Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

458

J.C.M. Baeten J.W. Klop (Eds.)

CONCUR '90

Theories of Concurrency: Unification and Extension

Amsterdam, The Netherlands, August 27–30, 1990
Proceedings

Springer-Verlag
Berlin Heidelberg New York London
Paris Tokyo Hong Kong Barcelona

Editors
J. C. M. Baeten
Department of Software Technology, CWI
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
and
Programming Research Group, University of Amsterdam
Kruislaan 409, 1098 SJ Amsterdam, The Netherlands

J. W. Klop
Department of Software Technology, CWI
Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
and
Department of Mathematics and Computer Science, Free University
De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands

CONCUR'90

CR Subject Classification (1987): F.1.2, D.1.3, D.3.1, D.3.3, F.3.1

ISBN 3-540-53048-7 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-53048-7 Springer-Verlag New York Berlin Heidelberg

Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
2145/3140-543210 – Printed on acid-free paper

Preface

The ESPRIT Basic Research Action 3006, CONCUR (Theories of Concurrency: Unification and Extension) started on September 1, 1989. The principal aims of the action are to explore the relationships among the different approaches to algebraic concurrency theory, and to develop a formalism applicable to a wide range of case studies. Verification of concurrent or distributed systems has up till now been undertaken only on a very small scale, in a haphazard way, and with a multitude of techniques and formal theories. For industrial applicability, it is essential that some unity emerge from the competing theories of concurrency, and that the verification process be supported by reliable software tools.
Coordinating partner of CONCUR is CWI (Centre for Mathematics and Computer Science) in Amsterdam, the other partners are the Universities of Edinburgh, Oxford, Sussex and Amsterdam, INRIA in Sophia Antipolis, and the Swedish Institute for Computer Science.
The conference CONCUR'90, hosted by CWI with the help of the University of Amsterdam, marks the end of the first year of CONCUR. In response to the call for papers, 54 papers were submitted. Of these, 31 were selected for presentation by the program committee. The selected papers appear in these proceedings, together with two articles by invited speakers. Unfortunately, due to lack of time, the other three invited speakers are only represented by abstracts.
The editors want to thank the members of the program committee with all their subreferees, and members of the organizing committee for all their efforts.

Amsterdam, July 1990

List of referees

S. Arnborg
J.C.M. Baeten
F. Baiardi
M. Bellia
J.A. Bergstra
G. Berry
E. Best
A. Bouali
G. Boudol
F. Boussinot
E. Brinksma
B. Buchberger
I. Castellani
Zhou Chao Chen
R. Cleaveland
J. Davies
R. De Nicola
R. De Simone
R. Devillers
A. Eliëns
J. Esparza
A. Fantechi
E. Fehr
G.L. Ferrari
A. Galligo
P. Gardiner
H.J. Genrich
R.J. van Glabbeek
S. Gnesi
J. Goguen
U. Goltz
R. Gorrieri
J.F. Groote
H. Hansson
C.A.R. Hoare
J. Hollman
P. Inverardi
D. Jackson
He Jifeng
B. Jonsson
M. Josephs
J.W. Klop
A.S. Klusener
E. Kranakis
K.G. Larsen
B. Lisper
H.H. Løvengreen
W.F. McColl
Q. Miller
R. Milner
B. Monien
U. Montanari
O. de Moor
P. Mussi
M. Nielsen
E.-R. Olderog
F. Orava
J. Parrow
A. Ponse
L. Priese
A. Ravn
J.J.M.M. Rutten
J.B. Saint
A. Maggiolo Schettini
S. Schneider
G. Sjödin
A. Skou
E. Smith
B. Steffen
E.P. de Vink
F.W. Vaandrager
L. Wallen
G. Winskel
J. Woodcock

Table of Contents

Partial Order Semantics of Concurrent Programs

Eike Best
Institut für Informatik, Universität Hildesheim, D-3200 Hildesheim

(Abstract of an Invited Lecture at CONCUR'90)

So–called arbitrary interleaving has traditionally been a popular method for describing the possible behaviours of a concurrent program (viz., its operational semantics). This method has been very successful, both because it is generally easy to formalise and because many (one might argue: all) interesting properties of a concurrent program can be expressed and analysed using interleaved sequences. Partially ordered sets have been suggested as an alternative description method. The concurrency that may be present in the behaviour of a concurrent program can be expressed in the associated partial order by means of the absence of any ordering. In addition and relating to these two approaches, a wide spectrum of intermediate notions, variations and extensions have also been investigated. There have occasionally been controversial discussions relating to the different possible ways of defining the operational semantics of concurrent programs. Sometimes, the interleaving approach has been rejected out of hand because it does not represent concurrency. At other times, the partial order approach has been rejected because it seemed technically too cumbersome to formalise.

I would like to argue in this talk that both interleaving semantics and partial order semantics (and any other appropriate semantics) can – and should, whenever appropriate – be defined peacefully side by side. In this view, the important problems become the following ones:

(i) What is the nicest way of formalising any of the desired semantics?

(ii) What is the precise relationship between the two (or more) semantics?

(iii) Which is the most appropriate semantics to analyse a given property, or class of properties?

A well–known instance of problem (iii) is the notion of fairness: while interleaving semantics suffices to express various interesting fairness notions, it turns out that some of them can be defined and analysed more lucidly in the partial order framework[1].

Recent investigations have produced ample knowledge about the questions (i)–(iii) claimed to be important above in the framework of fundamental models of concurrent systems, such as Bergstra/Klop's ACP, Hoare's CSP, Milner's CCS, Petri's Nets and others. These basic models are immediately useful to describe the standard flow of control of concurrent programs. As a rule, they need to be extended for the purpose of describing other important features of concurrent programs, such as the propagation of data values and non–standard control flow constraints, as might be induced by a priority operator or by the use of timers; in the sequel, these other features will be called 'non–basic'. In existing concurrent programming languages, non–basic features tend to play a prominent rôle; in particular, the correctness of programs written in languages with such features will in general depend crucially on their proper use.

This talk explores the partial order semantics of concurrent programs not only in the presence of standard flow of control, but also taking into account data structures, a priority construct and timing aspects. The following particular instances of the more general questions identified above will be discussed:

(i) What is a satisfactory way of defining a partial order semantics, paying particular attention to the non basic features?

(ii) What is the relation to interleaving semantics, again with particular regard to the non–basic features?

(iii) What class of properties is likely to be amenable to analysis in terms of partial orders?

The discussion will be motivated and guided by an investigation underway within the Esprit Basic Research Action No.3148 DEMON (Design Methods Based on Nets). This study has two aims: (a) to give a partial order semantics to occam–2 using the Petri net model[2], and (b) to design a programming notation with a Petri net semantics.

[1] The transcription of this talk will contain pointers to the literature.

[2] occam–2 has all of the basic and non–basic features mentioned above.

SCONE : A SIMPLE CALCULUS OF NETS

Roberto Gorrieri Ugo Montanari
Dipartimento di Informatica — Università di Pisa
Corso Italia 40 — I - 56100, Pisa, Italy

ABSTRACT

A simple calculus of Place/Transition Petri Nets, called SCONE, is introduced. Relationships between SCONE and the subset of CCS without restriction and relabelling, called RCCS, are studied by showing that RCCS can be implemented onto the net calculus. The implementation is given by means of a suitable mapping from RCCS transitions to SCONE computations, resulting in a finite net representation for RCCS agents. By quotienting the transition system of RCCS with respect to the implementation mapping, we induce also a "true concurrent" semantics for RCCS. These results are developed in the framework of "graphs with algebraic structure" as explained in [MM88, DMM89, MY89, F90, FM90, Co90].

1. INTRODUCTION

Among the various approaches to the semantics of concurrency, we distinguish two: the so-called "interleaving" approach and the "true concurrent" one. The main merit of the former is its well-established theory. A concurrent system is described by a term of a language, which gives rise to a transition system. The states are themselves terms of the language and the transitions are defined by means of a deductive system in structural inductive form, as proposed by Plotkin [Plo81] with his *Structured Operational Semantics* (SOS for short). Equivalences among states / terms are defined according to a suitable notion of observation and the useful result is that observational congruences have a nice axiomatization. Unfortunately, there is a serious drawback: this approach relies on the well-known idea of describing system behaviours as sequences of transitions, a too simplistic view in many practical cases when information about distribution in space, about causal dependency or about fairness must be provided. On the other side, in the "true concurrent" approach, which started from the pioneering work of Petri [Pet62], this kind of information can be easily given, but net theory has not yet reached a completely satisfactory theoretical treatment if compared with the firm results coming from the interleaving side. Rephrasing and extending the ideas developed for the interleaving approach to the "true concurrent" case can be considered the main goal of a branch of concurrency theory. The present paper aims at giving a contribution in this direction.

The basic model of Place/Transition Petri Nets has recently received, by the second author in joint work with J. Meseguer [MM88], a simple algebraic description by showing that a P/T net can be *statically* described as an ordinary directed graph equipped with a commutative monoidal operation $\oplus$ on nodes, and *dynamically* as a graph with also two operations on transitions (the parallel composition operator $\otimes$ and the

Research supported in part by EEC Basic Research Action n.3011 CEDISYS.

sequential composition operator ;), together with suitable axioms for identifying those computations which are observationally identical. Unfortunately, Petri Nets, at least in their usual formulation, are not very suited for modular description of concurrent system: to get this capability we should have operators for building new nets from existing ones, but Petri Nets do not have any (finite) syntax generating them and so no general theory of composition and decomposition can be defined for them. Therefore, we should restrict our attention to a particular class of nets possessing such a syntax, which thus forms naturally a language for nets. To be more precise, we are interested not only in defining a language whose formulae specify distributed systems, but also in describing the nets representing their behaviour as a *calculus*. Hence, extending Plotkin's paradigm to distributed systems, formulae of the language would denote markings of the net, while net transitions would be defined by means of a syntax-driven deductive system. The title emphasizes this aspect of the paper.

By observing that transition systems, as well as P/T nets, are nothing but ordinary directed graphs, we discover that the notion of directed graph is a possible unifying mathematical tool for investigating the relationship between the two approaches to the semantics of concurrency. Moreover, it can be easily shown that an SOS specification, yielding the interleaving operational definition of a language as a transition system, can be described in algebraic form: the transition system is a two-sorted algebra with states and transitions as sorts [MY89, F90, FM90, Co90]. Therefore, the other common link between the two approaches is the algebraic structure for nodes and transitions. Indeed, SOS specifications and Petri Nets are both specializations of the graph concept obtained by adding (different) algebraic structure on nodes and transitions; in this way, graphs defined as two-sorted algebras represent the uniform framework we were looking for.

A calculus for nets can be introduced by defining an algebra for the nodes of the graph in such a way that it can be seen as a free algebra of markings, generated by the places. Therefore, the algebra must possess, among others, also the commutative monoidal operator $\oplus$. Here, as a case study, we have introduced a Simple Calculus Of NEts (SCONE). It has been admittedly chosen as simple as possible but with the necessary operators for considering it a real language for nets. The combinators generating places are prefixing, nondeterministic composition and the recursive definition construct. The combinators generating net transitions comprise the prefix, local choice, and the synchronization operators. The axioms of the calculus (act, sum-< and sum->, below) represent the set of the generators of the algebra and the inference rule (sync, below) is its sole operation, building a new transition from a pair of given transitions. In this way, the terms of the algebra denote the *proofs* of the transitions in the corresponding SOS specification. To help intuition, a transition is represented in the format $t : \nu - \mu \rightarrow \nu'$ where t is a proof term and $\nu - \mu \rightarrow \nu'$ is the corresponding SOS triple. SCONE is thus described by the following calculus in algebraic form, with axioms for associativity and commutativity of $\oplus$ on nodes and of | on transitions:

act)	$[\mu,\nu\rangle$	:	$\mu\nu - \mu \rightarrow \nu$	
sum-<)	$\nu \ll+ \nu'$	:	$\nu + \nu' - \varepsilon \rightarrow \nu$	
sum->)	$\nu' +\gg \nu$	:	$\nu' + \nu - \varepsilon \rightarrow \nu$	
sync)	$t \mid t'$	:	$\nu_1 \oplus \nu'_1 - \tau \rightarrow \nu_2 \oplus \nu'_2$	where $t:\nu_1 - \lambda \rightarrow \nu_2$ and $t':\nu'_1 - \lambda^{\sim} \rightarrow \nu'_2$.

We want to stress the similarities of our construction with respect to Milner's [Mil89]. CCS is defined as a single whole transition system by means of an SOS specification. Similarly, our net calculus defines a single whole net. Moreover, if one is interested in the behaviour of a particular CCS agent, the relevant piece of transition system is the part reachable from the state corresponding to the agent; analogously, we can single out a sub-net corresponding to a SCONE marking. This is a fairly new result in the context of nets. As a matter of fact, several algebras have been proposed for Petri Nets [K78, GM84, W85, W87, Go88, Ch89] in such a way that a Petri net can be specified by a formula of the proposed algebra, but they

lack the pleasant feature of having a single net comprising all the agent subnets. Recent ideas proposed by Degano, De Nicola and Montanari [DDM88a, DDM88b, DGM88, DDM89] go in the direction of transforming concurrent calculi, like CCS or CSP, into net calculi. In a sense, here we try to algebraically formalize some of the ideas developed there and in other related works [Old89, T89]. A first attempt in this direction is [MY89] where, however, the resulting algebraic structure does not correspond to a net.

According to Milner's paradigm, the next step in giving the semantics of a concurrent language consists of defining the equivalence classes of its computations according to an intended notion of observation. The possible observation out of a net computation is not unique: several notions have been developed, among which we mention *firing* or *step sequences* (sequences of steps each with *one* or *at least one* firing transition), *nonsequential processes* (unfoldings of the net N from an initial marking) [GR83], and *commutative processes* [BD87]. Defining the algebra of Petri net computations by means of the operation of parallel and sequential composition, Degano, Meseguer and Montanari have shown an elegant axiomatization of these notions in [DMM89], where actually a slight refinement of classical nonsequential processes, called *concatenable processes*, is considered. Out of these three different notions, we choose concatenable processes because they faithfully represent causal dependencies and, even more importantly, are equipped with a general operation of sequential composition of partial orders.

SCONE is an extremely simple language. Instead of defining a richer calculus of nets, one can design a truly concurrent calculus as an SOS specification, as usual, and then implement it with a suitable mapping from its transition system to the Petri Net of SCONE. This idea has been greatly influenced by the categorical formulation of Petri Nets, as proposed in [MM88], which provides a flexible tool for relating system descriptions at different levels of abstraction by means of suitable morphisms, called *implementation* morphisms, in the category of net computations. An implementation morphism, indeed, can map a net transition to an entire net computation. As a case study, we apply this paradigm to a sub-set of CCS not dealing with restriction and relabelling, we call RCCS. First, we introduce an algebraic presentation of the SOS specification for RCCS and then we show how to map the algebra of RCCS to the algebra of SCONE. The implementation morphism is another manner of looking at a denotational semantics for RCCS with SCONE as semantic domain. The combinators of SCONE are sometimes more elementary than those of the subset of CCS we model, so that, for instance, a RCCS transition must be mapped to a SCONE computation. Thus, the semantic morphism maps graph transitions to net computations by mapping basic operators of (the algebra of) RCCS to derived operators of (the algebra of) SCONE. The relevance of this result is that this mapping can be seen as an instance of a more general algebraic methodology for *implementing* concurrent languages (also in interleaving form) into others (possibly distributed).

As already observed, e.g., in [T89], the graph representation of an agent in interleaving semantics is usually larger than its net representation. Indeed, not all the RCCS agents have a finite transition system representation. As an example, the transition system reachable from the state "rec x.αx|αx" is infinite. Nonetheless, we prove that for any marking ν, the SCONE sub-net reachable from ν is always *finite*. Therefore, by means of the implementation mapping, we give a finite net representation to any RCCS agent. In the example above, the transition system for "rec x.αx|αx" has a natural net representation in SCONE as a self-loop transition labelled by α with two tokens in the unique place as initial marking. In the concluding section we will discuss the relationship with similar proposals [Go88, T89].

As a by-product of the implementation morphism, we get a "true concurrent" semantics for CCS as a quotient of states and computations of the RCCS transition system. Such a semantics is shown to be consistent with the classic interleaving one, and also with the "true concurrent" semantics given by *permutations* of transitions [BC89, F90, FM90].

The paper is organized as follows. An account of the algebraic formulation of Petri nets and of the axiomatization of processes is presented in Section 2. The SOS specification of RCCS in algebraic form is

given in Section 3, while the proposed calculus of nets is introduced in Section 4. In Section 5 we describe the implementation mapping from RCCS to SCONE and then, in Section 6, we prove that the induced semantics is consistent with respect to Milner's and also faithfully represents causality. In Section 7, we give some hints about the extensions of the present approach needed for dealing with richer variants of CCS, i.e. to deal with restriction; then, we discuss the relations of our investigations with some related works [GMM88, DDM89, Go88, T89] and also with recent results on the connections between Petri Nets and Linear Logic [AFG90, GG89, MaM89]. Finally, appendices are added to help the reader not familiar with category theory and the algebraic construction of Petri processes. Appendix A is an introduction to the basic definitions of category theory used throughout the paper (see [ML71] for more details). In Appendix B we recall from [DMM89] the definitions of categories of symmetries, of processes and of concatenable processes.

2. Petri Nets and Processes

Here we recall the definition of Place/Transition Nets proposed in [MM88], and the $\mathcal{P}[N]$ construction of a slight refinement of Goltz-Reisig processes [GR83], called *concatenable processes*, introduced in [DMM89].

Definition 2.1. *(Graph)*

A *graph* G is a quadruple $N = (V, T, \partial_0, \partial_1)$, where V is the set of nodes (or states), T is the set of arcs (or transitions), and ∂_0, ∂_1 are two functions, called source and target respectively: $\partial_0, \partial_1 : T \to V$. A *graph morphism* from G to G' is a pair of functions $\langle f,g \rangle$, $f : T \to T'$ and $g : V \to V'$ which preserve the source and the target functions: $g \cdot \partial_0 = \partial'_0 \cdot f$ and $g \cdot \partial_1 = \partial'_1 \cdot f$. This, with the obvious component-wise composition, defines the category Graph. ♦

Definition 2.2. *(Petri Nets)*

A *Place/Transition Petri Net* (net, in short) is a graph $N = (S^{\oplus}, T, \partial_0, \partial_1)$, where $S^{\oplus}$ is the free commutative monoid of nodes over a set of *places* S. The elements of $S^{\oplus}$, called also the *markings* of the net N, are represented as formal sums $n_1a_1 \oplus \ldots \oplus n_ka_k$ ($a_i \in S$, n_j is a natural number) with the order of the summands being immaterial, where addition is defined by $(\oplus_i n_ia_i) \oplus (\oplus_i m_ia_i) = (\oplus_i (n_i+m_i)a_i)$ and 0 is its neutral element.

A *Petri Net morphism* h from N to N' is a graph morphism – i.e. a pair of functions $\langle f,g \rangle, f : T \to T'$ and $g : S^{\oplus} \to S'^{\oplus}$, preserving source and target – where g is a monoid morphism (i.e. leaving 0 fixed and respecting the monoid operation $\oplus$). With this definition of morphism, nets form a category, called Petri, equipped with products and coproducts. ♦

In other words, a Petri Net is an ordinary graph where the nodes are defined as an algebra with the elements of the (possibly infinite) set S as generators and $\oplus$ as the only operation which is monoidal and commutative. Notice that finite multisets over a (possibly infinite) set S coincide with the elements of the free commutative monoid having S as set of generators.

To represent net computations observed as processes, we construct certain monoidal categories. Here we do not give a formal presentation of the construction, which can be found in Appendix B, but only an intuitive exposition of the relevant definitions. We first introduce a set of constant transitions, called *symmetries*, and axiomatize them. Given u in $S^{\oplus}$, a symmetry $p : u \to u$ is a transition expressing the fact that in a marking the tokens on the same place can be permuted. Since $n_1a_1 \oplus \ldots \oplus n_ka_k$ is the formal representation of any u in $S^{\oplus}$, a symmetry $p : u \to u$ can be represented as a vector of permutations $\langle \sigma_{a_1}, \ldots, \sigma_{a_k} \rangle$ where σ_{a_i} is a permutation of n_i elements. A suggestive graphical representation of a symmetry p on $3a \oplus 2b$ where $\sigma_a = \{1 \to 2, 2 \to 3, 3 \to 1\}$ and $\sigma_b = \{1 \to 2, 2 \to 1\}$ is depicted in the first operand of Figure