Verifying Protocols with Executable Linear Logic*

P. Merino and J.M. Troya

Depto. Lenguajes y Ciencias de la Computacion
University of Malaga , 29071 Malaga , SPAIN

Abstract. This paper describes a new formalism for the specification and automatic verification of temporal properties where the counting and the distinction of the events are critical in deciding the correctness of systems. The specification formalism is based on giving an operational interpretation to a fragment of Linear Logic, which is specially suitable to express counting and changes between states. This operational semantics gives us a natural way to implement a model checking algorithm in a similar way to the automata-theoretic approach to verification.

1 Introduction

The purpose of verification of concurrent systems is to reduce the risk of serious design errors produced by multiple processes that interact simultaneously with one another. One of the most successful approaches to verification is to automatically compare a *specification* of the system requirements against the *model* of a proposed design for satisfying them, both described with formal languages (the *dual language* approach). If the comparison consists in deciding whether the model of a system meets its Temporal Logic specification the verification method is called *model checking* [1]. The tools in this framework usually employ *Linear-time Propositional Temporal Logic* (LPTL) [2] for the requirements, while a wide range of formalisms can be used to describe the model of the protocol [3].

The problem we are interested in is specification and verification of properties in protocols for distributed systems, that are becoming more prone to errors due to their reconfiguration capability. This characteristic means that the system potentially offers a flexible environment for modification and extension. In these kinds of systems, it is usual to have processes changing their communications channels, and creating new processes that can leave the system at any time, depending on the user's requirements. In general, these kinds of changes are not predicted at the time that the model and the properties are defined. This is the case with the new communication protocols designed to support multimedia and multipeer applications [4-7] or to support mobile computing [8]. We employ the terms system, distributed system or protocol throughout the paper to refer to protocols for distributed systems. The term trace is employed for the observable behaviour of a distributed system.

It is well known that LPTL is unable to specify some classes of properties, such as counting or distinguishing several occurrences of the same event [2]. Counting is important in the case of requirements about scheduling policies, some quality of service parameters (QoS) or protocols with sequence numbers. Distinguishing events is necessary to deal with reconfigurable systems, with an arbitrary number of similar processes. There are several proposals for extending this logic to solve these problems [9-14]. This paper describes how to employ a different logic, *Linear Logic* [15], for the specification and model checking of distributed systems, mainly for the properties that LPTL is unable to express.

Linear Logic (LL) provides a natural way to express counting and changes between states, which makes it suitable to specify the requirements where the counting of events is critical in deciding the correctness of the system. There are also many computational interpretation of the proofs in LL that can be used to reason about the specification itself [16, 17]. The approach in this paper consists in

* Extended Abstracts of LOPSTR'98, Eighth International Workshop on Logic- based Program Synthesis and Transformation, 15-19 June 1998, Manchester, UK. Technical Report Series, Department of Computer Science, University of Manchester, ISSN 1361-6161. Report number UMCS-98-6-1. http://www.cs.man.ac.uk/cstechrep/titles98.html

using LL as the basis to define a new specification formalism called *Linear Observers*[1], which is obtained by giving an operational interpretation to some connectives of LL. Model checking with Linear Observers consists in employing the operational meaning of the connectives in a similar way to the automata-theoretic approach to model checking [18]. From the operational point of view, the observer is a transition system that takes a trace of a protocol as its input alphabet, and tries to reach an accepting state from a given initial state. The main feature of the observer compared with the standard automata is the use of multisets of propositions as states, instead of single atoms. The soundness existing between the logic basis and the operational interpretation opens the possibility of using proof based transformations of the specification in order to optimize the model checking process.

Although there are well known logic programming languages based on LL such as Lolli [19] or Linear Objects [20], we propose a new formalism instead of extending these languages. The reason is that we intend to design a simple language clearly suitable for the specification and efficient model checking of properties related to a formal model of the protocol, following the *dual language* approach to verification. This objective is not compatible with preserving the semantics of the languages mentioned above. The same reason eliminates some formalisms based on multisets, like Petri nets or GAMMA[21], as an alternative to Linear Observers.

This paper provides the main ideas on the use of *Linear Observers*. Section 2 describes how to define this formalism from a subset of Linear Logic. The syntax and semantics for model checking is described in section 3. Section 4 contains comparisons to related work. The last section contains conclusions and future work.

2 From Linear Logic to *Linear Observers*

Linear logic (LL) was proposed by Girard as an alternative to Classical Logic to represent the notion of change. In this new logic there is a replacement of propositions in classical logic by linear propositions, which can be produced and consumed according to the connectives of the logic. Since the whole set of connectives can make its application conceptually and computationally intractable, it is usual to choose only a few of them for particular application domains. In this section, we first explain the role of the tensor fragment of LL to construct *Linear Observers*. Then an overview of how to append new LL connectives to construct more complex observers is given.

2.1 The tensor fragment of Linear Logic. The tensor fragment of LL is composed by the tensor connective "⊗" to construct formulas and the four rules in figure 1 to construct proofs. A *formula* in this fragment is either a propositional atom or the tensor product A ⊗ B, A and B being formulas. The informal meaning of A ⊗ B is that both A and B are available "at the same time". It is usual to abbreviate a formula A ⊗ A ⊗ ... ⊗ A as A^n. It is also usual to think of the tensor formula as a *multiset* of propositional atoms, the number of occurrences of the same atoms being its *multiplicity*.

Structural Rules *Logical Rules*

$$\frac{}{A \vdash A}\ (\text{Identity}) \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\ (\text{Cut}) \qquad \frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\ (\otimes_L) \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\ (\otimes_R)$$

Figure 1. Tensor fragment of LL

2.2 Linear Observers and the tensor fragment of LL. *Linear Observers* is a formalism based on a computational interpretation of non cut-free proofs in the tensor fragment of LL. The relation between the observer behaviour and the LL proof is as follows:

[1] The term *Linear Observers* denotes the specification language. A particular property expressed with this formalism is called a linear observer.

- Running the observer over a particular trace is represented as a theory in LL. This theory is composed of two groups of axioms, one representing the observer transitions, and other the events in the trace.
- The conclusion of the proof Γ |— A represents the initial (Γ) and accepting (A) states in the observer.
- The possibility of finding a proof for Γ |— A in the theory means that the observer can recognize the trace.

The proof in figure 2 is employed to explain the description above. This proof represents the verification of the following property:

> " Given a system with processes *receiver* (R) and *sender* (S), they send messages *ack* and *data* alternatively, and starting with receiver it is possible to reach a state where the sender has got three *ack* messages".

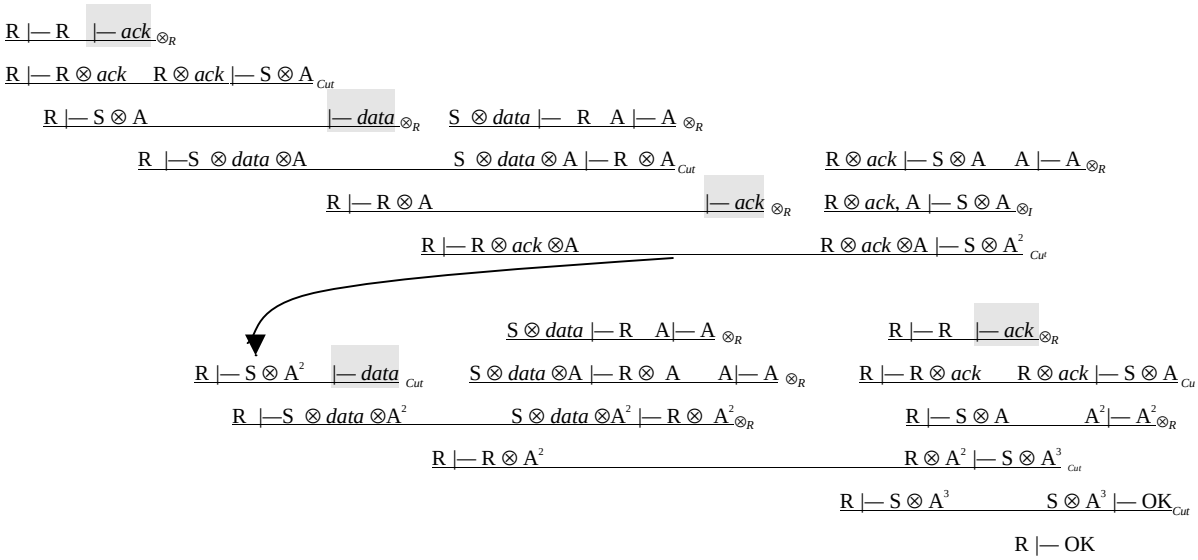

Figure 2. LL proof for the sequent R |— OK

The axioms corresponding to the observer transitions are the set

{ R ⊗ *ack* |— S ⊗ A S ⊗ *data* |— R S ⊗ A³ |— OK}

The first one specifies that after sending the *ack* message the turn passes to the sender process (S), but also the occurrence of *ack* is recorded with the linear atom A. The second one specifies that the turn returns to receiver (R) after sending *data*. The third one specifies that the proposition OK is available if it is the sending turn and three *ack* messages were received. The atoms R, S , A and OK will be called *facts* in the observer, because they represent the history of the trace. Note that these axioms are employed (reused) in the proof as many times as necessary (we assume an unlimited number of copies). The axioms corresponding to the events are the set

{ |— *ack* |— *data* |— *ack* |— *data* |— *ack* }

This set represents a particular trace to be verified, and they are supposed to be non reusable. These axioms are consumed sequentially in the proof (shaded in the figure), according to the interpretation of the proof proposed by Gehlot [22]. The order of consuming is shown in the proof by reading it left to right and top down inside every branch. The conclusion R |— OK represents that the initial observer state is R and the desired reachable condition is OK.

2.3 Other LL features. Using other LL connectives can enrich the expressive power of the observers. One extension is to give the observer the adaptation capability to activate new transitions as a response to the reconfigurable nature of many distributed systems. This adaptation implies that

the axioms for some transitions cannot be employed until a particular one is used in the proof. In the final definition of *Linear Observers*, there is also a special guard associated to the observer transitions which imposes a set of *additional constraints* over the multiplicities of some facts. In a linear logic interpretation these constraints mainly corresponds to use the linear negation ($^{\perp}$).

3 The specification formalism

This section describes the syntax and the operational semantics of *Linear Observers*.

3.1 Syntax. Instead of using pure LL connectives, a linear observer can be expressed in a more friendly textual or graphical form. The *textual representation* is constructed with the following set of elements. The *facts* are encoded as strings starting with capital letters, "0" and "1" being predefined. The *multiplicities* are natural numbers. The *additional constraints* on facts are expressions using relational and boolean symbols and the names of the facts to represent their multiplicities. The *guards* over the trace are boolean expressions with the predicates defined in the actual modelling language. The symbols "**:**", "**->**" and "**.**" Are employed as *separators*.

The observer is encoded with the following constructions:

(1) $m_1\ A_1$**,** $m_2\ A_2$**,** ...**,** $m_p\ A_p$ **:** E **:** G **->** $n_1\ B_1$**,** $n_2\ B_2$**,** ...**,** $n_q\ B_q$**.** *(p, q≥ 1)*

encodes a *static transition* ; $m_1\ A_1$, $m_2\ A_2$, ..., $m_p\ A_p$ being the *precondition*; $n_1\ B_1$, $n_2\ B_2$, ..., $n_q\ B_q$ being the *postcondition*; m_i and n_i being the mutiplicities; E being a expression defining the *additional constraint on facts*; and G being a classical logic proposition with no free variables.

(2) Prec **:** E**:** G **->** Post**;** $Prec_1$**:**E_1**:** G_1 **->** $Post_1$**;** .... **;** $Prec_r$**:**E_r**:** G_r **->** $Post_r$**.** *(r ≥ 1)*

encodes a *adaptative transition* ; every $Prec_i$, E_i, and $Post_i$ being a *precondition*, additional constraint, and *postcondition*, respectively; and every G being a classical logic predicate with free variables satisfying that any variable in G_i it is also in G.

(3) $m_1\ A_1$**,** $m_2\ A_2$**,** ...**,** $m_p\ A_p$**.** *(p≥ 1)* encodes a multiset of *initial multiset* of facts

(4) **->** $m_1\ A_1$**,** $m_2\ A_2$**,** ..., $m_p\ A_p$**.** *(p≥ 1)* encodes a *accepting condition*

A *correct textual specification* of a linear observer is a sequence of the previous constructions, containing at least one clause for an *Accepting Condition*, the order of the constructions being irrelevant. Given the proof discussed in section 2, we can define the code below for the linear observer. The guards `send(ack)`and `send(data)` are related to the model of the protocol being verified (in this example there is no additional constraint over the facts).

```
R.
R :: send(ack)  -> S, A.
S :: send(data) -> R.
S, 3 A :: true -> OK.
->OK
```

3.2 Semantics. The problem we are considering is to define a notion of satisfaction in order to decide whether a trace of a distributed system is a model of a linear observer. We now define both elements to be compared and the notion of satisfaction.

The trace. We assume a particular modelling language in each implementation of the model checking method, so the system states and the events to be analysed depend on the modelling language.

Definition 1. A *trace* is a sequence $(s_0, s_1, s_2, \ldots)$ such that: s_i is a global state and, for i > 0, s_i is produced by a transition t_i in s_{i-1}; t_i is stored as a part of s_i; s_0 is the initial global state for verification, with $t_0 = null$. Given a trace α, the functions $head(\alpha)$ and $tail(\alpha)$ give the first state and the sub-trace starting in the second state of α, respectively.

Definition 2. A *state formula* is an assertion about individual states in a trace, which are constructed with a *set of predicates* Ψ evaluable over every global state s_i. Predicates in Ψ can contain free variables, and $sf(\Psi)$ represents the set of state formulas from Ψ.

Definition 3. The *evaluation* (truth assignment) of a state formula f without free variables in s_i, is given by the function $eval(f, s_i)$. The *evaluation* of a state formula φ with free variables in the state g_i is a pair $<B, S>$ obtained with the function *evalfree()* defined as

$evalfree(\varphi, w) = <true, \{ V_1/c_1, ..., V_n/c_n \} > iff \exists c_1,...,c_n \mid eval(cl(\varphi, \{ V_1/c_1,.., V_n/c_n \}), w) = true$
$evalfree(\varphi, w) = <false, \{ \varepsilon \} > otherwise$

where B is the truth assignment; S is a substitution; and $cl(\varphi, \{ V_1/c_1,.., V_n/c_n \})$ represents the closure of the formula φ by replacing every V_i with c_i.

The transition system. A linear observer is defined to recognise a trace, so its transition system must consider the predicates evaluable in the trace.

Definition 4. The *transition system* associated with a linear observer is a *tuple* $S = < A, \Psi, T, E, G, I, Pre, Post, H_0, F >$, where A is a finite set of elements called *facts*; Ψ is the set of predicates to construct state formulas; T is a finite set of elements called *Transitions*, which contains a special element called *internal transition* denoted by i; $Pre: T \to \mu A$, and $Post: T \to \mu A$ are *precondition* and *postcondition* functions, which give a multiset of facts to each transition; $E: T \to bexpr(A)$ is the *additional constraint* function, which gives for each transition a boolean expression over multiplicities in multisets from A; $G: T \to sf(\Psi)$ is the *guard* function, which gives a closed state formula for each transition; $I: T \times sf(\Psi) \to T$ is the *instantiation* function, and produces a new set of transitions for a given transition, depending on the current state of the trace; $H_0 \in \mu A$ is a multiset of facts representing the observer's *initial multiset* of Facts (the initial history); F is a finite set of *accepting conditions*, each one is defined as a multiset $B \in \mu A$. There is a set of transitions where this function is not defined called *static transitions* T_s.

Definition 5. The *state* of a transition system (associated with a linear observer) in respect to a trace is a tuple $<H, T^a, \alpha>$ where, $H \in \mu A$; T^a is the set of *active transitions*, and contains elements (t, g) with $t \in T$ and $g \in sf(\Psi)$; and α is a trace. $T^a{}_0$ is the set of initial active transitions, and contains pairs (t_i, g_i) satisfying $t_i \in T_s \wedge g_i = G(t_i)$

Definition 6. (*Transition enabling*). One transition $t \in T$ with a guard containing a set of free variables $\{ V_1, V_2, ..., V_n \}$ $(n >= 0)$ is *enabled* in the state $<H, T^a, \alpha>$, iff $(Pre(t) \leq H) \wedge E(t) \wedge \exists c_1, c_2, ..., c_n \mid evalfree(G(t), head(\alpha)) = <true, \{ V_1/c_1, V_2/c_2, ..., V_n/c_n \} >$

Definition 7. (*Transition rules*). Given the state $<H, T^a, \alpha>$, one step in the observer with respect to the trace produces the new state $<H', T^a', \alpha'>$, and the transition is denoted $<H, T^a, \alpha> \to^t <H', T^a', \alpha'>$. The step is defined with the following rules:

Rule 1: *Evolution without instantiation*

$$\frac{\exists (t, g) \in T^a,\ t \neq i, t \in T_s) \wedge enabled(t, <H, T^a, \alpha>)}{<H, T^a, \alpha> \to^t < H - Pre(t) + Post(t), T^a, tail(\alpha)>}$$

Rule 2: *Observer evolution with instantiation*

$$\frac{\exists (t, g) \in T^a, t \neq i, \neg (t \in T_s) \wedge enabled(t, <H, \alpha>)}{<H, T^a, \alpha> \to^t < M - Pre(t) + Post(t), T^a + I(t, g, head(\alpha)), tail(\alpha)>}$$

Rule 3: *Evolution with the internal transition*

$$\frac{\forall (t, g) \in T^a, t \neq i \neg (enabled((t, g), <H, T^a, \alpha>))}{<H, T^a, \alpha> \to^i <H, T^a, tail(\alpha)>}$$

The internal transition is only employed when no other transition is enabled. The operations "-" and "+" for multisets are defined with no effect over the predefined facts "0" and "1".

The notion of satisfaction. Finally, we define whether the trace is a model of the observer.

Definition 8. A linear observer with a transition system S = < *A, Ψ, T, I, Pre, Post, E, G, H_0, F* > accepts a trace α if and only if it is possible to construct a sequence of applications of the transition rules starting in the state <H_0, $T^a{}_0$, α> that reach a state <H_f, T^a, α′> satisfying $H_f \leq F$.

3.3 Model checking. The verification problem we are considering is the following. Given a system with a designated global state s_0, and a linear observer *O*, with initial state o_0, check if any finite trace starting in s_0 satisfies *O*. Our problem is to produce and check all the traces of the system using the rules above (1-3). This work can be carried out with a depth first search of the state space produced by the evolution of the system and the observer. The algorithm must employ the composite global state <*s, o*>, where *s* is the system global state and *o* is the observer state. The algorithm must consider the removing of duplicate states to prevent cycles, and it can be implemented by extending current verifiers. The method proposed in this paper has been implemented by extending the verifier SPIN [23], and current results confirm that this is a practical and efficient approach to model checking.

The correctness of our approach is established by the following results.

Proposition 1 (termination). The verification of a finite state distributed system against a deterministic linear observer with a depth first search method can be performed in a finite number of steps.

Proof: Linear observers are not finite-state automata like the ones usually employed in automatic verification. Actually they could have an arbitrary number of states, which in practice depends on the trace we are trying to analyse. But, as the trace is finite, due to the removing of loops, the algorithm always terminates. ♦

Proposition 2 (soundness) Given a finite trace of length N, and a linear observer with only static transitions O with initial state o_0, the model checking algorithm reports the recognition of the trace by the observer for a given accepting condition *ac* if and only if there exists a linear logic proof for the sequent Γ |— A, where Γ and A represent the initial (o_0) and the accepting (*ac*) states in the observer. The theory for the proof is composed of axioms representing the observer and the trace.

Overview of the proof: A similar proof was developed by Gehlot in the case of a theory with no axioms for the trace [22]. In order to transform our problem into a Gehlot problem, we can employ the axioms $g_0 \vdash g_1$ $g_1 \vdash g_2$ ... $g_{n-1} \vdash g_n$ in order to represent the trace, every g_i being an event. Then, we can transform our conclusion into the sequent Γ, g_0|— A in the theory defined by the observer plus the new axioms. ♦

Work is in progress to extend proposition 2 to linear observers with adaptative transitions**.**

3.4 Example. The following small example shows the two main characteristics of Linear Observers, counting and adaptation. The system to be verified is a bounded channel with one input α and one output β. The properties required for this channel are non-losing and non-duplication of messages. These properties seem to be specified with the LPTL formula send(α, M) → send(β, M), where M is an auxiliary variable. But this specification fails for these properties in the case of duplicate inputs (for example, with input: *a, a, a, b* and output *a, b* the verdict for the formula is true). The following linear observer represents the violation of these properties:

```
1: send(α, M) -> D ;  D :: send(β, M) -> 0.
1: D==0:  send(β, M) -> Error.
-> Error.
```

The observer employs the fact D to count the number of messages in the channel, and it employs the variable M to dynamically activate one transition to check if every input message becomes an output message. It also has a transition to produce the fact Error in case of an undesired output. The additional constraint (D==0)is employed as a special precondition for that transition. The predefined atoms "1" and "0" are employed as precondition and postcondition, respectively. The properties are satisfied if the verification of all the traces ends without reaching the accepting condition Error.

4 Comparisons with related work

Comparison with Temporal Logic. Temporal Logic is extended with new elements to verify temporal properties with some kind of counting or adaptation. In ETL [9] and TL^n [10] new temporal operators can be defined for counting. Koymans proposed augmenting temporal logic with new axioms to distinguish between different occurrences of the same event [11]. In [12], the problem of counting is solved using history variables. Both problems can be studied using first-order temporal logic instead of the propositional version [2]. Some kind of reconfiguration is considered in [13] and [14] towards systems with many similar processes; both papers explain how to use indexes and quantifiers (the first one is based on a branching-time logic). Although some of these approaches, specially the use of quantifiers, make the specification more unreadable and very prone to errors, they can specify most of the desired temporal properties. We cannot deal with all kinds of temporal properties, and in particular Linear Observers can not express properties about infinite traces. In contrast, the advantage of our proposal is the simplicity of the specifications to analyse a particular set of potentially critical errors.

Comparison with other works on Linear Logic. The closest work to this paper is the use of the tensor fragment of LL to construct proofs for firing sequences in Petri nets [22]. Our main contribution is to consider the notion of the trace as an external element to the net, and the operational interpretation of the whole system (net + trace). In [24], the authors give a model theoretic interpretation to LL connectives, which is suitable for specifying complex reachability conditions on Petri nets, and which could be the basis for our work. But they employ a non-trivial combination of connectives, which could introduce doubts about the correctness of the specification itself.

Comparison with automata for verification. There are several kinds of automata to specify and verify temporal properties of distributed systems with infinite traces [11]. These classes of automata are clearly superior to Linear Observers in order to specify general liveness properties. But our proposal is more suitable for properties on finite traces, because the complex cycle detection algorithm required by the others it is not necessary. Moreover, our formalism possesses the advantage of its logic base.

5 Conclusions

Linear Observers is a suitable formalism to specify temporal properties where counting events is critical. The class of properties that we can realistically verify are variants of the bounded existence property defined in [25] (a given state/event must occur *k* times within a scope). The encoding of complex facts in the history of the execution as multisets allows us to define powerful variants of this bounded existence pattern.

In the paper, we provide evidence that it is possible to employ model checking for the verification of distributed systems against the properties defined with the observers. The way that we define the formalism by giving an operational interpretation to linear logic connectives opens the possibility of specifying more complex properties within the model checking framework. The formalism has been implemented, and is being employed for the verification of realistic protocols.

References

1. Clarke, E. M., Emerson, E. A., and Sistla, A. P., *Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications,* ACM Trans. on Programming Languages and Systems, 8 (2), 244-263 (1986).

2. Manna, Z., and Pnueli, A., *The Temporal Logic of Reactive and Concurrent Systems* - Specification - , Springer-Verlag, New York, 1992. p. 427.

3. Clarke, E. M., Wing, J. M., *Formal Methods: State of the Art and Future Directions*, ACM Computing Surveys, ACM 50TH Anniversary Issue Workshop on Strategic Directions in Computing Research, 28(4):626-643 (1996).

4. Guilmet, M., Thomas P., Traverson, B., *Design, Implementation and Validation of a Multi-Peer Protocol using Estelle*. Formal Description Techniques, III, North-Holland, 487-5502 (1991).

5. Fredlund, L., Orawa F., *Modelling Dynamic Communication Structures in LOTOS*, Formal Description Techniques, IV, North-Holland, 185-200 (1992).

6. Najm, E., *Dynamic Configuration in LOTOS*, Formal Description Techniques, IV, North-Holland, 201-216 (1992).

7. Merino, P., Troya, J.M., *Modelling and Verification of the MCS Layer with SPIN*, The Spin Verification System, Dimacs Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 32:101-110, 1997.

8. Tompson, J.B., (Ed.) *Special Issue Mobile Computing*, Computer Communications 19(4), 1996

9. Wolper, P., *Temporal Logic can be more Expressive*, Information and Control, 56:72-99, 1983..

10. Kurshan, R. P., Pinter, S. P., *Extensions of Temporal Logic for Counting with Applications to Model Verification*, Formal Description Techniques, IX, North-Holland, 373-386 (1990).

11. Koymans, R., *Specifying Message Passing Systems Requires Extending Temporal Logic*, in Temporal Logic in Specification, LNCS 398, 213-223, Springer-Verlag, 1987.

12. Hailpern, B.T., *Verifying Concurrent Programs Using Temporal Logic*, LNCS 129, Springer-Verlag. 1992.

13. Clarke, E. M., Grumberg, O., *The Model checking Problem for Concurrent Systems with Many Similar Processes*, in Proc. of the Computer Aided Verification, LNCS 575, 189-201, Springer-Verlag, 1991.

14. German, S. M., Sistla, A. P., *Reasoning About Systems with Many Processes*, Journal of ACM, 39 (3):267-735 (1992).

15. Girard, J. Y., *Linear Logic*, Theoretical Computer Science, 50 (1987).

16. Abramsky, S., *Computational Interpretations of Linear Logic*, Theoretical Computer Science, 111:3-57.

17. Alexiev, V., *Applications of Linear Logic to Computation: An Overview*, TR93-18, Dept. of Computing Science, University of Alberta, USA, 1993

18. Vardi, M. Y., and Wolper, P., *An Automata-Theoretic Approach to Automatic Program Verification* in, Proc. of the Symp. on Logic in Computer Science, Cambridge, 1986.

19. Hodas, J.S., Miller, D., *Logic Programming in a Fragment of Intuitionistic Linear Logic*. Information and Computation, 110(2):327-365.

20. Andreoli, J.M, Pareschi R., *Linear Objects: Logical processes with built-in inheritance*. New Generation Computing, 9(3-4):445-473.

21. Banatre, J.P, Metayer D., *Programming by Multiset Transformation*. Communications of the ACM, 36(1):98-111.

22. Gehlot, V., *A Proof-Theoretic Approach to Semantics of Concurrency*, Phd. Disertation, University of Pennsilvania, USA, 1992.

23. Holzmann, G., *Design and Validation of Computer Protocols*, Prentice-Hall, New Jersey, 1991.

24. Engberg, U., Winskel, G., *Linear Logic on Petri Nets*, Tech. Report BRICS RS-94-3 Aarhus University,Denmark, 1994.

25. Dwyer M.B., Avrunin S. A., Corbett J,C, Property Specification Patterns for Finite-state Verification *2nd Workshop on Formal Methods in Software Practice*, 1998.