W. Vogler

Modular Construction and Partial Order Semantics of Petri Nets

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

Author

Walter Vogler
Institut für Informatik, Technische Universität München
Postfach 20 24 20, W-8000 München 2, FRG

CR Subject Classification (1991): D.2.2, F.1.2, F.4.3

ISBN 3-540-55767-9 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-55767-9 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera ready by author/editor
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Foreword

The concept of the computer and the ways to use it as well as the understanding of what (theoretical) computer science is or should be have undergone rather deep changes since the middle of the 1930s when the first real computers were designed and built and the first theoretical concepts of computers and computability, of programs and their description were developed and studied. Traditional computer science is based on the paradigm of sequential computations to evaluate functions, briefly sketchable by the metaphor of a single individual considered as a calculator (see Turing's argumentation for the definition of his machine model). Today, the computer serves much more diverse and complex purposes; it has to be considered as a component in a distributed, interactive system composed of humans, computers, machines, and other artificial or natural dynamic systems, such that notions like communication, coordination, collaboration, cooperation, etc., play a more and more important role - and the new metaphor now is a group of individuals engaged in some or all of the four "co"-activities just mentioned (for more details on this, see Lect. Notes in Computer Science 555).

This new view of informatics is not only due to technological developments and application requirements, but also to research in theoretical informatics, in particular in the field of Petri nets. Petri nets allow us to model (distributed, concurrent) systems by a formalism which separately represents (local) actions, (local) states and the (local) interrelations between the holdings of states and the executions of actions (which means that the structure as well as the dynamics are described in the same formalism).

The formalism of Petri nets allows for very detailed descriptions on an operational level. This makes it necessary to develop methodologies and techniques for modular construction of Petri nets and for appropriate behaviour descriptions. Since there are also several other formalisms for concurrent systems, there also exist many modular construction techniques and behaviour notions for such systems - and in particular there has been much discussion on the adequacy of semantic notions, in particular (since Petri introduced the idea in 1976) on the necessity of using partial-order semantics. In this book, Vogler studies this issue in depth - he starts with the most basic techniques of Petri net construction and with requirements on the nets to be constructed, and then shows which notions are minimally required. This way, he can for example prove that failures semantics (which originally was developed for TCSP) is just the right notion to characterize the constructability of deadlock-free nets by TCSP-like parallel composition. The most important contribution perhaps is Vogler's study of action refinement - he was the first to provide formal results on the adequacy of partial order semantics and branching equivalences for the study of action refinement, and moreover he even shows that only a restricted type of partial orders (the interval orders) are necessary.

Vogler's approach allows him to systematically study the broad spectrum of construction, semantics and equivalence notions in such a way that many of their inherent properties and their interrelationships become much clearer, and many more or less philosophical discussions of the past are now obsolete. On the basis of this (and selected work

by other authors), it should now be possible to study problems of practical applications of these notions within concrete methodologies of system construction.

München, June 1992 Wilfried Brauer

Preface

Petri nets are a well-known model for parallel systems, used for both applications and theoretical studies. Like any formal model, they can be used for specification, modelling and analysis; Petri nets in particular offer a graphical representation and a clear view of concurrency. For the design of large systems, modular construction is indispensible; hence, considerable effort has been spent on studying the modular construction of Petri nets. This book presents some contributions to this research area.

In bottom-up design, nets are put together and the intention is to determine the behaviour of the composed system from the behaviour of its components; as operators for the combination of nets we consider parallel composition with synchronous and with asynchronous communication. For the top-down design, we study the refinement of the elementary parts of nets, i.e. of places and transitions. A refinement step is performed with one of two possible intentions in mind. Either the refined net is expected to have the same behaviour as the unrefined net, in which case we speak of a behaviour-preserving refinement; or we expect that refining two nets with the same behaviour leads to nets that have the same behaviour again, in which case we speak of an equivalence-preserving refinement; the equivalence-preserving refinement of transitions is also called action refinement.

This book presents behaviour descriptions that support these modular construction methods of nets. Many such descriptions are possible. Therefore, special care is taken to justify the descriptions presented here by showing what is called full abstractness; i.e., when considering some construction method, we not only present a suitable behaviour description, but also show that it makes exactly those distinctions of nets that are necessary to support the given method and to take into account some simple feature of behaviour like deadlock-freeness. For example, failure semantics is the right behaviour description for constructing deadlock-free nets using parallel composition with synchronous communication. As one of the highlights, we show that in order to support action refinement and to take into account failure semantics some form of partial order semantics is necessary.

This work is a revised version of my *Habilitationsschrift* written at the Technische Universität München. It would not have been possible without the support I have received from many people. First of all, my thanks go to Professor W. Brauer for the good working atmosphere he has created in his group and for his helpful advice and valuable comments over the last few years. I also would like to thank Professors M. Broy and M. Nielsen, who acted as further referees of my *Habilitationsschrift*. I am particularly grateful to Professor R. Halin, who guided my way through graph theory before I changed over to computer science.

I have profited from numerous discussions with many people, and I am especially grateful to all my former and present colleagues from Hamburg and München. For many years, Dirk Taubner has shared an office and his knowledge especially on failure semantics with me, and our discussions have helped me a lot. I am also greatly indebted to Eike Best, Jörg Desel, Volker Dickert, Rob van Glabbeek, Robert Gold, Ulla Goltz, Astrid

Kiehn, Wolfgang Reisig, Thomas Tensi, and Rolf Walter.

Work on this book was partially supported by the Deutsche Forschungsgemeinschaft, Sonderforschungsbereich 342: Methoden und Werkzeuge zur Nutzung paralleler Rechnerarchitekturen, TU München, and the ESPRIT Basic Research Action No. 3148 DEMON (Design Methods Based on Nets).

Last not least, I thank Harald Hadwiger and Dieter Stein, who have helped me enormously to transform my notes into a LaTeX document.

München, June 1992 Walter Vogler

Contents

Chapter 1

Introduction

A concurrent system, such as a network of processors, an operating system, or a manufacturing system, consists of several partly autonomous components, which run in parallel and influence each other by interactions. Thus, to determine the behaviour of the system it is not enough to know how each of its components in isolation transforms initially given input objects like data or raw materials into output objects produced at the end. In the case of a sequential system this would be sufficient; for example, the behaviour of a procedure in a sequential program can be defined as a function from memory states to memory states. But for a concurrent system, we also have to know how each component reacts to outside influences and how it influences its environment while it is running. In fact, the same applies to the entire system in so far as it interacts with its environment and is thus itself a component of a larger system. Furthermore, the activities of the components may be unsynchronized; as a consequence the interaction of the system components may have the effect that the entire system behaves nondeterministically, even if its components are deterministic. For example, if two senders share a channel, the behaviour of the system may depend on the timing of their messages.

In view of these complications, the design of concurrent systems particularly requires a formal method. In the design process an informal conception is transformed into a formal system model, and the first benefit of the formal model is that its development helps to uncover deficiences and ambiguities in the informal conception. Once the model is completed, it can be analyzed formally and relevant properties can be verified, or at least the concurrent system can be tested by means of a simulation.

In the approach we adopt in this book, the behaviour of a concurrent system is described in terms of the actions it can perform. Here an action is any activity that we view as a conceptual entity; in particular, it may be an act of communication. A simple behaviour description of this kind is the set of all possible sequences of actions. But this semantics has been criticized in two points.

First, the concurrent execution of actions is seen as equivalent to arbitrary interleaving, i.e. to executing these activites in an arbitrary order. Thus concurrency is simply reduced to some form of nondeterminism. Such a semantics is called an interleaving semantics. Alternatively, one could try to represent concurrency explicitly, e.g. by describing a system run by a partial order of actions. Such a semantics would be 'truly concurrent'.

Secondly, the above semantics gives no information about the nondeterministic choices

that have been made during a system run, about the branching structure of the system behaviour. Such a semantics is called a linear-time semantics. In order to give a branching-time semantics one must not only compare system runs, but also consider how internal conflicts are resolved. Prominent branching-time semantics are failure semantics [BHR84] and bisimulation [Par81,Mil83].

These considerations show that there is more than one way of defining the behaviour of a system. Which definition is chosen depends on the system properties that are regarded as relevant. It is even feasible to view a simple property as the behaviour of a system, e.g. the semantics of a system possibly is just to be or not to be free of deadlocks. Thus we cannot expect to find *the* behaviour of a concurrent system. Instead we can compare various semantics and study their properties. A most important requirement for a semantics is that it support the modular construction of systems.

To reduce and manage system complexity we have to design large systems in a modular fashion either bottom-up by composing subsystems with known behaviour in such a way that we can determine the behaviour of the whole system from its parts, or top-down by refining parts of a rough model by more detailed system descriptions. In the latter case we either ensure that the behaviour is essentially preserved or proceed again in such a way that we can determine the behaviour of the refined system from that of the rough model and the refinement, in which case we speak of an equivalence-preserving refinement. In all these cases, systems are constructed from building blocks, and a semantics supports the modular construction of systems if it describes the behaviour of the building blocks, i.e. their interfaces, in such a way that we can control the behaviour of the entire system as just described.

This book contributes to the theory of designing concurrent systems with Petri nets. A Petri net is a formal system model based on concepts from automata theory, linear algebra and graph theory. Besides the general advantages of a formal model and the verification methods based on linear algebra, Petri nets are additionally attractive since – as graph-theoretic objects – they have a graphical representation. Already in the design process this graphical representation offers a visual impression of the concurrent system and how it is built from subsystems and distributed in space; it gives a clear image of concurrency, sequentiality and conflict, both on the concrete visual level and on the abstract graph-theoretic level.

In particular, the visualization of concurrency makes it very natural to consider concurrency as a feature that deserves a proper presentation on the semantic level. Petri net theory has a long tradition in studying 'true concurrency' in the semantics of concurrent systems. Most often 'true concurrency' is captured by giving a semantics based on partial orders, and partial orders also invite a graph-theoretic representation as Hasse diagrams. On the other hand, the branching structure of systems has not been given much attention.

It must also be mentioned that modularity as described above has been a somewhat weak point of Petri net theory. A Petri net is defined as a whole and not in the first instance obtained by composing subnets; correspondingly its semantics, i.e. the firing rule or a derivative of it, does not rely on the semantics of some subnets, although the firing rule is local in character.

This is totally different in process algebras like CCS [Mil80,Mil89], TCSP [BHR84,

Hoa85] or ACP [BK84]. Here systems are described by process terms, which are by nature built from subterms. Naturally, the semantics of a process term is obtained from the semantics of its subterms, no matter whether the semantics is an operational semantics defined according to the structured operational approach of [Plo81] or a denotational semantics. Thus process algebras are a priori compositional.

Traditionally, in process algebras concurrency has been reduced to interleaving; this may be due to their roots in algebra. Often it has been argued that interleaving is simpler than 'true concurrency' and just as expressive, i.e. sufficient for any practical purpose. Instead, the emphasis has been on studying the branching structure of processes. Thus, while neglecting the dimension of interleaving versus 'true concurrency', process algebra has concentrated on the orthogonal dimension of linear-time versus branching-time semantics, and vice versa for Petri net theory.

In recent years, both these approaches have increasingly influenced each other, and a lot of effort has been made to combine their respective merits. Partial order semantics for process terms have been developed, see e.g. [BC87,DDNM88,Old89,NEL89,Ace89]. Furthermore, semantics in terms of Petri nets have been given to process algebras, such that a process term, which is an operator applied to some subterms, is translated to a net that is an appropriate composition of nets related to those subterms; see e.g. [GV87,Gol87, Gol88b,Tau89]. This allows one to give some partial order semantics to process algebras by translating a term to a net and taking (one of) its partial order semantics. Viewed the other way, those nets that are translations of process terms form a restricted class of nets for which several compositional semantics can be given if we apply results from process algebra. Similarly, several authors have suggested solving the compositionality problem of Petri nets by working with restricted, more structured classes of nets that are built from very simple nets, see e.g. the state-net-decomposable nets studied in [Bes88a], and see [BDC92] for a survey.

Other authors have concentrated on solving the compositionality problem for the unrestricted class of all nets. They have suggested various transformations that are behaviour-preserving in some sense, and various composition operators such that the behaviour of a composed net (in some sense) can be determined from the behaviour of its components, see e.g. [And83,Bau88,Ber87,DCDMS87,Gra81,Mül85,Sou91,SM83,Val79,Vos87]. This is the area in which this book is located.

The system models in this book are labelled place/transition-nets without capacities, i.e. place/transition-nets where the transitions are labelled with actions. As indicated above, these are uninterpreted names of activities. Their use allows one to abstract from details of a system description that are of no importance for the user of the system. Transitions with the same label represent the same activity in different internal situations. Very important is the use of λ-labelled transitions, which represent internal activities that are invisible for the user; thus we can abstract from activities that are important on a low-level system description when we consider the system behaviour on a higher level.

For the modular construction of nets we concentrate on two sorts of operators: parallel composition with synchronous or asynchronous communication for the bottom-up design of nets, and refinement of transitions and places for the top-down design. These operators are especially interesting since they are also graphically meaningful. Given two nets,

parallel composition corresponds to composition by merging transitions in the synchronous case and to merging places in the asynchronous case. Given one net a refinement replaces a basic net element, i.e. a place or a transition, by some net, thus giving a more detailed description of a local state or an activity. Other operators are only touched upon, e.g. hiding, which allows one to abstract from details by turning some visible actions into internal actions.

Typical for the approach of this book is the situation in Chapter 3. We define an operator $\|$ (more precisely a family of operators indexed by a set of actions that are to be synchronized), and then we want to find out which nets, if combined with any environment via $\|$, can be exchanged without changing the behaviour of the composed net; we call such nets *externally equivalent* following [Bau88]. At the same time we want to explore which behaviour notions are suitable, since there is no general agreement about this point. This makes the situation slightly obscure: either we should fix our behaviour notion, and then we can try to characterize externally equivalent nets; or we should fix the exchanges of nets we want to carry out, and then we can try to find out which behaviour is preserved by such exchanges. Fortunately, our approach leads to quite a satisfactory solution for this matrix of problems. First, we fix a very simple sort of behaviour, namely we just distinguish deadlock-free nets from those that can deadlock. Then we give an *internal* characterization of the corresponding external equivalence, i.e. we determine when nets can be exchanged in any environment without referring in our characterization to all possible environment nets; namely, nets are externally equivalent if and only if they have the same failure semantics. When proving this we find that exchanging failure-equivalent nets preserves the failure semantics of the composed net. Thus our composition operator together with failure semantics is compositional in the sense that we can determine the behaviour of a composed net from the behaviour of its components. At the same time we discover that exchanging failure-equivalent nets preserves behaviour in a much stronger sense than originally required; thus our equivalence works for a whole range of behaviour notions.

External equivalence is closely related to testing equivalence in the sense of [DNH84], which refers to a notion of observability. In principle the reasoning for the above external equivalence can also be expressed in terms of observability. But in this book the argument is not that deadlock or divergence (infinite internal looping) are observable in some sense, but rather that these are important features of behaviour that we must control in the modular construction of a system. External equivalence is also closely related to full abstractness [Mil77], which is a stronger requirement: it considers the exchange of two nets in any context built by applying the operators under consideration possibly many times, while we consider contexts where we have only one application of an operator. In other words, an internal semantics characterizing an external equivalence describes the interface of a building block such that this description is sufficient to deduce the relevant behaviour of a *system* constructed from two building blocks; on the other hand, a fully abstract semantics describes the interface of a building block such that we can determine the interface of a *building block* constructed from two building blocks, and thus we can determine the relevant behaviour of a system constructed from any number of building blocks. Naturally, the latter is to be preferred in general. But in all natural cases we consider, external and fully abstract equivalences coincide, and thus our results

are stronger if we start with the weaker requirement, i.e. if we start with the study of external equivalences.

Our results can also be seen as a justification of failure semantics; they show that failure equivalence is just the right equivalence, if we want an equivalence that is compositional with respect to $\|$ and are mainly interested in the deadlocking behaviour of systems. This view is especially interesting in Chapter 5, where we consider action refinement. This operator refines some action a by a more detailed description of this activity; it replaces *every* a-labelled transition by a copy of some net. Regarding the dispute of interleaving versus 'true concurrency', our results in Chapter 5 show that for a congruence for action refinement that respects e.g. failure equivalence the power of partial order semantics is needed. Thus we justify the use of 'true concurrency'.

As explained above, by modular construction we understand either composition or refinement, and the latter is subdivided into behaviour- and equivalence-preserving refinement. Composition is studied in Chapters 3, 4, and 7. Chapters 3 and 7 are concerned with synchronous communication, where Chapter 7 studies nets with capacities contrary to the model we use in general. Asynchronous communication is treated in Chapter 4. These chapters develop suitable interface descriptions for the composition of systems from building blocks. Behaviour-preserving refinement is studied in Chapter 4; modules are characterized that are suitable for replacing a transition or a place in any context, and these results are obtained by putting behaviour-preserving refinement in the framework of composition. Chapters 5 and 6 are devoted to equivalence-preserving refinement; Chapter 5 is concerned with linear-time and failure semantics, Chapter 6 with various types of bisimulation. Here the emphasis is on the behaviour of the rough model. We develop interface descriptions for the rough model that together with the inserted refinement nets allow us to deduce the interface description of a partly refined model and finally the relevant behaviour of the detailed system.

In more detail, we proceed as follows. Chapter 2 introduces Petri nets, where Section 2.1 briefly reviews the basic notions. Section 2.2 is devoted to the linear-time partial order semantics of Petri nets; we define the well-known processes (of nets) and partial words and adapt them to labelled nets. Complementarily, Section 2.3 describes some points in the linear-time/branching-time spectrum for the interleaving case; we define two failure-type semantics, one taking account of divergence and the other not, and some versions of bisimulation.

In Chapter 3 we study parallel composition with synchronization of actions from some given set; as described above, we show that the two types of failure semantics we have introduced are just right for a compositional semantics if we are mainly interested in deadlock-free or deadlock- and divergence-free systems. In Section 3.3 we study some modifications. One concerns the treatment of infinite system runs, and we touch upon the problem of fairness. We consider an adaption to safe nets and to the case where we are interested in liveness (in the Petri net sense) instead of deadlock-freeness. In Section 3.4 we mention the further operators hiding, relabelling, and the choice operator.

If we restrict ourselves to the exchange of nets that are in some sense deterministic, we can improve our results. Not only do we get more favourable decidability results, we also can show that our simple requirement, that the exchange of equivalent nets preserves

deadlock-freeness, guarantees behaviour preservation in a much stronger sense; such an exchange results in a net that is bisimilar to the original net; see also [Eng85]. This is presented in Chapter 4, where in particular we show how these results can be applied to the refinement of places. Dually, we initiate the study of a parallel composition operator with asynchronous communication in Section 4.3; in Section 4.4 we explore which features make a net deterministic in this context, and show how the behaviour-preserving refinement of transitions fits into this framework.

Often we cannot expect that the refinement of transitions preserves behaviour, since the refined net may be able to perform some new actions that were not present in the unrefined net. In this case we would like to be able to determine the behaviour of the refined from that of the unrefined net. Thus equivalent nets, i.e. nets with the same behaviour, should be refined to nets that are equivalent again; in other words, the equivalence should be a congruence for action refinement. Such equivalence-preserving action refinements are studied in Chapter 5 and Chapter 6. In Chapter 5 we introduce a technique for action refinement and discuss, which refinement nets are suitable in this context. We show that partial order semantics is useful for defining congruences with respect to action refinement in Section 5.3, and we introduce a partial order semantics based on interval orders. This turns out to be just the right semantics in the sense that interval semiwords can be used to define three semantics that are fully abstract for action refinement and language-, failure- and failure/divergence-semantics respectively. We show this in Section 5.4, where we use interval words, a more or less sequential presentation of interval semiwords. The translation between these two descriptions of system runs is presented in Section 5.5 together with some decidability results.

In Chapter 6 we discuss congruences for action refinement of bisimulation-type. While partial orders are immediately useful for linear-time congruences, pomset bisimulation [BC87], a straightforward combination of partial order semantics and bisimulation, has turned out to fail for this purpose [BDKP91,GG89b]. History-preserving bisimulation, a more intricate combination of partial order semantics and bisimulation, is a congruence [BDKP91,GG89b]; but even this fails unless we restrict the use of internal actions. On the other hand, ST-bisimulation [GV87], which makes no explicit use of partial orders but is in fact closely related to interval semiwords, gives a congruence. We show that the ST-idea can be used to lift in a uniform way bisimulation, pomset bisimulation, history-preserving bisimulation and the newly introduced partial-word bisimulation to congruences with respect to action refinement without any restriction on the use of internal actions. At least in the first three cases we can also show full abstractness results.

For these considerations we restrict ourselves to event structures [NPW81], which can be seen as a special class of Petri nets, which are in particular acyclic. In many ways, this makes event structures theoretically easier to work with; but they have the considerable disadvantage that they have to be infinite in order to describe an infinite behaviour. The corresponding advantage of general Petri nets is somewhat lost when we work with history-preserving bisimulation. This type of bisimulation gives a detailed account of the interplay of causality and branching, and it has turned up in various papers – not only in the context of action refinement; but unlike the usual bisimulation it relates system runs instead of system states, and thus it necessarily refers to infinitely many objects if we are concerned with infinite behaviour. In Section 6.4 we give an alternative definition of a

bisimulation for safe nets without internal transitions; in this definition each system state is described by a marking together with a pre-order on its tokens, where the pre-order contains information on causality in the token generation. We can show that this OM-bisimulation (OM = ordered marking) gives the same equivalence as history-preserving bisimulation. As a corollary we obtain that history-preserving bisimulation is decidable.

Chapter 7 looks at partial order semantics from another angle. Continuing a research initiated in [HRT89], we consider compositionality for nets *with* capacities. Here we are not concerned with full abstractness; instead we consider quality criteria of partial order semantics based on net transformations that are very natural in the presence of capacities. We give several characterizations and define a new partial order semantics that seems to be the natural choice in this framework, since it is the minimal semantics satisfying all our criteria.

In total, we will present more than forty different net semantics. As explained above, it must be left to the reader to choose the right one for a specific application. For example, if interleaving semantics is all the reader is interested in, attention can be restricted to Chapters 3 and 4 after reading appropriate portions of Chapter 2, but then a hierarchical design by action refinement cannot be accomplished. Conversely, if the reader wants to know a good reason for partial order semantics or if action refinement is the main operator of interest, then the reader should turn to Chapters 5 and 6.

In many applications it will turn out that only safe nets are needed. In this case Chapter 7 can be left out, which is only interesting if we have varying capacities. Also, in some cases our decidability results are based on the decidability of the reachability problem, but they become quite simple in the case of safe nets. In this book, emphasis is put not only on showing that some semantics is sufficient to guarantee certain properties for the modular net construction, but also on the necessity of the distinctions made by the semantics. These necessity results can fail if the nets under consideration are restricted to some subclass; therefore it is important that we keep an eye on this practically important class of safe nets, sometimes developing appropriate variations as in Section 3.3.

For applications it may look like a severe restriction that the actions, which label transitions, are just uninterpreted names; but in principle we can also deal with arbitrary data. For example, if we have an action '$input(n)$', where n is meant to be a natural number – although formally '$input(n)$' is just a meaningless name –, then we can use actions '$input(n)_1$','$input(n)_2$', ..., one for each value of n. Of course, in this way the input of n requires infinitely many transitions, and correspondingly the variable n is modelled by infinitely many places, one for each value of n. This approach is perfectly sufficient for theoretical investigations as we present them here. For practical applications, most often some form of high-level net will be more adequate; see e.g. [Gen87,Jen87,Rei90]. High-level nets can be seen as an abbreviation for place/transition-nets, and thus one can expect that our results carry over; often these place/transition nets are – possibly infinite – safe nets, which underlines the importance of safe nets. On the other hand, using high-level nets it is often desirable to work on a symbolic level, and here a lot of work remains to be done in order to transfer our results.