Formal methods and tools for the development of distributed and real time systems : Esprit Project 3096 (SPEC)

Formal Methods and Tools for the Development of Distributed and Real Time Systems

by

W.P.de Roever-H.Barringer-C.Courcoubetis
D.Gabbay-R.Gerth-B.Jonsson-A.Pnueli
M.Reed J.Sifakis-J.Vytopil-P.Wolper

90/1

January, 1990

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author or the editor.

Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

Formal Methods and Tools for the Development of Distributed and Real Time Systems *
SPEC – Esprit Project 3096

(Extended abstract)

January 22, 1990

W.-P. de Roever – coordinator
H. Barringer, C. Courcoubetis, D. Gabbay, R. Gerth, B. Jonsson, A. Pnueli, M. Reed, J. Sifakis, J. Vytopil, P. Wolper – partners

Context

The Basic Research Action No. 3096, Formal Methods and Tools for the Development of Distributed and Real Time Systems, is funded in the Area of Computer Science, under the ESPRIT Programme of the European Community. The coordinating institution is the Department of Computing Science, Eindhoven University of Technology, and the participating Institutions are the Institute of Computer Science of Crete, the Swedish Institute of Computer Science, the Programmimg Research Group of the University of Oxford, and the Computer Science Departments of the University of Manchester, Imperial College, Weizmann Institute of Science, Eindhoven University of Technology, IMAG Grenoble, Catholic University of Nijmegen, and the University of Liège. This document contains the synopsis, and part of the sections on objectives and area of advance, on baseline and rationale, on research goals, and on organisation of the action, as contained in the original proposal, submitted June, 1988. **The section on the state of the art (18 pages) and the full list of references (21 pages) of the original proposal have been deleted because of limitation of available space.**

Contents

*to appear in the EATCS bulletin 1990

1 Action Synopsis

This action addresses the difficult and important task of developing concurrent and distributed systems. It stresses the specific problems associated with real-time systems. The aim is to obtain formal and rigorous methods for the development of such systems.

Recent research has produced specification formalisms and associated development and verification tools that have been successfully applied to small examples. For instance, temporal logic, automata, process algebras, and assertional methods have all been used with some success. However, none of these formalisms completely solves the problem and all have their deficiencies.

The goal of this action is to alleviate these deficiencies and to provide frameworks for the specification and development of distributed real-time systems that are both practically adequate and theoretically sound. Our main insight is that the deficiencies of existing formalisms are mainly complementary and that *combining these formalisms* can be very effective. Besides combining formalisms, our research will proceed in two major directions: *broadening the scope of present formalisms*, and *narrowing the gap between specification and executable code*. We now give more detail about these directions.

Combining existing formalisms. We want to consider *combined* formalisms, which fuse together two or more complementary formalisms and thus combine the advantages of both.

Broadening the scope of existing frameworks. We want to be able to handle real time and reliability of systems. Most researchers have considered these topics either as a special (even though admittedly harder) case of concurrent systems, or as topics whose study should be postponed until we understand basic concurrency better.

Narrowing the gap between formal specification and executable code. We want to investigate two complementary approaches. The first follows the paradigm of development by which some abstract, high-level specification of the system to be constructed is transformed and refined into an efficient implementation (still satisfying the initial specification).

The second approach strives to obliterate the (at least in logic-based approaches) traditional dividing line between specification and implementation languages, through identifying an executable subset of the specification language and viewing the passage from a declarative specification into an imperative implementation as transformations occurring within the same formalism.

2 Objectives and Advances: Baseline and Rationale of the Action

Introduction

Few tasks are more important and challenging than building real-time or embedded systems. Moreover, the complexity of such systems is continuously increasing. This, together with the sobering experience that the difficulty in building such systems grows much faster than their size, makes a disciplined, systematic and rigorous methodology essential for attaining a reasonable level of dependability in such systems.

Responding to these considerations, an intense research activity has developed within the Computer Science community. This research has produced specification formalisms and associated development and verification tools that have been successfully applied to small examples. For instance,

- temporal logic,
- automata,
- process algebras, and
- assertional methods

have all been used with some success.

None of these formalisms completely solves the problem and all have their deficiencies. For instance, some properties can be hard to express in some formalisms whereas they are easy to express in others. Specifications written in a formalism that is not executable nor is amenable to algorithmic verification methods can be hard to debug. The scope and applicability of the various formalisms is not always sufficient as they are unable to take into account some important characteristics of the system being designed. Prime examples are real time constraints and scheduling policies.

The goal of this proposal is to alleviate these deficiencies and to provide frameworks for the specification and development of distributed real-time systems that are both practically adequate and theoretically sound. Our main insight is that the deficiencies of existing formalisms are mainly complementary and that *combining these formalisms* can be very effective. Besides combining formalisms, our research will proceed in two major directions: *broadening the scope of present formalisms*, and *narrowing the gap between specification and executable code*. These directions are discussed in more detail now.

Combining existing formalisms. We will consider *combined* formalisms, which fuse together two or more complementary formalisms. One of these suggestions, which recurs in the detailed plans of each of the individual groups, is the combination of a transition-based formalism, such as a finite-state automaton or a process algebra together with a logic-based formalism such as temporal logic. Indeed, transition based formalisms are ideal for describing local sequencing requirements, whereas logic-based formalisms are well suited for the representation of global requirements and constraints (including timing constraints). A step in this direction has already been achieved in the connection established between temporal logic and automata.

Broadening the scope of existing frameworks. We basically want to be able to handle real time. Most researchers have considered real time either as a special (even though admittedly harder) case of concurrent systems, or as a topic whose study should be postponed until we understand basic concurrency better. This leisurely approach is quite acceptable from theoreticians, but actual builders of systems see real time as the most crucial area in which formal support is necessary. We feel that the problems of real time can be effectively tackled by a concerted effort. Actually, some of the participants in this proposal have already worked on formalising these issues. Let us note that handling real time effectively will not only require the development of specification and development frameworks but might also require a revision of the basic models that have been used so far in dealing with concurrency. Besides real time, we are planning other

extensions of existing formalisms, for instance adapting them to genuinely distributed models of computation (for instance models including asynchronous communication) or to models containing probabilistic information. The latter could make it possible to handle reliability issues.

Narrowing the gap between formal specification and executable code. We will pursue two complementary efforts. The first follows the paradigm of development, by which one starts with an appropriately high level specification of the system to be constructed, and then, by a sequence of transformations, refinements, reductions, and simplifications, derives an efficient implementation of a program that satisfies the initial specification. Recommendations for an environment providing a partially automated support for this sequence of steps is an integral part of the proposal. This implies that we will have to consider subjects like compositionality, modularity and reification in our research on specification formalisms and the associated verification methods.

Our second effort to narrow the gap between abstract specification and executable code will strive to obliterate the (at least in logic-based approaches) traditional dividing line between specification and implementation languages. By identifying an executable subset of the specification language, we can view the passage from a declarative specification into an imperative implementation as transformations occurring within the same formalism. Such a unified approach has been proved very successful in the sequential case, as illustrated for example by PROLOG.

Much more will be learned from coordinated efforts progressing simultaneously in the three directions we just outlined than by individual efforts pursuing just one of these. For instance, work on real time can benefit from combined, more convenient, specification formalisms and from more executable specifications. Conversely, it is common that by working on a harder problem (e.g. real time) one finds better solutions for a simpler problem (e.g. traditional concurrency). Finally, working on systematic development frameworks can yield new ways of integrating different specification formalisms and benefiting from their respective advantages.

The participants in this proposal have been responsible for significant developments in the theoretical aspects of the specification and development of concurrent systems (e.g., temporal logic). They are involved in building tools that use these developments, and in applying these tools to real problems.

To fully benefit from the large potential cross-fertilization between the three main directions we have outlined, we are planning to organize our research not so much according to these direction but rather along an axis going from the theoretical aspects (e.g. models) to the more applied ones (real life applications). At most points on this axis each of the three proposed directions will be present. The main points we will consider are the following:

1. Models for concurrency, reliability and timing.
2. Specification formalism, interrelations and combinations.
3. Theories of verification and refinement.
4. Executable subsets of specification formalisms.
5. Tools and supporting environments.
6. Applications.

Although this list suggests a hierarchy with dependencies going upwards, work on these topics is more closely interconnected. For example, the verification and refinement theories that we develop will influence our specification formalisms. Indeed, these theories express the way in which we want to reason about systems. And, in our view, the applications should strongly influence, even drive, the theories, formalisms, models and supporting tools that we study. See section 8 for more details.

Next, we consider in more details our six general topics, and outline the main directions to be taken in each. Additional details are provided by the individual group plans.

3 Models for concurrency reliability and timing

3.1 Established theory

Ignoring the timing of systems, most researchers take one of two approaches to modelling the occurrences of events in a concurrent system: either, they describe the possible orders in which the events may occur, or they describe the causal relations that exist between the occurrences of events.

The first type of models, based on the *interleaving* of events, totally orders event occurrences, even if such occurrences originate in different unrelated parts of the system and are causally independent. The main advantage of this approach is the relative simplicity of the underlying mathematics.

The alternative type of models, based on "true" concurrency, use partial orderings to describe explicitly the fact that some events are independent, and may therefore occur concurrently, while others are causally related. Such models offer a more faithful picture of a system's behaviour, because they distinguish between non-determinism and non-causality.

If we consider the timing of systems, neither interleaving nor partial order models provide an adequate description. Although partial order models do describe the independence of events, and hence their potential concurrence, they do not identify which events actually concur and they do not specify the duration of events, as indeed interleaving models fail to do, too.

For systems that can be considered as globally clocked, so that all event occurrences can be related w.r.t. a single time-scale, there are some promising generalizations of interleaving models that do describe timing behaviour. These models are based on the assumption that enabled transitions are taken within some a-priori bounded delay period and were developed by some of the participants of this proposal. If no delay at all is incurred, these models are also known as *maximal parallelism* models.

Another aspect the models considered above do not support explicitely is the notion of reliability. They can be adapted to some representation of failures, such as fail-stop models. However, a more realistic treatment of reliability requires the introduction of probabilistic information concerning faults.

3.2 Directions of research

We propose to concentrate on the following problem areas:

3.2.1 Modelling time

The established theory as given above deals with execution models for concurrency, i.e. different ways of modelling occurrences of events in a concurrent system. These execution models all have an underlying notion of time incorporated. This fundamental notion of time can be decoupled from the execution models built upon it. In this way we can study the possible models for time itself separately (especially in the context of real-time). Such models can be differentiated by aspects such as

- properties of the ordering like
  - total/partial
  - discrete (e.g. the integers) / dense (e.g. the rational and real numbers)
  - complete (e.g. the integers and the real numbers)
- what operations on time are needed (addition, subtraction ...)
- whether time has a definite starting point or not, e.g. the natural numbers vs. the integers, or the non-negative reals vs. the complete set of reals etc.
- whether time is point-based or based on intervals.

Although some of the above aspects may be linked with certain execution models (total ordering with interleaving, partial ordering with true concurrency, intervals with overlapping etc.), it is interesting to investigate combinations of certain execution models with different models of time, e.g. interleaving can be based equally well on discrete as on dense time.

3.2.2 Interleaving versus partial order models

It is important to study relations between these two classes of models, thus laying the foundation for extending specification and verification techniques, proven successful for interleaving models, to models of "true" concurrency. Among other questions, we will consider the possibility of refining the model itself through the development process. For example, one may start by giving a specification in terms of the interleaving model and then express some design decisions, such as the decision to implement the system on four parallel processors, using the partial order model [Pnu86].

Another important issue is modelling events other than as point-actions. Obviously, models in which, e.g., the occurrence of events can overlap without necessarily concurring are more realistic. Although this claim is ultimately motivated through considering timed models in which events have duration, we initially wish to study its consequences independently from such complications. Here, too, we need to know the relations with other models and their use in the development process.

3.2.3 Models for communicating systems

The modelling of communicating systems has motivated the study of new models (synchronization trees, acceptance models, failure models...), for essentially two reasons:

- Traditional relational semantics and trace semantics have been proved to be not sufficiently powerful to model essential aspects of their behaviour.
- A communicating system being designed by composition of its components, a notion of encapsulation is needed, which is introduced at the model level by using an observability criterion.

The following problems deserve study:

- Study of relationships between these different models.
- Definition of models whose semantics are "optimal" with respect to significant properties of communicating systems.

3.2.4 Introducing time and resource constraints

The timed models developed by some of the participants have a number of deficiencies. One aspect that these models do not fully cover is the influence of limits on computational resources. These limits give rise to *scheduling*, *process priorities* and *interrupts*. Such implementation details have a significant impact on the timing of systems and, hence, can no longer be ignored in the context of this proposal.

The main direction of research in these problems is to conceive of appropriate abstractions, by which such implementation-dependent factors can be represented. Such abstractions should be able to provide a reasonable representation of constructs such as interrupts and time-outs which are essential to the detailed description of a real-time system.

A second area in which these timed models are deficient, is that of loosely coupled distributed systems. The assumption of a common time-scale is sensible for tightly coupled or single processor systems but becomes moot for, say, a country-wide airline reservation system. For all practical purposes, such systems should be considered to consist of independently executing and communicating agents with no strong causal connections between them. To model such systems, we want to extend partial order models — describing the independence of the agents — with timing information about the individual agents.

3.2.5 Probabilistic Models for random selection and failure rates

There are at least two motives for introducing probabilistic notions in models. The first is that some distributed algorithms actually have a random component. The second is that some features, e.g. fairness, failure rates, etc. require probabilities to be accurately modeled. For example, much of the current work in protocol design is driven by the need to overcome the unreliability of the transport medium. However, when using formal methods to verify a protocol, one is forced to make unrealistic assumptions since the probability of failure across the medium is not quantifiable. Note that once the basic probabilities of failure of the components of a system are known, the probability of failure of the combined system can be determined in a systematic way (see section 4.2.4).

One important aspect that will be considered is how the introduction of probabilities interacts with the other aspects of models considered above.

3.2.6 Model refinement and reification

A common thread runs through the research directions discussed above: the various models that will be developed must take their place in a *hierarchy* (or lattice) of models, that guides the development or refinement of system specifications. For instance, implementation decisions such as the number of available processors or particular scheduling strategies, ideally, should not appear in the initial model that describes a system's required behaviour. Such decisions should appear in subsequent refinements of the model, that are more implementation oriented.

This refinement process must be studied in depth. The correct notions of refinement must be established together with suitable refinement criteria.

Another central issue is what we have dubbed *reification*. Received wisdom has it, that system interfaces do not change during the refinement process. Stated differently, the interface that is used is that of the most concrete level on which the system is modelled. Conceptually, this position is not tenable: the fact that a certain event in reality corresponds to a signal that appears on a 16 bit wide parallel port must be introduced only at the level at which this becomes relevant to the modelling of the system. The same should hold for various refinements of data structuring during development.

Hence, *reification* (of interfaces) must be studied on its own and integrated into the model refinement process.

4 Specification formalisms, interrelations and combinations

4.1 Existing Formalisms

In this section we consider the existing formalisms that will be the starting points of our research efforts. We group them into four broad categories.

4.1.1 Temporal Logics

Temporal logics were first studied in philosophical logic as a framework in which to reason about time and related concepts such as tenses. It was introduced to the computer science community by Pnueli who proposed it as a tool for reasoning about concurrent programs. Pnueli's approach is based on the similarity that exists between the sequence of successive states of the world that temporal logic was designed to reason about, and the sequence of states a program goes through during its execution. Since Pnueli's first paper, computer scientists (among which several participants in this proposal) have introduced and studied a number of versions of temporal logic. These start with the linear version of temporal logic (the one used by Pnueli in his original work) and its more expressive extensions like the *extended temporal logic* introduced in [Wol83], the *fixpoint temporal logic* of [BKP86, BB86], the *dense temporal logics* of [BKP86, SZ85], and the *real time temporal logics* of [KdR85].

As opposed to linear time temporal logic, where each time instant has a unique immediate successor, branching-time temporal logic [BAMP81] allows each time instant to have several direct successors. This can be useful, for instance, in explicitely modeling the non-determinism of a program. In a similar way, versions of temporal logic suitable for reasoning about partially ordered computations, e.g. those of a distributed system, have been introduced [PW84]

The success of temporal logic is due in part to its ability to express properties of concurrent programs, but also to the fact that it is the basis of a number of verification methods, some of which are algorithmic. The most noteworthy of these is model checking in which one can algorithmically check that a finite-state program verifies a propositional temporal logic specifications. Tools based on this approach have been implemented, for instance, the system Xésar [RRSV87].

4.1.2 Process Algebras

Process algebras are term algebras whose constants represent atomic processes and whose operators represent process constructors. Their semantics is usually given as a transition relation on terms that represents the effect of actions, together with a congruence relation on terms that is preserved by transitions.

A variety of process algebras has been proposed; however, all of them have some common features. For instance, they all include operators corresponding to functional or sequential composition, nondeterministic choice, and parallel composition of processes. Furthermore, different semantic theories have been developed on these algebras by considering various classes of models (bisimulation semantics, acceptance semantics, failure semantics etc). These semantics naturally induce a congruence relation on the terms, which can be used to compare processes (represented by terms) and hence can form the basis of verification tools.

An interesting generalisation of these algebras is to consider a formalism in which an expression denotes a *set* of processes, rather than a single process. This has been proposed in [GS86] and is an example of our efforts to combine formalisms. Indeed, considering terms that denote sets of processes makes process algebra easier to integrate with logic based formalisms. This is because a single logic formula rarely specifies a single process, but rather a family of processes, which are all feasible implementations of that specification.

4.1.3 Automata Methods

Specification by state-transition formalisms, such as finite automata, is one of the most appealing and natural methods for practitioners working in system building. Moreover, for finite-state transition systems, one can algorithmically check some properties of the system; for instance, the accessibility of (un)desirable states. Of course these methods are limited by the explosion of the state space that occurs in concurrent systems.

It is quite interesting that state-transition formalisms are closely related to the two formalisms we have just discussed. Indeed, as we have seen, the semantics of process algebras is often given in terms of transition systems. Hence, a natural link. Although, traditionally, automata methods have been associated with simple trace semantics, whereas process algebras are usually associated with more discriminating semantics, such as acceptance semantics or even observational equivalence.

The link with temporal logics, although less obvious at first glance, is even closer. Indeed, the semantics of the linear version of temporal logic is also defined in terms of traces. The only difference being that temporal logics consider infinite traces whereas automata are usually oriented towards finite traces. However, there is a theory of automata on infinite behaviours (ω-automata) that has been extensively studied. It has been shown that there is a very close relationship between these ω-automata and temporal logic. Basically, for any propositional temporal logic formula, one can construct an ω-automaton that accepts exactly all the sequences satisfying the formula.

This correspondence is the theoretical basis of the combination of temporal logic and automata specification methods. It has already been used fruitfully, for instance in the work of Wolper where it is shown that many temporal logic verification problems can be recast in automata-theoretic form. Also, using ω-

automata enables finite-state verification methods to handle properties that were previously outside of their scope.

These ideas have already been incorporated in prototype verification systems built by participants in this proposal. For instance, Courcoubetis has constructed an automatic system that supports specification and verification by ω-automata. Also, a structured version of communicating finite automata has been proposed by the Statechart language, whose semantics and applicability has been studied by Pnueli and by the Eindhoven group.

4.1.4 Assertional methods

The classical approach to the specification of partial correctness of sequential programs, by a pair of assertions - a pre-condition and a post-condition, has been extensively extended over the past several years to cover general safety properties of concurrent systems. By introducing invariants over communication histories, and similar constructs, into the basic correctness construct $\{p\}S\{q\}$, these methods can now specify and verify general safety properties, [Zwi89]. Recent extensions in [Hoo87] already enable the specification and verification of timing constraints for globally clocked systems (see 3.1). The obvious advantage of these methods is their compositionality and modularity [Zwi89], which is an essential ingredient for any theory supporting well-structured development.

4.2 Directions of research

Specification formalisms are central to the problem of developing reliable and concurrent real-time systems. Hence, there are many directions for research we associate with this topic. Some of these topics go beyond the narrowly defined scope of specification formalisms and address the use of such formalisms in system development and verification.

4.2.1 Combined formalisms

Many of the participants in this proposal, each in his own way, have come to realize that, while his favorite formalism enjoys numerous advantages, it could benefit from some of the characteristics unique to other approaches. The idea of combining formalisms is thus prevalent in many of the individual plans.

The most seemingly promising combination is that of a logic based approach and of a transition-based approach. In the logic based approach, the behaviour of the system is described by formulas of a logic. Typical examples of logic used for this purpose are the various program logics and in particular temporal logics. In the transition based approach, the system is described in a language directly interpretable by some abstract machine (transition system).

One of the advantages of the logic based approach is that logic specifications are *conjunctive* in character. That is, a logic formula represents a class of behaviours and the specification of a system is given as a conjunction of formulas describing the intersection of the classes of behaviours specified by the conjuncts. As a consequence, logic specifications are easily modifiable and the specification process is flexible. Another obvious advantage of this approach is abstractness, i.e., independence w.r.t. implementation choices.

Compared to logic based formalisms, transition based formalisms have some very attractive features, too. They use very primitive concepts: the concepts of state and transition. Descriptions are concrete and easier to understand by people not familiar with logics. Furthermore, the possibility of graphic representation is a substantial advantage.

Logics and transition systems are also different in the type of properties they describe easily. Logics lend themselves better to the description of global properties such as termination, mutual exclusion and fairness. On the contrary, transition based formalisms are better adapted to the characterisation of situations where there exist "tight" relations between events, such as sequencing and precedence properties. Clearly, such formalisms, by their nature, cannot directly express global properties at all.

As we have alluded to in previous sections, logics and transition systems are not incompatible in their principle. Given a suitable logic and a suitable transition system, it is possible to establish an exact correspondence between logic based and transition based specifications. This is for instance the case for propositional temporal logic and finite-state ω-automata. Similar relations can be obtained between process algebras and transition based systems.

All these considerations make it very desirable to develop specification languages that combine logic based and transition based approaches. Examples are combining process algebras with temporal logic, embedding Lustre within general temporal logic, adding temporal logic (and timing) constraints to automata or Statechart specifications, and many others. Note that even if some theoretical results show the possibility of building such combined specification languages, the problem is far from being completely solved. Substantial work still needs to be devoted to making these formalisms practically usable.

4.2.2 New or extended specification formalisms

This section addresses new or extended specification formalisms at the exclusion of extensions geared towards real-time and reliability, which are the subjects of special sections below. There are several reasons for introducing new specification formalisms. One of the most basic of these is a change in the underlying model. For instance, if one goes from a total order model of computation to a partial order model, the specification formalism has to be adapted accordingly. This has, for instance, prompted the development of temporal logics geared towards partial orders. Nevertheless, it is not always sufficient to adapt an existing formalism to a new model. Indeed, the new model might be able to represent some properties that the existing formalism cannot describe. For example, stating that a particular program always has a parallelism degree of two (which means that if we allocate it two processors, they will never be idle) requires a specification formalism based on a partial order model and doesn't even make sense in a total order (interleaving) model.

Another direction is, within a given model, to find the most suitable formalism in which to express a given property. In general, one is trying to establish a delicate balance between ease of expression, generality and the amenability of the formalism to verification methods. In most case, the more general the specification formalism, the harder the associated verification method, but the more easier its use in specification becomes. For example, one can express much more with first-order temporal logic than with propositional temporal logic, but whereas propositional temporal logic is decidable and can be the basis of algorithmic verification methods, first order temporal logic is not even completely axiomatizable. It is thus very important to find ways of expressing properties in the weakest possible formalism. For example, it was shown in [Wol86] that specifying an unbounded buffer, which technically requires first-order temporal logic, could be done in propositional temporal logic given a natural and usually verifiable assumption on the implementation.

On the other hand, some classes of properties require, in principle, second order extensions. Examples of such properties are order preservation of messages on lossy channels (see [Koy87]) and *reification relations* (see section 3.2.6). Reification typically relates high-level event occurrences with *sequences* of low-level event occurrences — namely, those that implement the high-level events. Both classes of properties require the ability to label or partition (sequences of) event occurrences. We believe that this type of approach to extending the applicability of specification formalisms can be practically significant.

4.2.3 Extensions due to real time

We will consider extensions to real time for all the formalisms considered above. These extensions should include the ability to express timing constraints such as

- maximal time (e.g. time-out)
- cycle time (e.g. clocks, periodic behaviour)
- dead time (e.g. delay)

- minimal time (e.g. the maximum rate of stimuli from the environment).

Such extensions should be able to express these constraints explicitely but are not restricted to it. All the considerations we mentioned above, that call for the development of new models, which represent in an abstract way implementation-dependent elements such as scheduling policy, priorities and interrupts, should have a counterpart in the specification formalisms.

4.2.4 Incorporating timing and reliability information

It is not unusual that in the timing analysis of concrete real-time systems, one can use fixed bounds on the time necessary to execute the components of the system. Once these bounds are known, it is possible to check that the overall behaviour of the system satisfies some global constraints. Basically, this is done by computing the time necessary to execute the relevant parts of the system from the time necessary to execute the low level components of these parts.

To make this style of analysis systematic, one can use the paradigms of type and type checking. The timing information associated with a module becomes a type of this module. Then, given rules to derive the type of a module from the type of its components, it is possible to check that the "timing type" of the module satisfies given requirements. Consequently, the problem of checking that the system satisfies its timing requirements can be recast in the form of a type checking problem. The adavantage of this is that one can rely on the existing theory of types. Also, the problem of constructing a system satisfying real-time constraints can be viewed as building a program from some type-theoretic specification. This makes existing formal frameworks for this problem usable. Note, however, that in this approach one has, at some level, to rely on the existence of components for which the timing information is known.

One can imagine applying the same paradigm to reliability information. The types used would then represent, for example, probabilities of failure. The type checking would then amount to verifying that, given the probabilities of failure of its components, the probability of failure of the system is within some given bounds. Of course, one issue this approach does not address is how to assign failure probabilities to the basic components of the system and how to determine what failures are independent.

5 Theories of verification and refinement

Obviously, neither new models, nor new or extended specification formalisms, get us any closer to the development of concurrent and real-time systems, without the supportive verification and transformation theory. This theory enables us to provide the formal justification for each transformation or development step we perform on the way from specification to implementation. Naturally, well developed sound theories already exist for each of the formalisms listed above. Some of the participants in this proposal were instrumental in developing and improving these theories. However, once the additional factors of real-time and reliability are added, and the models and formalisms modified, extended or combined, a corresponding change is to be expected in the verification theory.

It is important to realize that the verification theory associated with a formalism is not a secondary consequential derivative of the formalism, but rather a very significant aspect of the formalism. The success of a formalism is judged both by the ease and convenience of specifying in it, and by the ease and convenience of verifying valid properties. In fact, there are some very important aspects to the general utility of a formalism that are only brought out by its verification theory. We mention *compositional* and *modular* completeness and the ability to *refine* and *reify*.

Compositional and modular completeness express the manner in which we want to reason within the verification theories and make precise Dijkstra's principle of "separation of concerns".

The principle of *compositional program verification* asserts that the specification of a program should be verified on the basis of specifications of its constituent components, without knowledge of the interior construction of those components.

Compositional completeness of a verification theory is the requirement that whenever a program satisfies a certain specification it should be possible to *deduce* the validity of that specification in a compositional way using the proof theory.

Modular completeness, formulated explicitly for the first time in [Zwi89], is the stronger and more interesting principle: whereas the former notion only requires, for a given specification of the whole program, the *existence* of appropriate specifications for the parts from which the specification of the whole can be deduced, modular completeness demands that such a deduction can be found using *a priori given* specification for the parts.

Thus, modular completeness is seen to reflect the practice of large programming projects, where, ideally, the separate modules are designed only knowing the given *specifications* of the other modules.

For specification frameworks limited to invariance properties for CSP/CCS-like languages, these completeness problems have been investigated in [Zwi89]. We intend to address these questions for the verification theories of some of the formalisms considered in section 4.1.

The refinement and reification notions that we intend to develop for various models (see section 3.2.6) require verification theories too, since such models will in fact be described using the specification formalisms of section 4.1. Such theories must be evaluated both w.r.t. their capability as deductive systems for refinement and reification and w.r.t. compositional and modular completeness, since these express the way we want to reason within such theories.

The combination of modular completeness and refinement and reification capabilities is especially important for a verification theory, because a theory satisfying both properties allows specification and programs (or implementations) to be treated on the same par. Thus, such a theory will fit a combined specification formalism (see section 4.2.1).

For such theories, too, the foundations were laid in [Zwi89], although reification was not treated. The latter, however, concept is investigated to some extent in [Lam85]. A pilot study has been performed in the context of the Esprit project 937 (Descartes).

6 Executable subsets of specification formalisms

If specifications could be directly executed, the problem of developing systems from specifications would be substantially simplified. Even if the direct execution is too inefficient to be used in a real system, it would still be very useful for checking and debugging specifications. However, it is unlikely that a specification language which is sufficiently general and expressive would be executable in any reasonable way. One possibility is both to try and identify subsets of the general language that are executable and to provide means of transforming a general specification into an executable one. An advantage of this approach is that there is no sharp boundary between specification and implementation and thus the development process can take place in a single framework. One can even conceivably mix executable and nonexecutable specifications.

Two directions of research on declarative specification formalisms which are executable have been developed by participants in the proposal. The first is the language Lustre which can be considered as an efficiently executable subset of Lucid. The second one concerns temporal logic.

6.1 Extensions of Lustre

Lustre is a declarative language for real time programming whose programs are intended to operate over timed sequences. It can be viewed as an executable subset of a temporal logic. A compiler for this language has been developed, which produces efficient code by synthesizing the program control as a finite automaton at compile time. However, Lustre is not yet embedded in some logic based formalism.

6.2 Executable temporal logic

Here, the research goal is a temporal formalism that can unify in a natural way the best features of existing, seemingly incompatible, formalisms and which will at the same time be capable of transforming itself into an executable form in a way similar to the logic programming languages. The approach is to define a strongly expressive temporal language with fixed point operators and meta-language features which will be suitable for our specification requirements. One should then focus on the two following problems:

1. Developing theorem proving and meta-language capabilities for this temporal language that will make it useful as a declarative language. The meta-language capability should make it possible to reason about exceptions to the rules of the language. This is important for handling reliability where failures are exceptions to rules.

2. Isolating the executable component of the language. For this, mathematical results can be used to rewrite the expressions of the language into an equivalent combination of sentences talking purely about the past and purely about the future. The future is to be read imperatively (make this future statement true) and should be made executable. A preliminary examination of this approach has shown that it can provide a unification of some of the already existing formalisms for handling concurrent and distributed systems. The unification comes out naturally and brings out the intrinsic advantages of each approach.

7 Tools and supporting environments

A number of participants in this proposal have been heavily involved in the construction of automatic tools for the verification of finite-state systems. The experience they gained has helped to convince many people that, for some applications such as the design of communication protocols and hardware devices, automatic verification is a workable approach. These automatic verification tools are based on one of two methods: using decision procedures or using model checking algorithms. However, decision procedures do not provide a viable approach for global verifications of non-trivially sized programs.

A different approach to verification that will also be pursued by participants in this proposal is the use of software tools to reason about programs written in the language CSP.

7.1 Verification using temporal logic model checkers

This approach starts by building a finite-state machine representing the program P. To verify the program, one then checks that all the sequences generated by this automaton satisfy (are models of) the temporal logic fomula. This *model checking* approach can be quite efficient, in fact, linear in the size of the model. This approach has been applied with success in a branching time logic framework, e.g. by Clarke et al. and by Sifakis et al.

7.2 Research directions in verification tools

7.2.1 Controlling state explosion in model checking systems

Although model checkers seemingly provide an efficient verification approach for finite-state systems, there are limitations in their current use. At present, model checking systems can handle state descriptions containing upwards of 100,000 states. Unfortunately, this is the number of states in the finite-state description of the global behaviour of the program and such a figure can be reached, in the worst case, by a system containing only 5 parallel processes, each of 10 states. The state explosion that occurs in parallelism is a very serious limitation of the automatic methods we have described. Hence the major research challenge in this area is the development of techniques for coping with such state explosion.

Inductive techniques Given a concurrent system composed of a network of (almost) identical processes (as is the case when modelling replicated hardware structures), do verification results which hold for some small fixed number of processes extend to hold in the general case? Unfortunately, this problem is, in general, unsolvable. However, Clarke et al. showed that, for specific instances of network topology and processes, one can establish certain properties of an arbitrary number of processes by verifying that the properties hold on a small fixed number of processes, which can be done effficiently using model checkers. We plan to develop these ideas and results, and determine useful network and process structures to which this type of technique is applicable.

Modular and compositional techniques A complementary approach is to apply modular and compositional techniques to temporal logic model checking. Currently the verification style of model checking is global, in the sense that a temporal property is verified for some given program by use of its global state transition graph. However, we believe that by careful decomposition and separation of concerns, the total verification effort can be considerably reduced.

In modular techniques, emphasis is placed upon reducing the size of the state machines by abstracting, where possible, complex states structures to a single state (or a much reduced network). One interesting, but limited, attempt in this direction has occurred in hierarchical verification of asynchronous circuits by Mishra and Clarke.

By compositional model checking, we mean verifying properties of complex networks via verification of properties of network components. Thus, large networks can be handled by a number of small model checking verifications. This complements the compositional reasoning in temporal logic specification, as in [BKP85] and others.

Other state machine minimisation techniques Still another approach is to reduce the size of the global state graph while generating it. For this, one can apply reduction methods which are based on equivalence relations on state graphs. Clearly, these relations must be compatible with the specification language. For example, if a logic is used to express specifications, equivalent states must satisfy exactly the same formulas.

7.2.2 Use of Decision Procedures

Decision procedures are not really usable for large program verifications. Nevertheless they can be helpful in a compositional framework and, as discussed in section 6, they play an important role in the direct interpretation of executable temporal logics.

7.2.3 First Order Temporal Reasoning

The methods we have described so far are only applicable to propositional temporal logics and finite-state programs. If one wants to go beyond this type of program and property, for example, to include general data types in specifications, one needs to use first-order temporal logic. There are no decision procedures for first-order temporal logic and hence one has to use proof systems. In this area, resolution based systems seem promising and are (as far as we are aware) the only ones for which proof editors have been built, e.g., [Cas86]. However, the rules provided in current work are rather low level. It is believed that the development of higher level resolution rules will provide more practicable automated theorem proving systems. Clearly, investigation of heuristics for the guidance of resolution proof must also occur.

An alternative is to translate the temporal formulas into first-order sentences interpreted over the time structure being used (e.g., the natural numbers). One can then use existing deduction systems for first-order theories.

A third possibility, promising but not so general, is to develop term-rewriting systems, currently used for reasoning about algebraically specified data-types, for handling temporal operators. The suggested approach

is apply Knuth-Bendix completion modulo temporal logic equations; it is hoped that this will lead to an automatic proof procedure. Once this is developed, the idea is to provide an automatic procedure to handle inductive proofs, which can not be encompassed by a proof system based on just equational reasoning.

8 Applications

One of the motivations of this action is to test the applicability of formal methods to the development of concurrent and distributed reactive systems. We intend to apply our methods to common paradigms taken from the domain of (real-time) communication protocols.

8.1 Real-time communication protocols

The development of communication protocols requires methods supporting:

1. The expression of real time constraints such as
   - minimal time between events, e.g. assumptions about the maximum rate of stimuli from the environment,
   - maximal time between events, e.g. the maximum response time of the system to an external stimulus from the environment,
   - cycle time, e.g. periodic behaviour of a group of stimuli with their responses.
2. The expression of nondelivery or corruption of a message.
3. The use of high level specification languages, especially of logic based ones, for the expression of service requirements. The formulation of such requirements can be done in terms of a few classes of non trivial properties such as order preservation of messages, properties characterising causality relations, properties expressing temporal constraints, atomicity (for broadcast protocols).
4. The use of formalisms allowing layered descriptions, abstraction and the definition of reification relations.

However, from the point of view of deriving correct implementations from a specification it is not sufficient to concentrate solely on function and the above stated temporal requirements. The possible implementations of a real-time system are usually restricted by the configuration and resources of the execution mechanism that will be used to run the system. Thus, in order to judge the feasibility of the implementation derived from the specification it is necessary to formalize the properties of the execution mechanism that will be used and compare these properties with assumptions made by the implementation. Hence, apart from the above high-level temporal requirements, paradigms of real-time systems also have to express more implementation-specific characteristics like

- the choice of architecture (multiprocessor/uniprocessor/sequential),
- the kind of scheduling policies used (fixed priority, dynamic priority, round robin, time slicing ...),
- the mechanism for the interaction with the environment (interrupts/polling).

To find the right level of abstraction for describing these implementation-specific characteristics is essential for deriving implementations from specifications and is a major research problem.

9 Organisation

9.1 Tracks – who does what with whom

The research to be undertaken can be quite naturally divided into 10 general subjects which we call *tracks*. Each of these tracks involves several of the partners in the project. Active collaboration will take place between partners participating in common tracks. The different tracks and the partners involved in each are listed below:

1. *Models for concurrency and timing*: Manchester & Imperial, Stockholm, Oxford, Eindhoven.
2. *Probabilistic models and reliability*: Manchester & Imperial, Crete, Stockholm, Oxford, Nijmegen.
3. *Mixed specification methods and languages*: Weizmann, Eindhoven, Nijmegen, Liège.
4. *Extending Temporal logic with data structures*: Manchester & Imperial, Eindhoven (observer).
5. *Timing Analysis*: Stockholm, Weizmann, Oxford, Grenoble, Nijmegen.
6. *Reification and Verification*: Manchester & Imperial, Stockholm, Weizmann, Oxford, Eindhoven, Grenoble (observer).
7. *Executable Temporal logic and Synthesis*: Manchester & Imperial, Weizmann, Grenoble.
8. *Model checking and data independence*: Manchester & Imperial, Crete, Stockholm, Weizmann, Grenoble, Liège.
9. *Tools*: Manchester & Imperial, Grenoble.
10. *Application*: Crete (observer), Oxford, Grenoble, Nijmegen (observer), Liège (observer), Eindhoven.

9.2 Circles – cross fertilization

To broaden the scope of collaboration and to foster cross fertilization, three *circles* will be created. Each circle is organized around a central theme. Each track is part of one (or more) circles. The circles are the following:

1. *Temporal logic.* Leaders: P. Wolper, H. Barringer.
2. *Timing and Reliability.* Leaders: J. Vytopil, C. Courcoubetis.
3. *Reification and Verification.* Leaders: B. Jonsson, W. deRoever.

The assignment of tracks to circles is given

Acknowledgement

The editor, Willem-P. de Roever, wishes to express his thanks to Pierre Wolper, Michiel Wijers, Kees Huizing, and Rob Gerth for collecting the sources of this document and combining them into a meaningful whole.

Tracks	TL	Timing & rel.	Reification & Ver.
Models for concurrency and timing		X	X
Probabilistic models and reliability		X	
Mixed specification methods and languages	X		
Extending temporal logic with data structures	X		
Timing analysis		X	
Reification and verification			X
Executable temporal logic and synthesis	X		
Model checking and data independence	X		
Tools	X		X
Application	X		X

Table 1: Assignment of tracks to circles

References

[BAMP81] M. Ben-Ari, Z. Manna, and A. Pnueli. The logic of nexttime. In *Eighth ACM Symposium on Principles of Programming Languages*, pages 164–176, Williamsburg, January 1981.

[BB86] B. Banieqbal and H. Barringer. A study of an Extended Temporal Language and a Temporal Fixed Point Calculus. Technical Report UMCS-86-10-2, University of Manchester, 1986.

[BKP85] H. Barringer, R. Kuiper, and A. Pnueli. A Compositional Temporal Approach to a CSP-like Language. In E.J. Neuhold and G. Chroust, editors, *Proceedings of the IFIP Working Conference "The Role of Abstract Models in Information Processing"*, pages 207–227, Vienna, 1985. North Holland.

[BKP86] H. Barringer, R. Kuiper, and A. Pnueli. A Really Abstract Concurrent Model and its Temporal Logic. In *Proceedings of the Thirteenth ACM Symposium on the Principles of Programming Languages*, pages 173–183, St. Petersberg Beach, Florida, January 1986.

[Cas86] Ross Casley. A proof editor for propositional temporal logic. Technical Report STAN-CS-86-1109, Stanford University, May 1986.

[GS86] S. Graf and J. Sifakis. A logic for the specification and proof of controllable terms of ccs. *Acta Informatica*, 23, 1986.

[Hoo87] J. Hooman. A compositional proof theory for real-time distributed message passing. In *PARLE 87, II*, pages 315–332, Berlin, 1987. Volume 259, LNCS, Springer-Verlag.

[KdR85] R. Koymans and W.P. de Roever. Examples of a real-time temporal logic specification. In *The Analysis of Concurrent Systems — Cambridge, September 1983, Proceedings*, pages 231–251, 1985, LNCS 207.

[Koy87] R. Koymans. Specifying message passing and real-time systems with real-time temporal logic. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 191–203, Vancouver, August 1987.

[Lam85] L. Lamport. An axiomatic semantics of concurrent programming languages. In K.R. Apt, editor, *Logics and Models of Concurrent Systems*, volume F13 of *NATO ASI*, pages 77–123. Springer Verlag, 1985.

[Pnu86] A. Pnueli. Specification and development of reactive systems. In *Proceedings of IFIP 86*, 1986.

[PW84] S.S. Pinter and P. Wolper. A temporal logic for reasoning about partially ordered computations. In *Proc. 3rd ACM Symposium on Principles of Distributed Computing*, pages 28–37, Vancouver, August 1984.

[RRSV87] J.L. Richier, C. Rodriguez, J. Sifakis, and J. Voiron. Xesar. Technical Report User's Manual, Laboratoire de Genie Informatique, IMAG, Grenoble, 1987.

[SZ85] Zhou Shengzong and Tang Zhisong. Designing and Verifying Distributed Programs Using Temporal Logic. Technical Report AS-IS-85, XYZ Report No. 4, Institute of Software, Academica Sinica, Beijing, China, December 1985.

[Wol83] P. Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56(1–2):72–99, 1983.

[Wol86] P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *Proc. 13th ACM Symp. on Principles of Programming*, pages 184–192, St. Petersburgh, January 1986.

[Zwi89] J. Zwiers. *Compositionality, Concurrency and Partial Correctness: Proof Theories for Networks of Processes, and their Connection.* Volume 321, LNCS, Springer-Verlag, 1989.

In this series appeared :

No.	Author(s)	Title
85/01	R.H. Mak	The formal specification and derivation of CMOS--circuits.
85/02	W.M.C.J. van Overveld	On arithmetic operations with M-out-of-N-codes.
85/03	W.J.M. Lemmens	Use of a computer for evaluation of flow films.
85/04	T. Verhoeff H.M.L.J.Schols	Delay insensitive directed trace structures satisfy the foam rubber wrapper postulate.
86/01	R. Koymans	Specifying message passing and real-time systems.
86/02	G.A. Bussing K.M. van Hee M. Voorhoeve	ELISA, A language for formal specification of information systems.
86/03	Rob Hoogerwoord	Some reflections on the implementation of trace structures.
86/04	G.J. Houben J. Paredaens K.M. van Hee	The partition of an information system in several systems.
86/05	J.L.G. Dietz K.M. van Hee	A framework for the conceptual modeling of discrete dynamic systems.
86/06	Tom Verhoeff	Nondeterminism and divergence created by concealment in CSP.
86/07	R. Gerth L. Shira	On proving communication closedness of distributed layers.
86/08	R. Koymans R.K. Shyamasundar W.P. de Roever R. Gerth S. Arun Kumar	Compositional semantics for real-time distributed computing (Inf.&Control 1987).
86/09	C. Huizing R. Gerth W.P. de Roever	Full abstraction of a real-time denotational semantics for an OCCAM-like language.
86/10	J. Hooman	A compositional proof theory for real-time distributed message passing.
86/11	W.P. de Roever	Questions to Robin Milner - A responder's commentary (IFIP86).
86/12	A. Boucher R. Gerth	A timed failures model for extended communicating processes.
86/13	R. Gerth W.P. de Roever	Proving monitors revisited: a first step towards verifying object oriented systems (Fund. Informatica IX-4).

86/14	R. Koymans	Specifying passing systems requires extending temporal logic.
87/01	R. Gerth	On the existence of sound and complete axiomati zations of the monitor concept.
87/02	Simon J. Klaver Chris F.M. Verberne	Federatieve Databases.
87/03	G.J. Houben J.Paredaens	A formal approach to distributed information systems.
87/04	T.Verhoeff	Delay-insensitive codes - An overview.
87/05	R.Kuiper	Enforcing non-determinism via linear time temporal logic specification.
87/06	R.Koymans	Temporele logica specificatie van message passing en real-time systemen (in Dutch).
87/07	R.Koymans	Specifying message passing and real-time systems with real-time temporal logic.
87/08	H.M.J.L. Schols	The maximum number of states after projection.
87/09	J. Kalisvaart L.R.A. Kessener W.J.M. Lemmens M.L.P. van Lierop F.J. Peters H.M.M. van de Wetering	Language extensions to study structures for raster graphics.
87/10	T.Verhoeff	Three families of maximally nondeterministic automata.
87/11	P.Lemmens	Eldorado ins and outs. Specifications of a data base management toolkit according to the functional model.
87/12	K.M. van Hee and A.Lapinski	OR and AI approaches to decision support systems.
87/13	J.C.S.P. van der Woude	Playing with patterns - searching for strings.
87/14	J. Hooman	A compositional proof system for an occam-like real-time language.
87/15	C. Huizing R. Gerth W.P. de Roever	A compositional semantics for statecharts.
87/16	H.M.M. ten Eikelder J.C.F. Wilmont	Normal forms for a class of formulas.
87/17	K.M. van Hee G.-J.Houben J.L.G. Dietz	Modelling of discrete dynamic systems framework and examples.

87/18	C.W.A.M. van Overveld	An integer algorithm for rendering curved surfaces.
87/19	A.J.Seebregts	Optimalisering van file allocatie in gedistribueerde database systemen.
87/20	G.J. Houben J. Paredaens	The R^2 -Algebra: An extension of an algebra for nested relations.
87/21	R. Gerth M. Codish Y. Lichtenstein E. Shapiro	Fully abstract denotational semantics for concurrent PROLOG.
88/01	T. Verhoeff	A Parallel Program That Generates the Möbius Sequence.
88/02	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specification for Information Systems.
88/03	T. Verhoeff	Settling a Question about Pythagorean Triples.
88/04	G.J. Houben J.Paredaens D.Tahon	The Nested Relational Algebra: A Tool to Handle Structured Information.
88/05	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable Specifications for Information Systems.
88/06	H.M.J.L. Schols	Notes on Delay-Insensitive Communication.
88/07	C. Huizing R. Gerth W.P. de Roever	Modelling Statecharts behaviour in a fully abstract way.
88/08	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	A Formal model for System Specification.
88/09	A.T.M. Aerts K.M. van Hee	A Tutorial for Data Modelling.
88/10	J.C. Ebergen	A Formal Approach to Designing Delay Insensitive Circuits.
88/11	G.J. Houben J.Paredaens	A graphical interface formalism: specifying nested relational databases.
88/12	A.E. Eiben	Abstract theory of planning.
88/13	A. Bijlsma	A unified approach to sequences, bags, and trees.

88/14	H.M.M. ten Eikelder R.H. Mak	Language theory of a lambda-calculus with recursive types.
88/15	R. Bos C. Hemerik	An introduction to the category theoretic solution of recursive domain equations.
88/16	C.Hemerik J.P.Katoen	Bottom-up tree acceptors.
88/17	K.M. van Hee G.J. Houben L.J. Somers M. Voorhoeve	Executable specifications for discrete event systems.
88/18	K.M. van Hee P.M.P. Rambags	Discrete event systems: concepts and basic results.
88/19	D.K. Hammer K.M. van Hee	Fasering en documentatie in software engineering.
88/20	K.M. van Hee L. Somers M.Voorhoeve	EXSPECT, the functional part.
89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a paralell program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.

89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.
89/13	A.T.M.Aerts K.M. van Hee M.W.H. Hesen	A program generator for simulated annealing problems.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.
90/1	W.P.de Roever-H.Barringer C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, pp. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.