases we must identify the communication capability of a real element of a network with the standard defined by a communication architecture.

The identification presupposes an abstract formal definition of the communication protocols. When defining formally the present network architectures the elements and methods of the theory of algeraic automata and formal grammars were applied.

In order to apply abstract algebraic automata the input and output functions are defined by high level programming languages and the connections between the automata are illustrated by diagrams.

The definition based on formal grammars is used mainly to define the protocols of the lower layers.

In the following sections the paper proposes theoretically based algorithms for the conformance examination and for the diagnosis of errors.

In Section II. the basic notions of the theory of algebraic automata and formal grammars are given.

Section III. formulates the conformance problem as that requiring the identification of algebraic automata. It refers to the theorems used in the development of the algorithm for the examination of the conformity and it deals with the structural properties of the automata and proposes methods to reduce the complexity of the algorithm suggested.

Section IV. concerns itself with the identity problem when the protocols are defined by formal grammars. These formal grammars are of type-3 in Chomsky's hierarchy and the languages generated by the grammars can be accepted by appropriate acceptor automaton which can be used in the conformity examination and error detection.

In Section V, we are able to see the possible application of the methods developed in the case of the DDCMP using the formal description given by Harangozó [3].

II. PRELIMINARY NOTATIONS AND GENERAL NOTIONS

Definition: A Mealy-type automaton is a system $M = (A,X,X,\delta,\lambda)$ where A,X and Y are (non-empty) sets; furthermore, δ: $A \times X \to A$ and λ: $A \times X \to Y$ are functions defined on $A \times X$. Sets A,X,Y are the sets of (internal)states, inputs and outputs, respectively. The functions δ and λ are called the transition function and the output function respectively.

In the present paper we deal mainly with the notion of a Mealy-type automaton so we omit the prefix "Mealy-type".

Let Z be a (non-empty) finite set. By the free semigroup F(Z) generated by Z we mean the set F(Z) of all (non-empty) finite sequences of elements of Z with the following rule of multiplication: if $p = x_1x_2...x_k$ and $q = x_1'x_2' ...x_l'$ are in F(Z) then $pq = x_1x_2...x_kx_1'x_2'...x_l'$. We shall use the names alphabet, letters and word for the set Z, elements of Z and element of F(Z) respectively. We consider the length $p = x_1x_2...x_k$ to represent $|p| = k$.

Let $M = (A,X,Y,\delta,\lambda)$ be an automaton. M is called initial if it has a distinguished $a_o(\in A)$ internal state. An initial automaton can be given in the form $M = (A,a_o,X,Y,\delta,\lambda)$.

Let $M = (A,X,Y,\delta,\lambda)$ be an arbitrary automaton and let $F_1(A)$, $F_1(X)$ and $F_1(Y)$ denote the free semigoups with identity element generated by A, X and Y respectively. The functions δ and λ can be extended to

$$\delta: A \times F_1(X) \to F_1(X),$$
$$\lambda: A \times F_1(X) \to F_1(Y),$$

respectively as follows. For an arbitrary state a ($\in A$) and input word $p = x_1x_2...x_k$ ($\in F_1(X)$) let

$$\delta(a,p) = a_1a_2...a_k \quad (\in F_1(A)),$$
$$\lambda(a,p) = y_1y_2...y_k \quad (\in F_1(Y))$$

where $a_1 = \delta(a,x_1)$, $a_2 = \delta(a_1,x_2),\dots, a_k = \delta(a_{k-1}x_k)$
and $y_1 = \lambda(a,x_1)$, $y_2 = \lambda(a_1,x_2),\dots, y_k = \lambda(a_{k-1}x_k)$

Let $M_{a_o} = (A,a_o,X,Y,\gamma,\delta)$ be an initial automaton.

The function

$$\alpha_{a_o} : p \to \lambda(a_o,p)$$

is called the function induced by the M_{a_o}.

Let a $(\in A)$ be an arbitrary internal state of the automaton M. By the function induced by the state a we mean the function induced by the initial automaton $M_a = (A,a,X,Y,\delta,\lambda)$.

Definition: Let $M = (A,X,Y,\delta,\lambda)$ be an automaton. The internal states a,b of M are equivalent if the functions induced by a and b are equivalent.

For an arbitrary automaton $M = (A,X,Y,\delta,\lambda)$, ϕ_A denotes the set of all functions induced by the states of M.

Definition: Let $M = (A,X,Y,\delta,\lambda)$ and $N = (B,X,Y,\delta',\lambda')$ be automata with identical inputs (X) and outputs (Y). The states $a(\in A)$ and $b(\in B)$ are said to be equivalent if they induce the same function.

If $\phi_A = \phi_B$ then the automata M and N are said to be equivalent.

Definition: Let $M = (A,X,Y,\delta,\lambda)$ be an automaton. The equivalence relation R on A is said to be a congruence relation if for all $a,b(\in A)$ and $x(\in X)$

$$aRb \Rightarrow \delta(a,x)R\delta(b,x)$$

and

$$\lambda(a,x) = \lambda(b,x)$$

An equivalence relation R on the set of states of M is said to have the substitution property it

$$aRb \Rightarrow \delta(a,x)R\delta(b,x)$$

for all $a,b(\in A)$ and $x \in X$.

An equivalence relation R on the set of the states M is output consistent if

$$aRb \Rightarrow \lambda(a,x) = \lambda(b,x)$$

for all a,b($\in$A) and x($\in$X).

<u>Definition:</u> Let $M = (A,X,Y,\delta,\lambda)$ be an automaton and let R be a congruence relation on A.
The automaton

$$M_R = (\{C_R[a]\}_{a\in A}, X, Y, \delta_R, \lambda_R)$$

is said to be the factor automaton of M induced by R where $C_R[a]$ denotes the class of A represented by $a\in A$ and

$$\delta_R\colon \delta_R(C_R[a],x) = C_R[\delta(a,x)]$$

$$\lambda_R\colon \lambda_R(C_R[a],x) = \lambda(a,x)$$

<u>Definition:</u> A formal grammar is a quadruple $G = (V_N, V_T, S, H)$ where V_N and V_T are disjoint alphabets, and H is a finite set of ordered pairs (P,Q) such that Q is a word over the alphabet $V = V_N \cup V_T$ and P is a word over V containing at least one letter of V_N. The elements of V_N are called non terminals and those of V_T terminals. $S\in V_N$ is called the initial letter. Elements (P,Q) of H are called productions or rewriting rules and are written $P \to Q$.

<u>Definition:</u> The formal grammar G is called left linear if each production has one of the two forms

$$X \to Yp \text{ or } X \to p$$

where $X,Y\in V_N$ and $p\in F_1(V_T)$.

<u>Definition:</u> The language L(G) generated by G is defined by

$$L(G) = \{p \mid p\in F_1(V_T) \text{ and } S\to^* p\}$$

<u>Definition:</u> Assume that the finite sets $X = \{X_1, X_2, \ldots X_n\}$ and $V' = \{\cup, *, \phi, (\ ,\ )\}$ are disjoint alphabets. A word p over $X\cup V'$ is a regular expression over X if:

(i) P is a letter of X or the letter ϕ, or

(ii) P is one of the forms $(R \cup Q)$, (QR) or Q^* where Q and R are regular expressions over X.

Each regular expression p over X defines a language $|P|$ over X according to the following conventions:

(i) Language $|\phi|$ is an empty language.

(ii) The language denoted by $|X_i|$ $(X_i \in X)$ consists of the word X_i.

(iii) For the regular expressions P and Q over X,
$|(P \cup Q)| = |P| \cup |Q|$, $|(PQ)| = |P||Q|$ and $|P^*| = |P|^*$

The languages defined by regular expressions over X are called regular languages over X.

Definition: The initial automaton $M_{a_o} = (A, a_o, X, \delta)$ without outputs accepts the language L over X with the subset A' of the internal states or in other woords the initial automaton M_{a_o} generates the language L with the subset A' of the internal states if

$$p \in L \iff \delta(a_o, p) \in A'$$

Theorem II.1. (Kleene, 1956) The languages generated by finite initial automata and only those are regular.

Theorem II.2. The languages generated by left (right) linear grammars and only those are regular.

III. ON THE CONFORMITY OF THE REALIZATION OF A GIVEN NETWORK ARCHITECTURE DEFINED BY ALGEBRAIC AUTOMATA

The elements of the theory of the algebraic automaton seem to be effective tools in the formal definition of network architectures [6-8, 10, 13-15].

The formal description by algebraic automata of a given network architecture means the definition of the function of the entities (functional units) of the layers of the network architecture by algebraic automata. The interconnection of the automata at their input and output corresponds to the inner relations between the elements of the given network layers.

Suppose that we are to develop a realization of a network which must conform to a given network architecture. During the development the question soon arises whether the function of a realized element conforms to the given network architecture. We are faced with the problem of deciding the equivalence of the

automaton model of the realised element with the (composite) automaton which corresponds to the functions of the entities realized.

For the algorithmic solvability of the conformance problem we need the following theorem:

Theorem III.1. Let $M = (A,X,Y,\delta,\lambda)$ and $N = (B,X,X,\delta,\lambda)$ be automata with common inputs (X) and outputs (Y), $|A| = m$, $|B| = n$. States $a \in A$ and $b \in B$ are non equivalent if there exists a $p \in F_1(X)$, $|p| \leq m+n-1$ and

$$\lambda(a,p) \neq \lambda'(b,p)$$

Collorary: If the set X of inputs is finite then the equivalence problem of states $a \in A$ and $b \in B$ can be decided in a finite number of steps. The equivalence problem of two finite automata with common inputs and outputs is algorithmically solvable.

From the equivalence of the two automata, it does not follow that M and N are isomorphic. Two equivalent automata M and N are isomorphic if and only if they are reduced.

Definition: The automaton $M = (A,X,Y,\delta,\lambda)$ is reduced if for an arbitrary pair $a,b(\in A)$

$$\alpha_a = \alpha_b \iff a=b$$

Theorem III.2. For every automaton $M= (A,X,Y,\delta,\lambda)$ there exists a factor automaton M_R which is equivalent to M and reduced.

Theorem III.3. If the automaton $M = (A,X,Y,\delta,\lambda)$ is finite then in a finite number of steps the reduced factor automaton M_R can be determined that is equivalent to M.

Proof. Let R_k be the equivalence relation on A:

$$aR_kb \iff \lambda(a,p) = \lambda(b,p)$$

for all $p \in F_1(X)$, $|p| \leq k$. It can easily be seen that

$$R_{k+1} \subseteq R_k$$

and

$$R_k = R_{k+1} \Rightarrow R_k = R_{k+j},\ j>1.$$

It can be proved that there exists such k, that

$$1<k \leq |A| - 1 \quad \text{and} \quad R_k = R_{k+1}$$

let k_o be the minimum of k having this property. It can be proved that for any pair a,b ($\in A$)

$$\alpha_a = \alpha_b \iff aR_{k_o} b \;.$$

and the relation R_{k_o} is congruent and the factor automaton M_{R_o} is reduced.

<u>Collorary.</u> In the case of finite automaton the reduction can be carried out in a finite number of steps.

From the practical point of view we can suppose that the automata describing the real network architectures are finite and reduced.

Let $M = (A,X,Y,\delta,\lambda)$ and $N = (B,X,Y,\delta',\lambda')$ be two reduced automata describing a functional unit of a given network architecture and the model of a possible realization of that functional unit respectively.

For any pair of a,b ($a\in A$, $b\in B$), $O(m^{|A|+|B|-1})$ number of operations is needed to decide their equivalence where $m = |X|$. This complexity can be reduced to a large extent if the automata M and N can be decomposed into more simple component automata. The decomposability of automata M and N depends on the given network architecture.

The following notions and theorems concern the algebraic structural properties and the decomposability of the algebraic automata.

<u>Definition:</u> The automata $M_1 = (A_1,X_1,Y_1,\delta_1,\lambda_1)$ and $M_2 = (A_2,X_2,Y_2,\delta_2,\lambda_2)$ for which

$$Y_1 = X_2$$

is a serial decomposition of the automaton M if

$$M = M_1 \ominus M_2 = (A_1 x A_2, X_1, Y_2, \delta, \lambda)$$

with

$$\delta((s,t),x) = (\delta_1(s,x), \delta_2(t, \delta_1(s,x)))$$

$$\lambda((s,t),x) = \lambda_2(t, \lambda_1(s,x))$$

<u>Theorem III.5.</u> The finite automaton M has a non trivial serial decomposition if and only if there exists a non trivial equivalence relation with a substitution property on the set of states of M.

<u>Definition:</u> The automata $M_1 = (A_1, X_1, Y_1, \delta_1, \lambda_1)$ and $M_2 = (A_2, X_2, Y_2, \delta_2, \lambda_2)$ are parallel decompositions of automaton it

$$M = M_1 || M_2 = (A_1 x A_2, X_1 x X_2, Y_1 x Y_2, \delta, \lambda)$$

with

$$\delta((s_1,s_2),(x_1,x_2)) = (\delta_1(s_1,x_1), \delta_2(s_2,x_2))$$

$$\lambda((s_1,s_2),(x_1,x_2)) = (\lambda_1(s_1,x_1), \lambda_2(s_2,x_2))$$

<u>Theorem III.6.</u> The finite automaton M has a non trivial parallel decomposition if and only if there exist two non trivial equivalence relations R_1, R_2 with the substitution property on the set of states of M such that

$$R_1 \cdot R_2 = \phi$$

The problem of conformity can be defined as follows. Let $M = (A,X,Y,\delta,\lambda)$ and $N = (B,X,Y,\delta',\lambda')$ be finite reduced automata describing a functional unit of a given network architecture and the model of the behaviuor of a realization of it respectively. Suppose that we can observe the function of the automaton from any initial state and that initial state can be reset. Let ϕ be a one to one mapping between the states of M and N.

It must be decided whether ϕ is isomorphy.

<u>Theorem III.7.</u> The problem of conformity is algorithmically solvable.

<u>Proof.</u> By theorem III.1. for all pairs of a, $\phi(a)$ their equivalence can be decided in a finite number of steps.

Let us suppose that the decision of the conformity of two automata M and N gives a negative result. It means that there exists at least one a ($\in A$) such that it is non equivalent to $\phi(a)(\in B)$. By theorem III.1. there exist at least one $p \in F_1(x)$ such that

$$|p| \leq 2|A|-1 \text{ and } \lambda(a,p) \neq \lambda'(\Phi(a),p)$$

The set D of such words p for all states a ($\in A$) is called the diagnostics of the conformity examination.

<u>Theorem III.8.</u> The set D of the diagnostics can be given in a finite number of steps.

IV. ON THE CONFORMITY OF THE REALIZATION TO A GIVEN NETWORK ARCHITECTURE DEFINED BY FORMAL GRAMMARS

The definition of communication protocols can be linked with the theory of formal grammars.

Let A denote the set of all possible states of a given entity E. Suppose that A is a finite set. Let i and e ($\in A$) denote the initial and final state of E respectively. The state of E is changed by receiving or sending messages. Let X denote the set of possible messages. We make a difference between the in- -coming and out-going messages even though they are of the same type.

We give, formally, all the series of messages that transforms E from state i to state e:

$$i \to^* e$$

where $\to^*$ means a finite number of interations of the one step transformations caused by reciving or sending a single message.

The transformations between the possible states of an entity E are restricted as follows.

Let E be in an arbitrary state a ($\in A$). For all messages $x(\in X)$ and states a ($a \in A$) suppose that there is at least a finite subset $A_{x,a}$ of the states which are all possible output states from state a ($\in A$) by applying the message x.

The function of E can be described by the following initial acceptor automaton M_i

$$M_i = (A, i, X, \delta)$$

with

$$\delta(a,x) = A_{x,a}$$

The function δ can be extended to

$$\delta: 2^A \times F(X) \to 2^A ,$$

Let

$$\delta(B,p) = \bigcup_{a \in B} \delta(a,p)$$

for all $B \subseteq A$.

Let L be the set of all strings of messages which transform the state of the entity E from i to e.

From the definition of the δ function the following theorem can easily be proved.

Theorem IV.1. Language L can be generated by automaton M_i with the subset {e} of A.

Theorem IV.2. Let G be the grammar canonically derived (see: [4]) from the atumaton M_i. Grammar G generates language L and is identical with the grammars given by Harangozó [1,2] for the classes of operation of the HDLC protocol.

Proof. Grammar G = (A,X,i,H) is defined by [4] as follows. For all $a,b \in A$ and $x \in X$

$$(a,xb) \in H \iff \delta(a,x) \ni b$$

Starting from this definition the theorem can easily be proved.

In some cases of communication protocols the final state of entity E is not defined (see: [3]). In this case language L consists of such $p \in F_1(X)$ for which

$$\delta(i,p) \neq \phi$$

Let $M_i' = (A \cup \{a_o\}, i, X, \delta')$ be an acceptor automaton, $a_o \in A$ and for all $a \in A$ and $x \in X$

$$\delta'(a,x) = \delta(a,x) \cup \{a_o\} \quad \text{if} \quad \delta(a,x) \neq \phi$$

$$\delta'(a,x) = \phi \text{ if } \delta(a,x) = \phi$$

and $$\delta'(a_o,x) = \phi \text{ for all } x \in X$$

<u>Theorem IV.3.</u> Language L can be generated by automaton M_i' with the subset $\{a_o\}$ of the extended states of E.

<u>Proof.</u> Let L′ denote the language generated by automaton M_i' with $\{a_o\}$. Assume first that $p \in L$;

$$p \in L \Rightarrow \delta(i,p) \neq \phi$$

From the definition of function δ' it follows that

$$\delta'(i,p) \ni a_o$$

consequently $p \in L'$.

Conversely, assume that $p \in L'$. It can be seen that

$$\delta'(i,p) \ni a_o \Rightarrow \delta(i,p) \neq \phi$$

which means that $p \in L$.

<u>Theorem IV.4.</u> Let G be the grammar canonically derived from the automaton M_i'. Grammar G generates language L and is identical with the grammars given by Harangozó [3] for the class of operation of the DDCMP.

<u>Proof.</u> Grammar $G = (A,X,i,H)$ is defined as follows:
For all $a,b \in A$, $x \in X$

$$(a,xb)\in H \iff b\in\delta'(a,x) \text{ and } b \neq a_o$$

and $$(a,x)\in H \iff a_o\in\delta'(a,x)$$

From that the theorem follows.

Assume that a communication protocol is defined by a right linear grammar G. There exists but not uniquely a finite acceptor automaton that accepts the language L generated by grammar G.

This ambiguity yields that the method developed in Section III cannot be applied in conformity examinations when the communication protocols are given by formal grammars.

The model of the realization of an entity E with the communication capabilities defined by a linear grammar G consists of an acceptor automaton which accepts the language L generated by grammar G, with an appropriate subset of states.

From the viewpoiunt of the conformity examination it is necessary to keep the number of states of the acceptor automaton to a minimum.

Let $\Lambda = \{b^D p^{(L)}\}_{p\in F_1(X)}$ denote the set of all left derivatives of language L. It is known that if grammar G is linear then set Λ is finite.

Define the equivalence relation $\equiv_L$ on $F_1(X)$ as follows:

$$p \equiv_L q \iff b^D p^{(L)} = b^D q^{(L)}$$

Theorem IV.5. Let ([p] denote the class of F(X) represented by p. The finite deterministic acceptor automaton

$$M = (\{C[p]\}_{p\in F(X)}, C[e], X, \delta),$$

where e denotes the empty word and

$$\delta(C[p],x) = C[px]$$

accapts language L with the subset $\{C[p]\}_{p\in L}$ of states.

It is a guess that automaton M has the minimum number of states.

V. APPLICATION POSSIBILITIES TO DECNET

In the development of a DECNET conforming to the Digital Network Architecture and in the conformity examinations encountered during the realization we can rely on the results of sections III and IV.

At the realization of the data link control layer we can use the formal description given by Harangozó [3]. The data link control layer is defined by a right linear formal grammar. The DDCMP specifies only the rules of the link set-up and data transfer and does not take into account the possible disconnection.

The grammar G defined in [3] derives the following language L

$$p \in L \iff \langle start \rangle \to^* p\langle s \rangle$$

where $\langle s \rangle$ is an appropriate state.

Let $= (V_N, V_T, \langle start \rangle, H)$ be the grammar given in [3] describing the operation of an entity in the data link control layer of the DNA.

Let us define the grammar $G' = (V_N', V_T', \langle start \rangle, H')$ where

$$V_N' = V_N \ , \ V_T' = V_T \text{ and}$$

$$H' = H \cup \{\langle n \rangle \to X \mid \exists m, m \in V_N \text{ and } \langle n \rangle \to X\langle m \rangle \in H\}$$

<u>Theorem V.1.</u> Language L′ generated by grammar G′ is identical with language L.

Language L is of type 3 and from that it follows that there exists an appropriate finite acceptor automaton M which generates L by a subset of states of M.

The acceptor automaton $M = (A, S, X, \delta)$ can be created in the following manner:

$A = V_N \cup \{A_o\}$ where $A_o \notin V_N'$ is an arbitrary symbol,

$S = \langle start \rangle, \quad X = V_T'$ and

$b \to x \notin H \Rightarrow \delta(b,x) = \{C \mid C \in V_N \text{ and } b \to xc \in H'\}$,

$b \to x \in H \Rightarrow \delta(b,x) = \{A_o\} \cup \{C \mid C \in V_N \text{ and } b \to xc \in H'\}$,

and

$$\delta(A_o, X) = \phi \quad \text{for all } x \in X.$$

Theorem V.2. The acceptor automaton $M = (A,S,X,\delta)$ accepts language L with the subset $\{A_o\}$ of the states of M.

The acceptor automata that accept language L are called line testers. Theorem V.2 states that there is a line tester for the class of operations of the DDCMP examined in [3]. The line testers give the possibility of the real time obsorvation of the data link control traffic and detection of the errors occurring during data link control communication.

According to theorem V.2 we can say that language L can be accepted by finite acceptor automata. Using the fundamental theorem of Kleene we can conclude that language L can be represented by a regular expression over the alphabet

$$T = \{\phi, e, \cdot, +, (,), {}^*\} \cup X$$

To coustruct a regular expression for language L we can use the following theorem.

Theorem V.3. (R.F. McNaughton and H. Yamada)
A regular expression representing language L accepted by a finite acceptor automaton M can be constructed algorithmically.

Proof. Let the states of M be indexed from 1 to n. Let us suppose that the index of the initial state is 1. Without any loss of generality we can suppose then that the "accepting" subset consists of one a_m element.

Let L_{ij}^k denote the language elements of L which transfer from state a_1 to a_j through the possible intermediate states $\{a_1, a_2, \ldots a_k\}$

For L_{ij}^k

$$L_{ij}^k = L_{ij}^{k-1} + L_{ik}^{k-1} L_{kk}^{k-1*} L_{kj}^{k-1}$$

holds. Using this egution we can say that the languages L^k_{ij} are regular for all possible i,j,k, and a possible regular expression for the language L^k_{ij} can be described in a finite number of steps.

It can be seen that

$$L^n_{1m} = L$$

From that the theorem follows.

A regular expression for language L very expessively describes the line traffic on the data link control level.

ACKNOWLEDGEMENT

I must express my special gratitude to Katalin Tarnai, Denise Ambrózi and József Harangozó for the many corrections and helpful suggestions resulting from a painstaking reading of the manuscript.

Very special thanks are due Miss Andrea Nagy and Mrs. Péterné Beron for pleasant accurate and fast typing service.

REFERENCES

1. Harangozo,J.:"Protocol Definition with Formal Grammars," Proc. Computer Networks Protocols,Liege 1978. pp. F6-I F6-IO.

2. Harangozo, J.:"Formal Language Description of a Communication Protocol,"Techn. Rep. No. 92, Central Research Institut for Physics, Budapest, Dec. 1977.

3. Harangozo, J.:"A DDCMP Adatkapcsolati Szintü Protokoll Formális Leirása,"Nem Publikált Anyag, 1982.

4. Peák, I.:"Bevezetés az Automaták Elméletébe," Egyetemi Jegyzet, 198O.

5. Trakhtenbrot, B.A., Barzdin, Ja.M.:"Finite Automata /Behavior and Systhesis/,"North-Holland Pub. Co. 1973.

6. Bochmann, G.V., Chung, R.J.:" A Formalized Specification of HDLC Classes of Procedures,"Proc. National Comm. Conf. Los Angeles, 1977. pp. O3A:2-1-11.

7. Bochmann, G.V.:"Finite State Description of Communication Protocols, "Computer Networks 2/1978/ pp. 361-372.

8. Danthine, A.A.:"Protocol Presentation with Finite-State Machine, "IEEE Tras. ln Comm. Vol. COM-28. No. 4. April, 1980. pp. 632-642.

9. Rudin, H., West, C.H., Zafiropulo, P.:"Automated Protocol Validation: One Chain of Development, "Computer Networks, 2/1978/. pp. 373-380.

10. Bochmann, G.V.:"A General Transition Model for Protocol s and Communication Services,"IEEE Trans. on Comm. Vol. COM-28. No.4, April, 1980. pp. 643-650.

11. Kawaoka, T., Yoshitake, S., Morino, K.:"A Method for Verifying Layered Protocol Product and Its Application to Data Communication Network Product," Proc. 4th. ICCC, 1980. pp. 379-384.

12. Tsukamoto, K., Itoh, T., Nomura, M., Tanaka, Y.:"A Study of Protocol Analysis for Packet Switched Network," Proc. 7th. Data Communication Symposium IEEE 1981. pp. 108-117.

13. Saito, T., Kato, T., Inose, H.:"Product Validation for Standardized Network Protocol,"

14. Sundstrom, R.:"Formal Definition of IBM's System Network Architecture,"Proc. National Telecomm. Conf. Los Angeles, 1977. pp. 03A:1-1-7.

15. SNA Format and Protocol Reference Manual: Architecture Logic SC30-3112-1 File No. S370-30.

16. Wacker, S.:"DNA: The Digital Network Architecture," IEEE Trans. on Comm. Vol. COM-28. No. 4. April, 1980. pp. 510-526.

17. Hartmanis, J., Stearns, R.E.:"Algebraic Structure Theory of Sequectial Machines,"Prentice-Hall 1966.

63-425