Understanding Petri nets. Modeling techniques, analysis methods, case studies. Translated from the German by the author

Book · July 2013

DOI: 10.1007/978-3-642-33278-4

W. Reisig
Humboldt-Universität zu Berlin
267 PUBLICATIONS **7,250** CITATIONS

Petri Nets

Modeling Techniques, Analysis
Methods, Case Studies

Wolfgang Reisig

February 22, 2013

Zuerst einmal kommt es darauf an,
verstanden zu werden.

Theodor Fontane

Put it before them briefly so they will read it,
clearly so they will appreciate it,
picturesquely so they will remember it
and, above all, accurately so they will be guided by its light.

Joseph Pulitzer

Progress is possible only if
we train ourselves to think about programs
without thinking of them
as pieces of executable code.

E.W. Dijkstra

Computer Science
is no more about computers
than astronomy
is about telescopes.

E.W. Dijkstra

Carl Adam Petri (1926 – 2010)

1988	Officer’s Cross of the Order of Merit of the Federal Republic of Germany
1988	Honorary professor at the University of Hamburg, Germany
1989	Member of the Academia Europaea, London
1993	Konrad Zuse Medal of the German Informatics Society (Gesellschaft für Informatik) for excellent achievements in computer science
1997	Werner von Siemens Ring for distinguished achievements in technology and science
1997	Member of the New York Academy of Sciences
1999	Honorary doctorate of the University of Zaragoza
2003	Commander of the Order of the Netherlands Lion
2007	Gold Medal of Honor of the Academy of Transdisciplinary Learning and Advanced Studies
2009	IEEE Computer Pioneer Award

Preface

Foreword

Dear reader:

The currently common graphic representation for nets as well as the methodology of their refinement and abstraction was invented in August 1939 to visualize my knowledge about chemical processes. I also devised a plan to analyze catalytic processes.

This plan failed due to the beginning of the Second World War, because the necessary equipment was no longer available. So I had no other alternative but to study my invention on paper.

In 1941, I was so excited about my father's report on Konrad Zuse's programmable computer that I could not stop thinking about it. However, it was not until the late 1950s that it became evident that the development of computer technology would greatly influence our society.

It seemed to me that a theoretical framework was needed to formulate and solve the basic problems that arose for the structure and the sensible use of this computer technology – independent from the current state of the technology, but in accordance with the laws of physics. I decided to lay the foundations of this framework. Indeed, I remembered my toy from 1939 and applied it to signals and bits, but formulated my results predominantly in mathematical form to achieve a higher acceptance of my dissertation on "Communication with Automata" (original German: "Kommunikation mit Automaten"). (Only after my exam, did I reintroduce the net graphics.)

My arguments were:

- The theory of computability is based on the Turing machine, consisting of an abstract automaton and an infinite tape. While the automaton can be implemented by means of a computer, the infinite tape cannot.

- To avoid a contradiction to the laws of physics (finite, invariant speed of light and the uncertainty principle), the infinite ribbon has to be replaced with two short shifting registers. Attached to the end of each register is a register cell factory that works at least as fast as it takes to access the register.

- The cells of the register are only allowed to communicate with their two neighbors, just as each transition can only affect its immediate neighbors.

- That means that the computability theory current at that time, with its global state changes, cannot be applied to the kinds of distributed systems that the technology is developing towards. Trying to conceive the Internet as a single automaton is simply not useful.

Since then, many others have helped to develop the net theory of distributed systems into an established field of knowledge, with many computer tools for the study of system properties hidden within the graphical notation, like behavioral invariants or the potential disassembly into components.

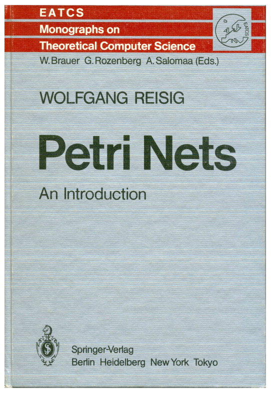
This book chooses the most important, most original, most successful and most basic concepts and presents them comprehensively. In doing so, it offers students and practitioners, in some cases even specialists of the field, an approach to or a new view on distributed systems.

I wish this book a wide distribution and a long usage.

St. Augustin, Germany, January 2008

Carl Adam Petri

Preface



EATCS
Monographs on
Theoretical Computer Science
W. Brauer G. Rozenberg A. Salomaa (Eds.)

WOLFGANG REISIG

Petri Nets

An Introduction

Springer-Verlag
Berlin Heidelberg New York Tokyo

Petri Nets – An Introduction [63] is a slim book that was published almost 30 years ago and was quickly translated into six languages and sold several thousand copies. For a long time, it was considered the standard reference for Petri nets.

Now it is time for a new introduction. Petri nets have been further developed in unbelievably diverse directions. “First model, then program” is a principle that increasingly dominates software engineering, and Petri nets are a popular modeling technique.

The biggest problem in writing an introduction to a topic is the selection of the content. Today we have a better understanding of which modeling techniques and analysis methods are truly central than we had 30 years ago. Those techniques and methods are presented here. Their usefulness is illustrated with the help of several examples, particularly the case studies in the last part of this book. The software engineer can use these case studies as a guide. At the same time, the theorist will find the classic results of Petri net theory, complemented with a few new concepts and a new taxonomy.

The formal arguments in this book are reduced to a minimum. Examples often have enough detail to sufficiently demonstrate the characteristics of the content. Furthermore, individual chapters can often be read independently from one another. For instance, the basics introduced in the first three chapters of Part I are enough to understand the case studies in Part III. Only their analysis requires knowledge of the methods presented in Part II.

This book was not written for a specific audience such as students, teachers, theoretical computer scientists or software engineers. Instead, it addresses a broad audience by compiling the central developments of the last 50 years of net theory and practice and presenting it in a comprehensible way. The selection of aspects presented here is, therefore, highly subjective.

Acknowledgments

This book was originally published in German. The German readers provided very valuable feedback and pointed out some inconsistencies in the terminology as well as various typos. Thanks to all of them.

From the late 1970s, Carl Adam Petri, in numerous conversations, explained to me his proposals for a basic theory of information processing systems. Without these conversations, this book would not have been possible. Jörg Desel contributed important arguments for the selection of the content. His advice influenced some chapters decisively. Since the early drafting phase, Rüdiger Valk pointed out numerous typing errors, inconsistencies, ambiguities and interesting citations to me. Before submission to the publisher, some colleagues proposed changes to the manuscript, which I gladly took up. In particular, Katharina Görlach, Jana Köhler, Bernd Krämer, Ernst Mayr, Agathe Merceron, Birger Peil, Christian Stahl, Hagen Völzer and Walter Vogler made valuable suggestions. I also thank Daniel Janusz, Andreas Hoheisel, Olaf Hochmuth, Rolf Hoffmann and Dominic Wist for pointing out inconsistencies. Annika Flemming made large contributions to the development of the exercises.

My special thanks to Birgit Heene. Patiently and tirelessly, she typed and arranged the numerous draft versions. Dirk Fahland, Katharina Görlach and Simon Heiden assisted her with great commitment.

Michael Rücker then took on the burden of translating the German text into English. I think he did a great job (also in putting up with my continuous criticism). My very special thanks goes to Tim Denvir, whose relentless scrutiny of the draft version found many mistakes and spawned various discussions on some amazing subtleties of the English language. With Ronan Nugent at Springer-Verlag, I shared numerous (and quite humorous) conversations about the book's shape and layout. I am grateful that he accepted most of my wishes. He also organized yet another proof reading process and engaged himself personally.

As a substantial extension, the English version also includes a concise compilation of the formal framework that is used throughout this book. For easy navigation and cross-reference, the relevant terms are highlighted in the margin of each chapter as well as in the formal framework itself.

Wolfgang Reisig, Berlin, February 2013.

Introduction

What Are Petri Nets and What Should They Be?

A central challenge of computer science is the appropriate construction of *systems* that contain IT-based components and that are embedded in an automated or organizational environment. Such systems are *modeled* so that clients, manufacturers, users, etc., can, with the help of the model, better understand what a system is supposed to do and how it can be implemented, used, varied and improved.

In the past 50 years, a multitude of modeling techniques has been proposed for such systems. They all have their strengths and weaknesses, as well as their preferred fields of application, user groups and tools. Petri nets are among the oldest modeling techniques of computer science.

For decades now, interesting theoretical questions about Petri nets have been posed and solved. Special subclasses have been studied and tools have been developed. Case studies have been conducted and successful projects have been implemented. In contrast to some other methods, which were favored for a short time and then forgotten, Petri nets have kept their place as one of the well-established modeling techniques. Initially, there was an interest in integrating Petri nets with other methods. However, over time, this has given way to a tendency to keep them separate.

With a lively, steadily growing group of interested researchers and users as well as ever-new software tools, for decades Petri nets have been in the best position to shape some foreseeable developing lines of computer science and to contribute to novel concepts like “model-based,” “ubiquitous,” “pervasive” or “disappearing” software engineering. The Petri net portal `www.informatik.uni-hamburg.de/TGI/PetriNets/index.html` offers rich material on this.

Carl Adam Petri himself originally designed his approach even more broadly. He looked for a theory of information processing in accord with the laws of physics, with deep-seated invariants in the traditions of the natural sciences and with formalisms that would allow for the description of a human-oriented, pragmatic handling of the technical possibilities of computer science.

What Does This Book Cover?

The book is divided into three parts. The first part describes how to use Petri nets for modeling. All concepts are explained with the help of examples; at first, with different variants of a cookie vending machine. We use the most powerful, most generic type of Petri nets for this (with real-world objects as "tokens"). It is not until Chapter 3 that we will use the more specialized and technically simpler type of *elementary* system nets with abstract "black" tokens. I chose this order so as to start with an intuitively very convincing and realistic model. From there on, I always introduce derived concepts like *sequential* and *distributed* runs, *scenarios* and additional notations for elementary system nets first, because they are more comprehensible this way. The examples are meant to show that scenarios, in particular, decisively deepen our understanding of a system and deserve special attention. The synthesis problem, i.e., deriving non-sequential behavior from a description of sequential behavior, can be solved very convincingly with Petri nets. It is therefore covered in a separate chapter.

The second part introduces techniques with which important properties of system nets can be formulated as well as algorithms with which one can prove or disprove their validity. For easier understanding, some concepts are only introduced for elementary system nets at first. This second part covers all essential analysis methods that are specific to Petri nets, particularly traps, place invariants, transition invariants, covering graphs and special techniques for free-choice nets. Some readers may be unfamiliar with the combination of the analytical power of traps and place invariants, the consistent distinction between “hot” and “cold” transitions, and the discussion of run properties. Temporal logic and particularly model checking offer generic analysis techniques, which are also successfully applied to Petri nets. This book will not explicitly cover such techniques, because they are not specific to Petri nets. However, ideas from temporal logic frequently influence this text implicitly, particularly the distinction between state and run properties.

The third part presents a selection of three case studies from very different fields. They show how diversely applicable Petri net models are. In addition, each case study introduces new, more generic concepts, properties and analysis techniques, which are also useful for very different modeling tasks. Among them are fairness properties, the combination of region theory and state properties and symbolic schemata for Petri nets.

In summary, the three parts of this book explain:

- Petri nets as a modeling technique,
- the theoretically well-founded analysis of Petri nets,
- case studies from very different areas of application.

Each chapter ends with exercises and recommendations for further reading. Particularly challenging exercises are marked with an asterisk. Texts with a colored background focus on historic or exemplary aspects of the main text. Finally, the appendix includes a compilation of the formal framework used throughout this book.

Conclusive Threads

The previous section has already mentioned the special case of *elementary* nets (with “black tokens”). Readers who only wish to study this type of net can follow the *elementary strand* of this book. Figure 1 shows which chapters and sections belong to this strand.

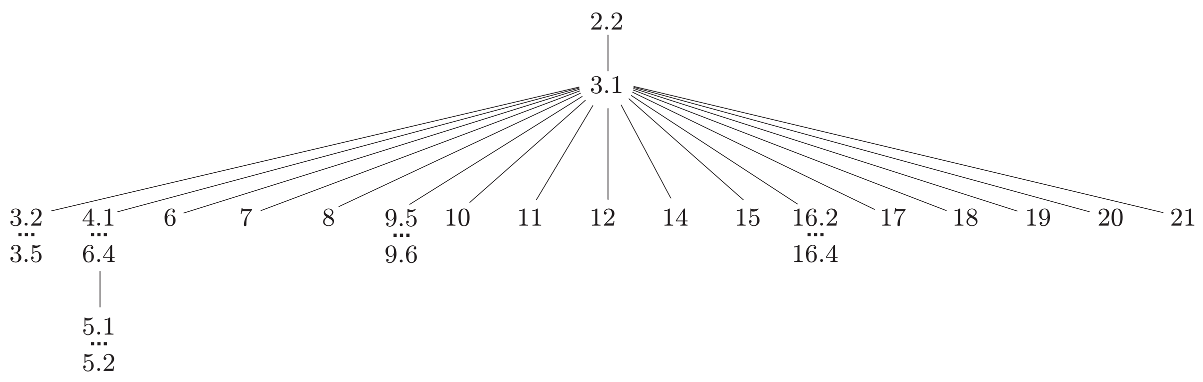


Figure 1: The elementary strand

Readers who only wish to study the modeling (but not the analysis) of systems can follow the *modeling strand*. This strand contains all the models of real systems appearing in this book. Figure 2 outlines the chapters and sections of this strand.

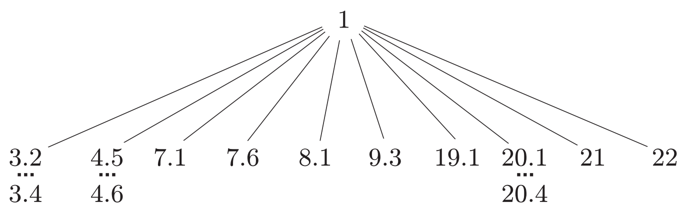


Figure 2: The modeling strand

Readers who are interested in technically simple, but challenging examples and case studies can choose the elementary models of the modeling strand, shown in Fig. 3.

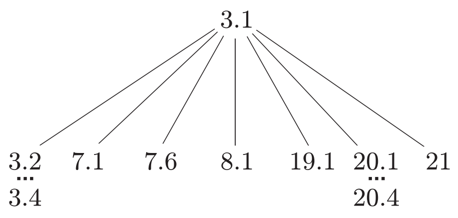


Figure 3: The elementary modeling strand

How Is the Content Presented?

Pertinent examples and intuitive descriptions often explain the content sufficiently well. In places where formal arguments are used, only basic standard notations are employed, for instance, for sets, functions, graphs, vectors, matrices, systems of equations and propositional logic. Everything beyond that (for instance, the handling of multisets) is explained in detail. The area of Petri nets has grown so rapidly that there is not yet any overall consistent and intuitively appealing terminology. This text constantly looks for compromises between consistency (identical names for identical things, different names for different things), intuitive comprehensibility and linguistic correctness of terms and notations. Some (few) terms are newly introduced here or are used in a new way. Readers who want to avoid formalism can still acquire a sufficient understanding of the content.

The page layout is designed to break up the text. The broad margin contains examples, annotations and alternative formulations. In detail, this means:

- The margin illustrates the text: the examples and annotations in the margin explain and illustrate the concepts in the text.
- The text explains the margin: readers who are (somewhat) familiar with Petri nets can often get along by only looking at the examples and annotations in the margin. The text then serves as confirmation and generalization.

Even without an analysis, Petri nets are a very interesting modeling technique. Chapters 1 and 3 are sufficient to understand the case studies in Part III.

When looking for a particular passage in the text, the reader can find help in the list of Examples and Case Studies in the Frontmatter or in the Index at the end of this book. Further information, feedback from readers and material for classes can be found at: `https://u.hu-berlin.de/understandingpetrinets`

Examples and Case Studies

Contents

Part I

Modeling Techniques

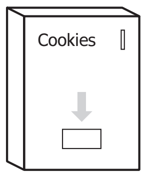
This first part is about modeling with Petri nets. In Chapter 1, we start with an example that employs the basic notations and shows how to use them appropriately. Chapter 2 then generalizes these notations. The common special case of *elementary system nets* is covered in a separate chapter. Finally, Chap. 4 juxtaposes the concepts of *sequential* and *distributed runs*. Chapter 5 then uses distributed runs to describe *scenarios*. Chapter 6 introduces additional notations and shows which are merely useful abbreviations and which actually increase the expressive power of elementary system nets. Part I concludes with the solving of the *synthesis problem* and the *composition* of nets.

An Example — Chapter 1

In this chapter, we will use an example to explain the basic graphical components of Petri nets and how they can be used to model discrete systems.

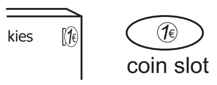
1.1 A Cookie Vending Machine

For this introductory example, we describe a *vending machine* that sells cookies. The machine has a *coin slot* and a *compartment* into which the packets of cookies are dropped. In the *initial state* of the cookie vending machine, the coin slot contains a coin. The cookie compartment is empty.



a cookie vending machine

Figure 1.1 models this as a Petri net: coin slot and compartment, both depicted as ellipses, are the *places* of the Petri net. The coin slot contains a euro coin, which is a *token* of the net in Fig. 1.1. The compartment does not contain any tokens. A distribution of tokens across places is a *marking* of a Petri net.



The machine can now collect the coin and produce a packet of cookies. In the Petri net in Fig 1.1, this is modeled as a *transition* t, depicted as a square (or rectangle).

transition

In Fig. 1.1, the transition t is *enabled*, because its incoming arc starts at a place containing a coin token, as required by the arc's label. Therefore, t can *occur* and thereby change the current marking. Intuitively, this can be understood as a movement or flow of tokens. As the directions and labelings of the arrows in Fig. 1.1 show, the coin "flows" out of the coin slot, and a cookie packet "flows" into the compartment. Arrows

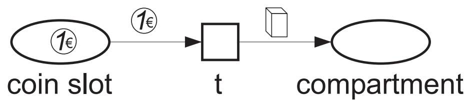


Figure 1.1: The cookie vending machine in its initial state

arc

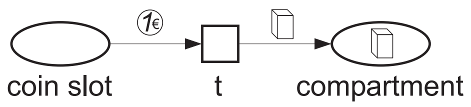
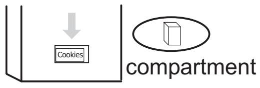


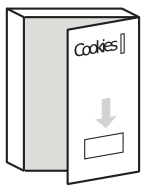
Figure 1.2: The cookie vending machine after the occurrence of t



depict the *arcs* of the net.

Figure 1.2 shows the new marking: The place coin slot does not contain any tokens, while the place compartment contains a cookie packet as a token. In this marking, t cannot occur.

1.2 A Look Inside



If we look inside the machine, we will find several components that store coins and cookie packets and handle the sale. Figure 1.3 in turn models the interior of the vending machine as a Petri net: there is a storage filled with five cookie packets and one – initially empty – cash box.

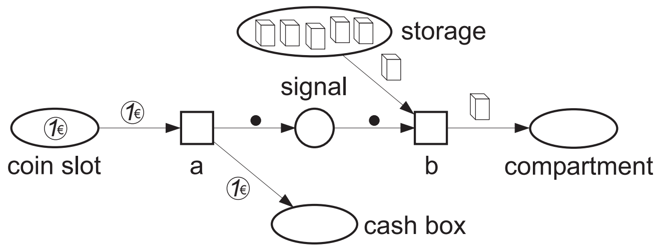


Figure 1.3: A look inside the cookie vending machine

a – collect coin

b – give out cookie packet

In the marking shown, transition a can occur. Its effect can be deduced from the arrows starting or ending in a: the coin disappears from the coin slot and appears in the cash box. Simultaneously, a signal for transition b is generated, depicted by a black dot. Figure 1.4 shows the marking after the occurrence of a.

Now, transition b can occur, because the arrows ending in b are labeled with objects that are actually present in the respective places: a black dot in signal and a cookie packet (even several) in the storage. After the occurrence of b, the marking shown in Fig. 1.5 is reached. It corresponds to the marking in Fig. 1.2. No more transitions can occur now.

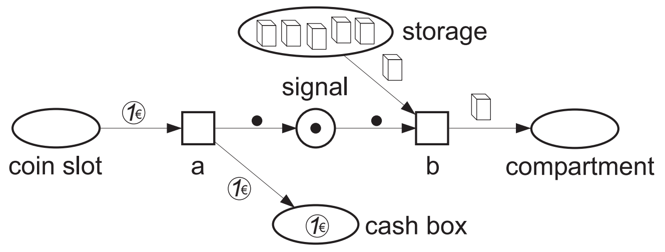


Figure 1.4: After the occurrence of a

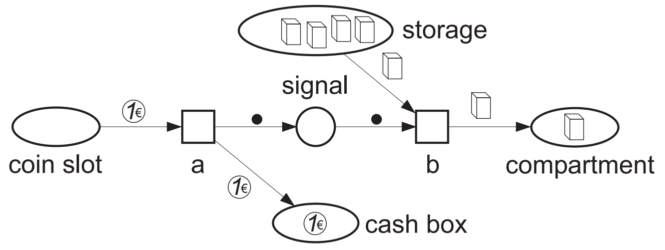
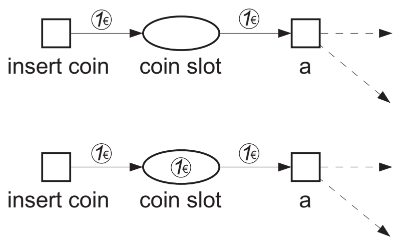


Figure 1.5: After the occurrence of b

1.3 The Interface

So far, we have modeled the vending machine as a *closed system*: coins and cookie packets are distributed across places and the occurrence of transitions redistributes them. What is missing are actions of the environment: someone inserts a coin, for example, or takes out a cookie packet. How do we model this?



As Fig. 1.6 shows, a transition insert coin sits in front of the – empty – coin slot. insert coin does not have any preconditions, so it can occur anytime. In the real world, of course, this action, in fact, does have further preconditions. Most importantly, the environment has to provide a coin. Likewise, the transition take packet models the taking of a packet out of the compartment.

The two transitions insert coin and take packet model the vending machine's *interface*. Both are enabled in the marking of Fig. 1.6. Connected with this is the label "ε", which we will deal with next.

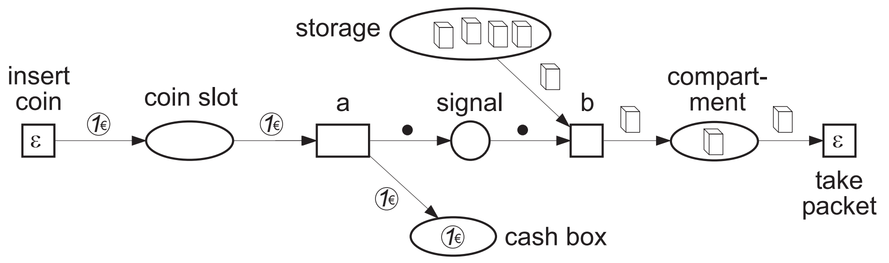


Figure 1.6: The cookie vending machine with the cold transitions of its interface

1.4 Hot and Cold Transitions

So far, we have tacitly assumed that an enabled transition is actually going to occur. For the transitions a and b of the cookie vending machine, this is appropriate: A coin in the coin slot, after all, is supposed to generate a signal, which in turn triggers the transport of a cookie packet. a and b are *hot transitions*. However, insert coin is a different matter. This transition is enabled in Fig. 1.6, for instance. In the real world, a customer triggers this transition by inserting a coin into the coin slot. Whether or not this will ever happen is not guaranteed. Therefore, we want to allow that insert coin stays forever enabled without ever actually occurring. In the model, insert coin is a *cold transition*. The transition take packet is cold as well: no customer is forced to take his paid-for packet out of the compartment.

cold transition

In general, the majority of a system model's transitions are hot. Cold transitions are much rarer. As in our example, cold transitions are often found at the interface to the unmodeled environment. To distinguish them from hot transitions, cold transitions are labeled with "ε" (as an indication of their often "external" character).

1.5 Runs

A *run* of a system model describes how transitions occur consecutively. A typical run w of the model in Fig. 1.6 starts with the insertion of a coin. This generates a signal (via transition

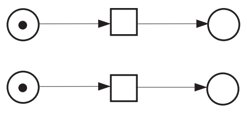
a) which then triggers the release of a packet (transition b).

A run is *complete* if it terminates in a marking that does not enable any hot transitions. The run w described above is complete, because w only enables the cold transitions insert coin and take packet. An initial part of w that enables one of the hot transitions a or b is an *incomplete* run.

The above run w can be extended by take packet to form another – also complete – run. It can also be extended by another occurrence of insert coin. In this case, however, a and b have to occur once more as well. The description "complete" is usually omitted. Incomplete runs are discussed only rarely.

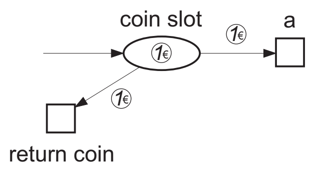
The cookie vending machine in Fig. 1.6 has several more runs apart from the above-mentioned run w. They are formed by the repeated occurrence of the net's transitions.

In the marking shown in Fig. 1.6, the first packet can be taken out of the compartment, the second packet can be dropped into it and the third coin can be inserted.


smallest example for independently occurring transitions

The transitions take packet, b and insert coin all occur independently from each other. How the repeated occurrence of transitions can form one or more runs is discussed in Chap. 4.

1.6 Alternatives



Our cookie vending machine does not accept every coin. Maybe the machine has a coin verifier that rejects faulty coins. Maybe the customer can manually eject the coin from the coin slot by pushing an appropriate button. The vending machine can return a coin to its environment for various reasons. However, we do not want to model all those details. For us, it is sufficient to model that a coin in the coin slot *either* reaches the cash box *or* is returned to the environment. We do not model why each alternative occurs.

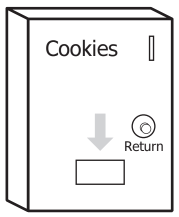


Figure 1.7 adds to the cookie vending machine the option of returning coins. After the insertion of the coin, the marking in Fig. 1.8 is reached. Now, all preconditions for the occurrence of return coin as well as a are met. However, only one of the two will actually occur: if a occurs, the coin slot will

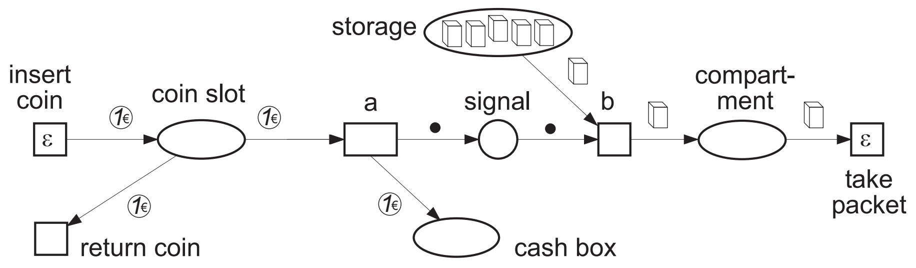


Figure 1.7: The cookie vending machine with the option of returning coins

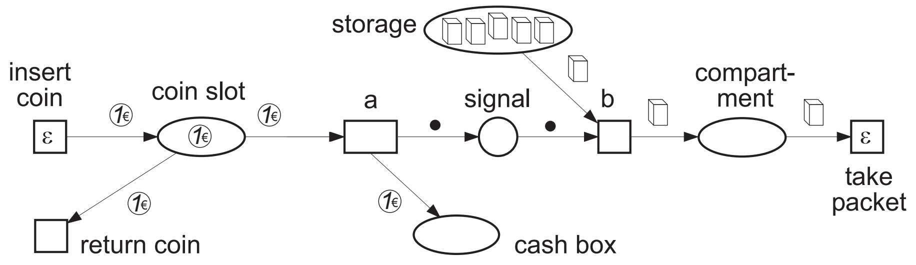
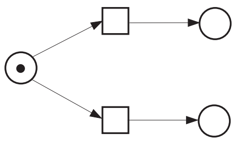


Figure 1.8: Two options: return coin and a



smallest example for a conflict

lose its token, and return coin will no longer be enabled. However, if return coin occurs, a will no longer be enabled. Both transitions are in *conflict* with each other.

1.7 Fine Tuning

After five coins have been inserted and five cookie packets have been given out, the storage is empty and the cash box contains five coins. The sixth inserted coin also reaches the cash box. It generates a signal which cannot be processed, however, because there are no more cookie packets in the storage. Figure 1.9 shows this marking. The sixth customer has paid, but does not get any cookies!

Therefore, we have to prevent a from occurring a sixth time. The sixth coin always has to be returned to the environment. For that purpose, the model is expanded as shown in Fig. 1.10. It now contains an additional counter. The counter is modeled as a place that always contains exactly one token. This token is always a natural number. Initially, this is the number of cookie

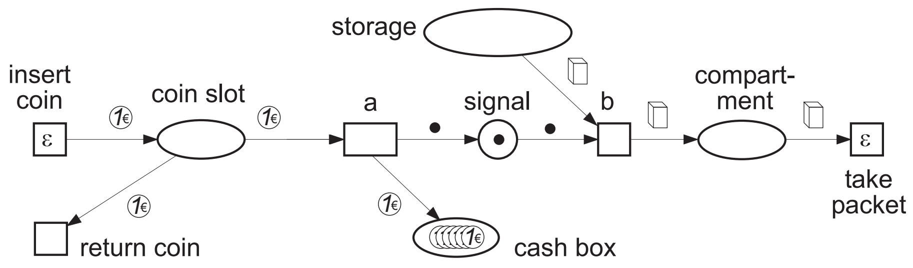
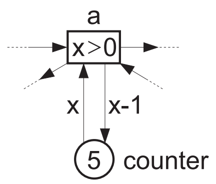


Figure 1.9: Reachable marking of the cookie vending machine



packets in the storage. Every occurrence of the transition a reduces its value by one.

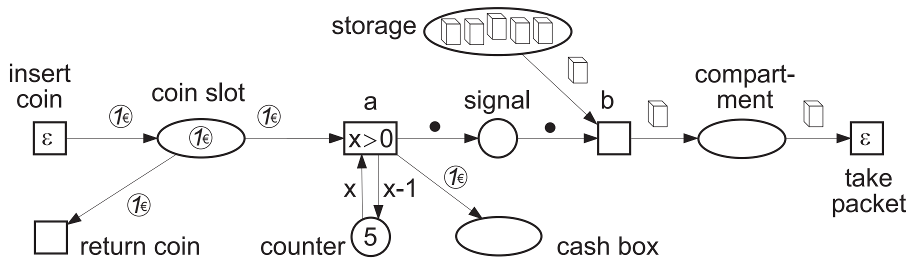


Figure 1.10: Addition of a counter

Technically, we achieve this with the help of a *parameter*, x. The preconditions for the occurrence of a now include an *occurrence mode*, that is, the replacing of x with a concrete value. From the marking shown in Fig. 1.10 with the occurrence mode x=5, the marking in Fig. 1.11 is reached.

The label x>0 in a states an additional precondition for the occurrence of a. It will prevent a from occurring, if the counter holds the token "0".

Finally, we wish to model that the coin slot can contain at most one coin, and that at most one signal is pending at any one time. Figure 1.11 models this by means of two additional places.

With that, we have modeled the essential components of the cookie vending machine as a Petri net. At the chosen level of abstraction, all aspects of the vending machine's structure and behavior are modeled correctly. What "correctness" means

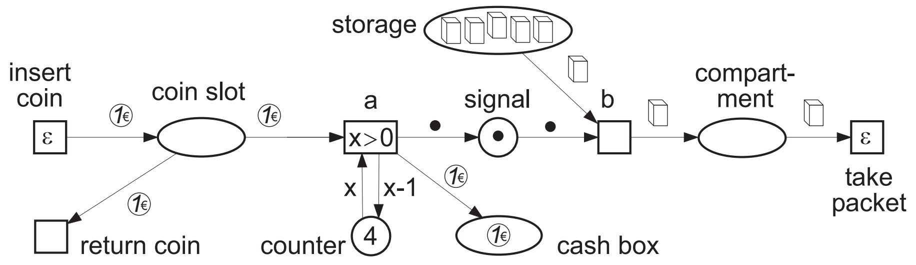


Figure 1.11: After the occurrence of a

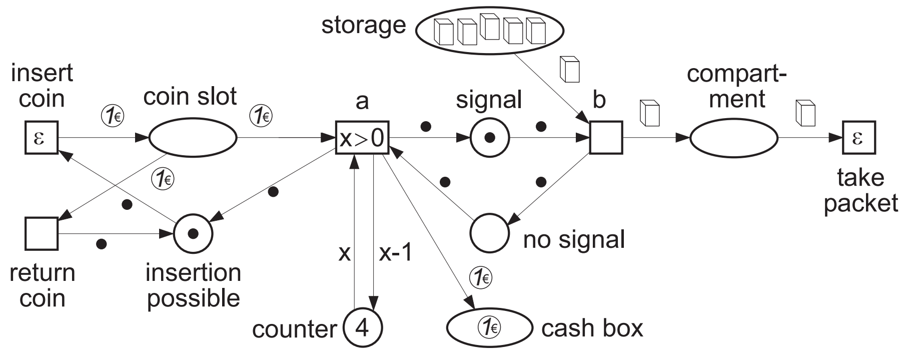


Figure 1.12: At most one coin in the coin slot and at most one signal pending

here and how correctness is verified are covered in Part II of this book.

1.8 Diverse Components

The model of the cookie vending machine shows what makes Petri nets so flexible: we model entirely diverse components that nevertheless can be expressed and combined with a uniform formalism. In the models in Figs. 1.3–1.11, places and their tokens describe:

- components of a real vending machine: coin slot, cash box, storage, compartment, with *coins* and *cookie packets* as tokens;

- technical components: the counter with a *number* as token and signal with a black token representing a pending signal;
- logical abstractions: insertion possible, signal and no signal with black tokens representing the logical value *true*.

Transitions describe:

- actions of the vending machine's interior, modeling the transport of objects and signals: a (accept coin) and b (give out packet);
- actions of the vending machine's interface: return coin;
- customer actions allowed by the vending machine, influencing its behavior: insert coin, take packet.

A global state of the cookie vending machine is modeled as a marking (distribution of tokens across places). One action, that is, a change from one state of the vending machine to a new one, corresponds to a step from one marking to another marking in the model. Several successive actions of the vending machine correspond to a run of the model.

By modeling all these diverse components with a uniform formalism, their reciprocal effects can be identified and analyzed. In that way, the correctness of the entire system and its effects on the real world can be convincingly verified.

Exercises

1. How many reachable markings does the system net in Fig. 1.10 have if the place counter initially holds the token 2 (instead of 5)? Based on this result, estimate whether the number of reachable markings of the unaltered net is greater than or less than 70.

2. In Fig. 1.10, replace the initial marking of the place counter with (a) 4 or (b) 6. Discuss what these variations mean intuitively.

3. Model a gambling machine with the following behavior:

   Initially, the machine is ready for a game and the player can insert a coin. An inserted coin reaches the cash box, and the machine changes into a state in which it pays out a coin from the cash box an arbitrary number of times, which may be zero. At some point, the machine stops giving out coins (at the latest when the cash box is empty) and becomes ready for another game.

Further Reading

Someone who models a system does not always immediately think about components that behave like places and transitions of a Petri net. Usually, a system is first theoretically broken down into abstract components, which are later on systematically refined. All modeling techniques based on Petri nets make use of refinement and composition. In connection with *colored* nets [38], hierarchical concepts are introduced. Girault and Valk [29] also recommend a refining approach in their extensive book on system design with Petri nets. Introductory texts of various kinds can be found in the anthologies of the latest two *Advanced Courses on Petri Nets* [18], [70].

How It Began

In the late 1950s, Carl Adam Petri, at the time a research associate at the Department for Instrumental Mathematics at the University of Bonn, Germany, thought very pragmatically about the implementation of recursive functions. After all, for such functions, it is generally not possible to predict how much space their calculations will consume. If the available resources are insufficient for a calculation, the data processing system should be *expandable*, to continue the calculation. This is more efficient than starting over with a larger system.

So Petri sought a system architecture that can be *expanded indefinitely*. Such an architecture does not have any central components, most of all no central, synchronizing clock, because every expansion enlarges the system in space. Connections to the clock would become longer, and longer cycles would demand lower clocking frequencies. At some point, the limitations of the laws of physics would have to be broken in order to further expand the system. Therefore, such an architecture inevitably has to make do without any synchronizing clocks.

In his famous dissertation “Kommunikation mit Automaten” (Communication with Automata) [56], Petri shows that it is actually possible to construct such an indefinitely expandable, *asynchronous* system architecture. It incorporates locally confined components communicating with each other via local interfaces.

Actions with *locally confined* causes and effects are the central idea of the nets proposed by Petri in [56]. Termed “Petri nets”, they became one of the most popular concepts of computer science. Only later did Petri start to use a graphical representation. Thus, the first text on Petri nets does not contain a single Petri net!

The Basic Concepts — Chapter 2

Now we take a little closer look at Petri nets, that is, at their structure of places, transitions and arcs, the fundamental data structure of multisets, the structure of markings and steps and lastly the reachable markings and the final markings. We explain this with the help of the (slightly modified) cookie vending machine.

2.1 A Variant of the Cookie Vending Machine

Figure 2.1 shows a modified version of the cookie vending machine previously shown in Fig. 1.10 (the denotations A ... H of the places and a ... e of the transitions make the notation easier). In addition to the five rectangular cookie packets, two

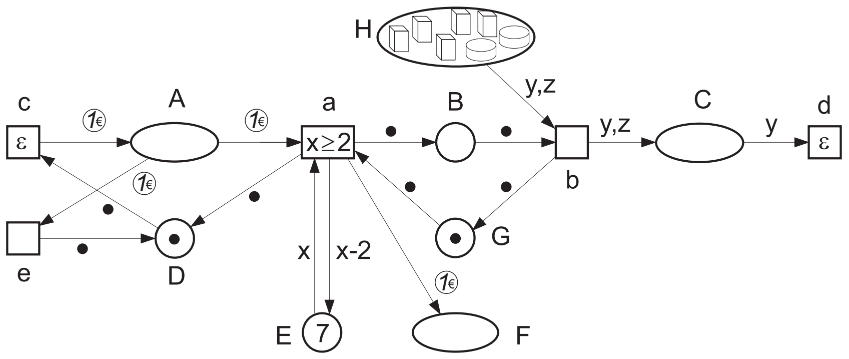


Figure 2.1: Two kinds of packets and giving out two packets at once

two kinds of tokens:

round packets are now in the storage H. The customer receives *two* cookie packets for one euro. The machine decides non-deterministically whether those packets are rectangular or round. Bought packets are dropped into the compartment C. The customer can remove one packet at a time (via the cold transition d). The net in Fig 2.1 will be used as an example throughout Chap. 2.

2.2 Components of a Net

The example of the cookie vending machine shows all the kinds of components that can occur in a Petri net.[1] We will look at them again individually and explain their roles in the model of the system.

Places

A Petri net is a structure with two kinds of elements. One kind of element is *places*. Graphically, a place is represented by a circle or ellipse. A place p always models a *passive* component: p can store, accumulate or show things. A place has discrete states.

Transitions

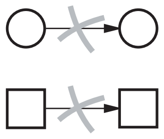
The second kind of elements of a Petri net are *transitions*. Graphically, a transition is represented by a square or rectangle. A transition t always models an *active* component: t can produce things, consume, transport or change them.

Arcs

Places and transitions are connected to each other by directed *arcs*. Graphically, an arc is represented by an arrow. An arc never models a system component, but an abstract, sometimes only notional relation between components such as logical connections, access rights, spatial proximities or immediate linkings.

[1]The literature gives a multitude of extensions and generalizations, which are not covered here.

In the example of the cookie vending machine, it is striking that an arc never connects two places or two transitions. An arc rather runs from a place to a transition or vice versa from a transition to a place. This is neither coincidental nor arbitrary, but inevitably follows if nets are used correctly to model systems, that is, if passive and active components are properly separated.

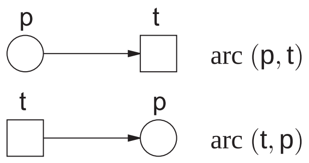


Net Structure

It is customary to denote the sets of places, transitions and arcs with P, T and F, respectively, and to regard arcs as pairs, that is, F as a relation $F \subseteq (P \times T) \cup (T \times P)$. Then

$$N = (P, T, F)$$

is a *net structure*. The places and transitions are the *elements* of N. F is the *flow relation* of N. Figure 2.2 shows the net structure of the cookie vending machine as shown in Figs. 1.10, 1.11 and 2.1.



net structure

place

transition

arc

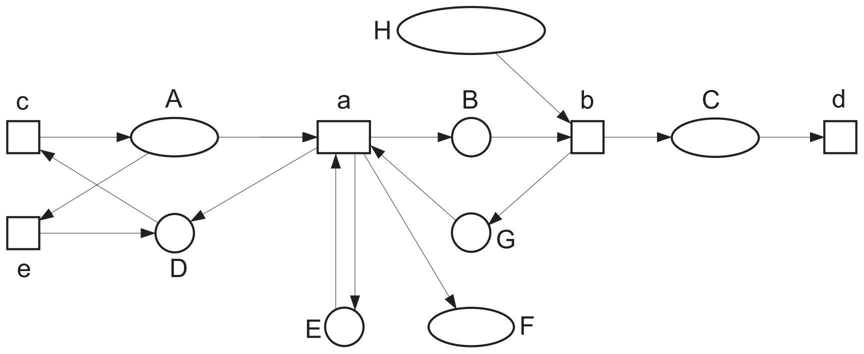
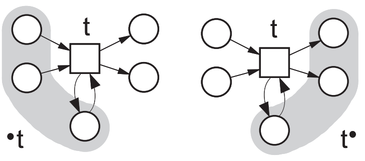


Figure 2.2: The net structure of the cookie vending machine



If a given context unambiguously identifies a net N, the *pre-set* $^\bullet x$ and *post-set* $x^\bullet$ of an element x are defined as

$$^\bullet x =_{\text{def}} \{y \mid yFx\} \quad \text{and}$$
$$x^\bullet =_{\text{def}} \{y \mid xFy\}.$$

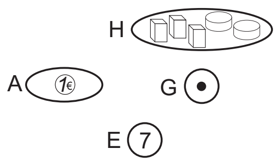
pre-set $^\bullet x$ post-set $x^\bullet$

loop

Two elements x, y of N form a *loop* if $x \in {}^\bullet y$ and $y \in {}^\bullet x$. For instance, a and E in Fig. 2.2 form a loop.

Markings

tokens in places:

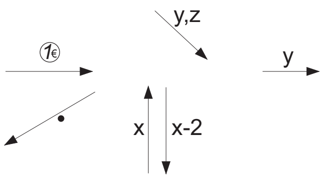


A *marking* is a distribution of tokens across places. A marking can be represented graphically by symbols serving as tokens in the respective circles and ellipses. For a system with an initial state, the initial marking is often depicted in this way. The symbolic tokens (for instance, 1€, ▯, 7) generally denote elements of the real world. This correlation is so strong that we do not distinguish between the symbolic representation and the real elements that they denote.

Next to symbolic tokens, abstract black tokens often occur, for instance, in the places D and G in Fig. 2.1. Such a token often indicates that a certain condition (modeled as a place) is met. It is also possible, and common, to represent concrete elements not by symbolic but by abstract black tokens. *Elementary* nets only utilize black tokens.

Labelings of Arcs and Transitions

labelings of arcs:



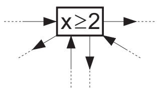
Arcs and transitions can be labeled with *expressions*. Next to elements of the real world, which have occurred in markings before, functions (for instance, a subtraction) and variables (for instance, x and y) can occur in such expressions. These expressions have a central property: if all variables in an expression are replaced by elements, it becomes possible to evaluate the expression in order to obtain yet another element. It is convenient to write the labeling of an arc (p, t) or (t, p) as

labeling

$$\overline{pt} \text{ or } \overline{tp} \qquad (1)$$

respectively. Statement (1) describes the tokens that "flow through the arc" at the occurrence of t.

labeling of a transition:



The variables in these expressions are *parameters* describing different instances ("modes") of a transition. Such a transition can only occur if its labeling evaluates to the logical value "true". The rest of this chapter describes this correlation in more detail.

2.3 The Data Structure for Petri Nets: Multisets

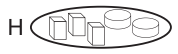
In a Petri net, the tokens of a place often represent objects that we usually do not want to distinguish. For instance, we are only interested in the number of coins in the cash box (although we could, for instance, distinguish them by their date of coining).

In general, examples of different *kinds of tokens* are mixed in a place, e.g., rectangular and round cookie packets in the storage H. They form a *multiset* a, formally a mapping

multiset

$$a : U \to \mathbb{N}$$

that maps every kind u of a *universe* U to the number of its occurrences in a.



a of H:

$$\begin{aligned} a(▯) &= 3 \\ a(⬭) &= 2 \\ a(u) &= 0, \quad \text{for any other } u \end{aligned}$$

We always assume a “sufficiently large” universe U that contains all examined kinds of tokens. We write the set of all multisets over U as

$$\mathcal{M}(U) \text{ or } \mathcal{M} \text{ for short}$$

set $\mathcal{M}(\mathcal{U})$ of all multisets

if the context unambiguously identifies the universe U.

The universe U can contain an infinite number of elements, for instance, all natural numbers. A multiset a over U can map the value $a(u) = 0$ to almost all $u \in U$. That means that u does not occur in a. Thus, a is *finite* if

the universe of the cookie vending machine:

▯, ⬭, 1€, 0, 1, 2, 3, 4, 5, 6, 7, ●

finite multiset

$$a(u) \neq 0 \text{ for only a finite number of } u \in U.$$

We write a finite multiset a with its multiple elements in square brackets $[\ldots]$. Consequently, the *empty* multiset is denoted by $[\,]$:

finite multiset:

$[▯, ▯, ▯, ⬭, ⬭]$

$a(▯) = 3, a(⬭) = 2$

$$[\,](u) = 0 \text{ for each } u \in U.$$

empty multiset $[\,]$

Multisets $a, b \in \mathcal{M}$ can be added: for each $u \in U$, let

sum of multisets

$$(a + b)(u) =_{\text{def}} a(u) + b(u).$$

They can be compared:

order on multisets

$$a \leq b \text{ iff for each } u \in U : a(u) \leq b(u),$$

subtraction on multisets

and b can be subtracted from a if $b \leq a$:

$$(a-b)(u) =_{\text{def}} a(u) - b(u).$$

With these notations, we describe dynamic behavior.

arithmetic operations on multisets:

[▯, ⬭] + [▯] = [▯, ▯, ⬭]

[▯, ⬭] ≤ [▯, ▯, ⬭]

$a \leq a$

[▯, ⬭, ⬭] − [⬭] = [▯, ⬭]

$a - a = [\,]$

The tokens of a place p usually belong to a *type*, that is, to a (small) subset of the universe U. In Fig. 2.1, for instance, only coins lie in the places A and F, only black tokens in D and G, only cookie packets in H and C and only numbers in E. If a place p only holds tokens of type τ, the type τ is assigned to the place p.

2.4 Markings as Multisets

marking

Initial marking M_0 of Fig. 2.1:

$M_0(\mathsf{H}) =$ [▯, ▯, ▯, ▯, ▯, ⬭, ⬭]

$M_0(\mathsf{A}) = M_0(\mathsf{B}) = M_0(\mathsf{C}) = [\,]$

$M_0(\mathsf{D}) = M_0(\mathsf{G}) = [\bullet]$

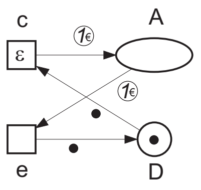
$M_0(\mathsf{E}) = [7]$

Now we can precisely define the term *marking*: A marking M of a net structure (P, T, F) is a mapping

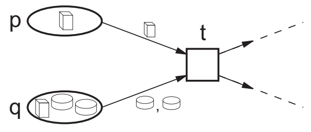
$$M : P \to \mathcal{M}.$$

That means that M maps every place p to a multiset $M(p)$. As explained in Sect. 2.2, a marking M describes a *state* of the modeled system. Given the significance of a system's initial state, the initial marking (usually denoted M_0) is often drawn into the respective net structure.

2.5 Steps with Constant Arc Labelings



Let us now examine the special case in which the arcs around a transition t are labeled with individual elements of a universe. This applies to the arcs around the transitions c and e in Fig. 2.1. In general, however, an arc is labeled with more than one element. Formally, this is a multiset, whose brackets [and] are not written in order to save space. Thus, with the notation of (1), for each arc (p, t) or (t, p) the following holds:



$$\overline{pt} \in \mathcal{M} \quad \text{and} \quad \overline{tp} \in \mathcal{M}.$$

We technically expand this notation for all places p and transitions t by

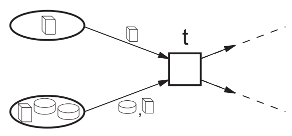
$$\overline{pt} = [\,] \quad \text{and} \quad \overline{tp} = [\,]$$

$\overline{\mathsf{pt}}$ = [▯]

$\overline{\mathsf{qt}}$ = [⬭, ⬭]

if no arc (p, t) or (t, p) exists, respectively.

A transition t can *occur* in a marking M if the related preconditions are met, that is if M *enables* the transition t.



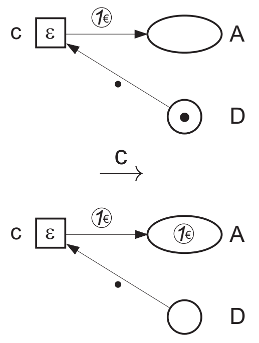
t is enabled

As in many other system models, we separate the *enabling* of t from the effect of the *occurrence* of t. Whether a marking M enables a transition t depends on the labelings of the arcs ending in t. M *enables* t if and only if

$$M(p) \geq \overline{pt}$$

for each arc (p, t). Thus, the initial marking of the cookie vending machine only enables the transition c.

the first step of the cookie vending machine:



If a marking M enables a transition t, it results in the *step*

$$M \xrightarrow{t} M',$$

in which the marking M' of each place p is defined as

$$M'(p) =_{\text{def}} M(p) - \overline{pt} + \overline{tp}.$$

formal:

$M_0 \xrightarrow{c} M_1$
$M_1(\mathsf{D}) = [\,], M_1(\mathsf{A}) = [\text{ 1€ }]$
$M_1(p) = M_0(p),$ for any other place p

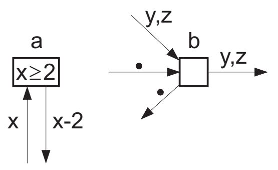
2.6 Steps with Variable Arc Labelings

An arc or a transition can be labeled with an expression a that contains variables. By assigning values to the variables in a, the expression a can be evaluated. If a is written onto an arc, the result is a multiset. If a is written into a transition, the result is either *"true"* or *"false"*. In order to calculate these values, the labelings of all arcs that end or start at a transition (the arcs *around* t) have to be taken into account simultaneously.

expression
transition condition
arc labeling

Put a little more technically: Let $x_1, \ldots, x_n$ be the variables of the arc labelings around a transition t. Let $u_1, \ldots, u_n$ be elements of the universe. Then

$$\beta : (x_1 = u_1, x_2 = u_2, \ldots, x_n = u_n)$$

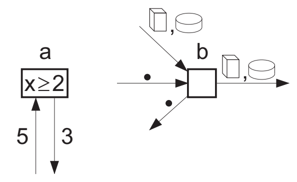


is a *mode* of t. In Fig. 2.1, for instance, the variables y and z occur in the arc labelings around the transition b. Thus, β_1 : (y = ▯, z = ⬭) is a mode of b. The transition b has three additional modes: β_2 : (y = ⬭, z = ▯), β_3 : (y = z = ⬭) and β_4 : (y = z = ▯). A mode β of a transition t creates

mode of a transition

mode β of the transitions a and b:
$\beta(\mathsf{x}) = 5, \beta(\mathsf{y}) = $ □, $\beta(\mathsf{z}) = $ ⬭

results in the constant arc labelings



for each arc (p,t) or (t,p) a multiset $\beta(p,t)$ or $\beta(t,p)$, respectively. Thus, in Fig. 2.1, $\beta_1(\mathsf{H},\mathsf{b}) = \beta_2(\mathsf{H},\mathsf{b}) = [$□, ⬭$]$. Another example is $\beta : (\mathsf{x} = 7)$, a mode of the transition t in Fig. 2.1, and it holds: $\beta(\mathsf{a},\mathsf{E}) = [7-2] = [5]$.

If $\overline{pt}$ does not contain any variables, then obviously

$$\beta(p,t) = \overline{pt}.$$

A transition t can itself have a labeling that contains variables. An example is the labeling $\mathsf{x} \geq 2$ of transition a in Fig. 2.1. For such a labeling i, a mode β of t creates a logical value, $\beta(i)$. For instance, for the labeling $\mathsf{x} \geq 2$ of a, the mode $\beta_1 : (\mathsf{x} = 7)$ creates the logical value $\beta_1(\mathsf{x} \geq 2) = [7 \geq 2] = true$.

Thus, a mode β of t creates multisets at the arcs around t. *A step of t in the mode β* is then defined as described in the previous section. Additionally, the labeling a of t has to evaluate to $\beta(a) = true$. For a step from M to M' via t in the mode β, we write

step

$$M \xrightarrow{t,\beta} M'.$$

The symbol β for the mode is often omitted and we write, for instance

$$\mathsf{x} = 5 \text{ instead of } \beta : (\mathsf{x} = 5).$$

M enables t in mode β

Put in a formal context, a marking M enables a transition t in the mode β of t if for each arch in the form (p,t):

$$M(p) \geq \beta(p,t)$$

and for the labeling i of t:

$$\beta(i) = true.$$

This then results in the step $M \xrightarrow{t,\beta} M'$, in which M' for each place p is defined by

$$M'(p) = M(p) - \beta(p,t) + \beta(t,p).$$

Again, let $\beta(p,t) = [\,]$ and $\beta(t,p) = [\,]$ if no arc (p,t) or (t,p) exists in N, respectively.

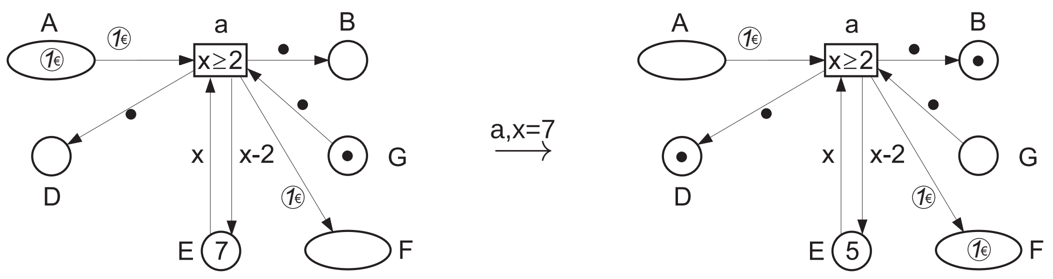
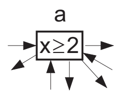


Figure 2.3: Step $M_1 \xrightarrow{\text{a,x=7}} M_2$

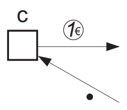
Consider the cookie vending machine in Fig. 2.1: After the step $M_0 \xrightarrow{\text{c}} M_1$, the marking M_1 enables the transition a in the mode x $= 7$, and in no other mode. Figure 2.3 shows the effect that the step

$$M_1 \xrightarrow{\text{a,x=7}} M_2$$

has on the surroundings of the transition a. The marking M_2, which is then reached, enables the transition b, because now, values can be assigned to the variables y and z. Every assignment of ▯ or ⬭ to these variables enables b. Thus, there exists a selection of four modes and hence four steps in M_2, as outlined in Fig. 2.4.



for $\beta($x$) = 7$: condition x ≥ 2 is met!



c has only one mode

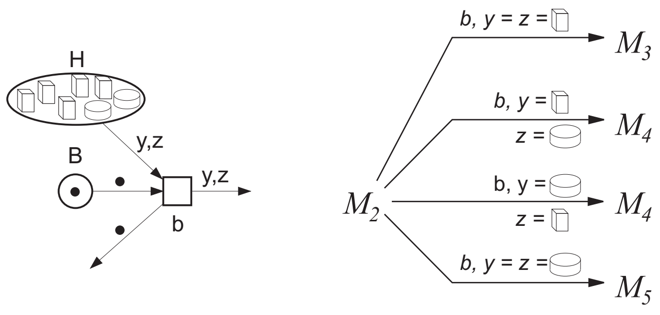


Figure 2.4: M_2 enables b in four modes

2.7 System Nets

We have now assembled the principal notations that enable us to describe a discrete, dynamic system, as for instance a cookie vending machine. According to the principles in Section 2.2, we use an appropriately labeled, finite net structure, N, to do this. A central term is that of *a marking of* N, that is, a distribution of tokens (multisets) across the places of N. Typically, the *initial marking* of N is denoted by M_0 and is explicitly drawn into N. M_0 describes the initial state of the modeled system. A transition t can be labeled with a *condition* and the arcs around t with *expressions*. These labelings show the various situations (*modes*) in which t is enabled, and the respective effects at the occurrence of t. Lastly, every transition is either *hot* or *cold*, where cold transitions are indicated by “ε”.

cold transition

A net structure together with an initial marking, transition conditions, arc labelings and cold transitions form a *system net*.

system net

System nets are used to model real, discretely changeable systems. Each place of a system net models a state component of the system and each currently existing token in a place models a currently given, but changeable, characteristic of that component. Each transition of a system net represents an action of the system. The occurrence of a transition describes the occurrence of the respective action. If, in doing so, a token reaches or leaves a place, the action respectively creates or terminates the corresponding characteristic of the state component.

2.8 Marking Graph

reachable marking

For a system net N and an initial marking M_0, a marking M of N is *reachable* if there exists a sequence of steps

$$M_0 \xrightarrow{t_1,\beta_1} M_1 \xrightarrow{t_2,\beta_2} \dots \xrightarrow{t_n,\beta_n} M_n$$

marking graph

with $M_n = M$. In general, infinitely many markings of N are reachable. The reachable markings and steps of a system net N can be compiled into the *marking graph of* N. Its nodes are the reachable markings, its edges the steps between the reachable markings of N. The initial marking M_0 of N is specifically highlighted. The marking graph is also often called the *reachability graph*. Figure 2.5 shows an initial part

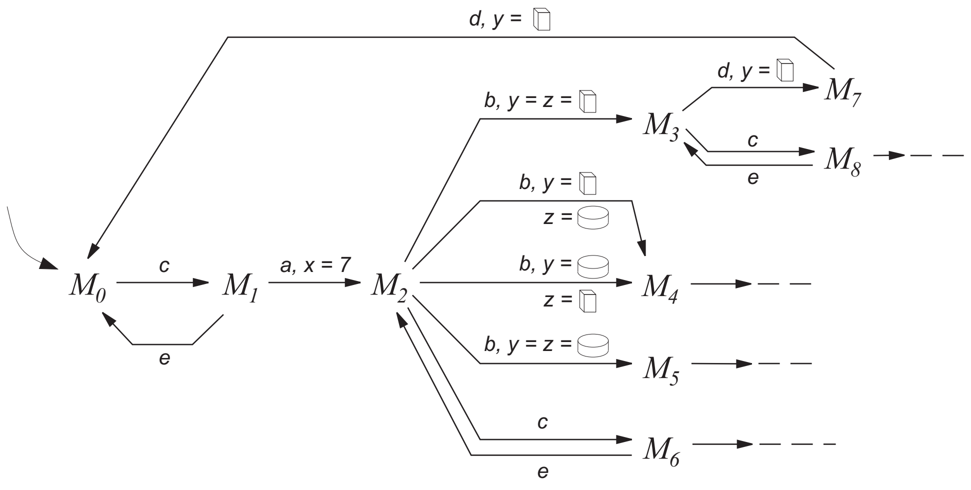


Figure 2.5: Initial part of the marking graph for the system net in Fig. 2.1

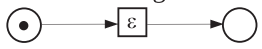
of the marking graph for the system net in Fig. 2.1. The complete marking graph has approximately 100 nodes. In contrast to the system net in Fig. 2.1, it would be extremely laborious and counterintuitive to use the marking graph as a model for the cookie vending machine. In principle, the marking graph of a system net is a suitable starting point for its (automated) analysis, as long as only a finite number of markings are reachable.

2.9 Final Markings

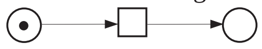
A system has reached a final state if it can remain in this state forever. The marking of the system net in Fig. 1.9 models such a state, in contrast to Fig. 1.10 and Fig. 1.11. A final state of a system corresponds to a *final marking*. In such a marking, no hot transitions are enabled. For instance, in the system net in Fig. 2.1, the initial marking is, at the same time, also a final marking (it only enables the cold transition c).

final marking

a final marking:



not a final marking:



Exercises

1. The system net in Fig. 2.6 expands the cookie vending machine in Fig. 2.1 by a transition f. In your own words, describe the effect and the function of f inside the cookie vending machine.

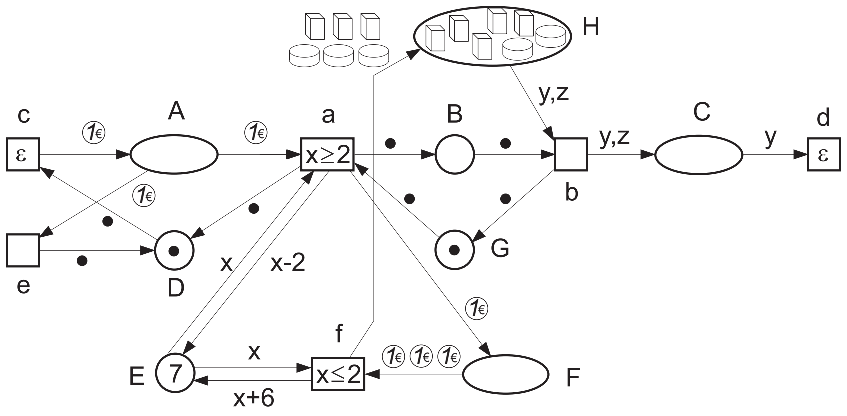


Figure 2.6: Expansion of the cookie vending machine in Fig. 2.1 by a transition f

2. Which of the markings $M_0, \ldots, M_8$ in the marking graph in Fig. 2.5 are final markings?

Further Reading

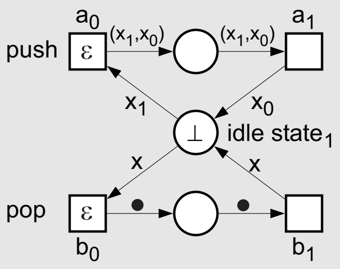
In the first two chapters of this book, we break with tradition in introducing the field of Petri nets. Usually, one begins with the technically simple case of a single kind of "black" token, a case we will not cover until the next chapter. Instead we have immediately introduced "individual" tokens, because they are intuitively more comprehensible, more realistic and more accurate. The price for this is a complex step rule. For such nets, the literature gives many more, ultimately equally expressive, representations. Widely used is a version of Petri nets called *colored nets* [38]. They emphasize the semantics of functions over multisets. Girault and Valk [29] also use such nets.

As is often traditional in mathematics, we do not always distinguish objects and functions from their symbolic representations in expressions, equations, etc. Our distinction between hot and cold transitions is found only sporadically in the literature on Petri nets. Damm and Harel used it in *Live Sequence Charts* [35] as a very apt way of expressing system specifications.

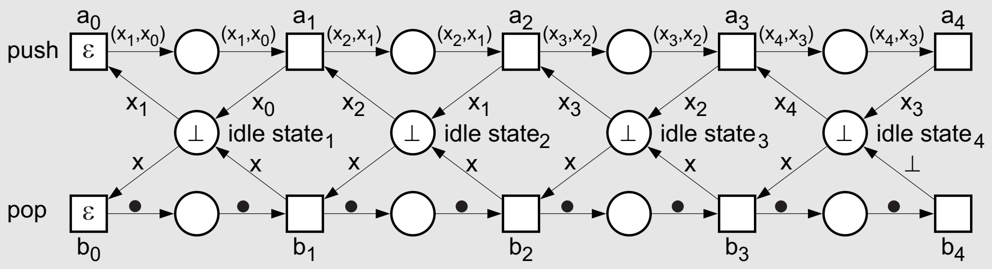
In the historical development, Genrich and Lautenbach [28] introduced nets with *individual* tokens as *predicate/transition nets* and in doing so emphasized the connection with logic.

A Universal, Expandable Architecture

In his dissertation, Petri designed a universal computer architecture that can be expanded an arbitrary number of times. We will explain this idea using the example of a finite, but infinitely expandable stack. It consists of a sequence $A_0 \dots A_n$ of *modules*, where each module A_i $(i = 0, \dots, n)$ has an *idle state* that stores either a value or – e.g., initially – a "dummy" $\perp$. The net



shows the module A_0 with its interface to the environment: The transition push accepts a value from the environment via the variable x_0, which may hold any value. The variable x_1 holds the previously stored value, which is passed on to the module A_1 via the transition a_1. The transition pop extracts the value stored in idle state. Via b_1 the module then receives the value stored in A_1. The net

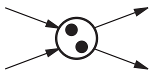


combines four modules to form a stack. Each module A_i behaves according to the pattern described for A_0. Each occurrence of push or pop triggers a wave that moves from left to right through the stack. It ends in A_4 by popping out the previously stored value or pushing in a "dummy" $\perp$ respectively. The transitions a_4 and b_4 are the extension points where another module A_5 can be attached.

Common Special Case: Elementary System Nets — Chapter 3

Petri nets can be used to describe how the control flow and data flow of a distributed algorithm or system interact. However, one often merely wants to express where a control flow currently stands, whether resources are available, how many messages are pending, etc.

For such an abstract view, it is not necessary to distinguish several kinds of tokens: only "black dots" are used as tokens (such tokens already occurred in the cookie vending machine).



Whenever a transition occurs, "exactly one black token flows through each adjacent arc". Because this does not have to be explicitly stated, the arcs do not have any labelings. Such system nets are called *elementary*.

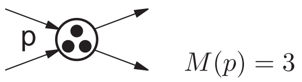
Important aspects of distributed and reactive systems can be modeled appropriately with elementary system nets. We will show this with three examples: an abstract variant of the cookie vending machine, the problem of mutual exclusion, and the crosstalk algorithm.

3.1 Elementary System Nets

elementary system net

An *elementary system net* N consists of a net structure $N = (P, T, F)$ with finite sets P and T of places and transitions, and an initial marking, $M_0 : P \to \mathbb{N}$. Some transitions are marked as *cold* (via the inscription ε). Each marking has the form

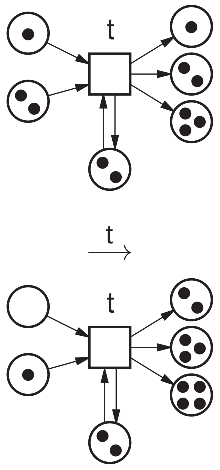
$$M : P \to \mathbb{N}.$$



Each place p holds $M(p)$ tokens.

$M(p) \in \mathbb{N}$ *not* to be confused with $M(p) : U \to \mathbb{N}$ for generic system nets!

The step rule of elementary system nets follows from the step rule of generic system nets: A marking M *enables* a transition t if $M(p) \geq 1$ for each place $p \in {}^\bullet t$. For a step $M \xrightarrow{t} M'$



the following holds:

$$M'(p) = \begin{cases} M(p) - 1 & \text{if } p \in {}^\bullet t \text{ and } p \notin t^\bullet \\ M(p) + 1 & \text{if } p \in t^\bullet \text{ and } p \notin {}^\bullet t \\ M(p) & \text{otherwise} \end{cases}$$

For an elementary system net N it follows from Sects. 2.8 and 2.9 that:

- the step sequences

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_n} M_n$$

  that start with the initial marking M_0 contain the *reachable* markings and steps,

- the *marking graph* of N has as nodes the reachable markings and as edges the reachable steps of N,

- a *final marking* of N does not enable any hot transitions.

3.2 An Abstract Model of the Cookie Vending Machine

Figure 3.1 shows an abstract variant of the model of the cookie vending machine in Fig. 1.11: it does not model euro coins and cookie packets anymore, only their respective amounts. Each

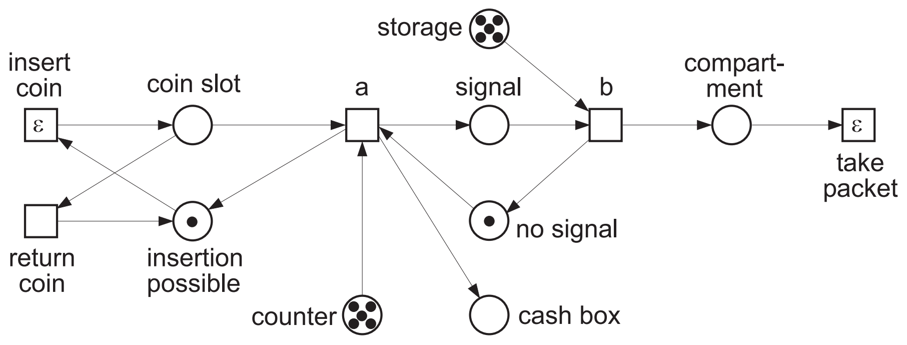


Figure 3.1: Abstract variant of Fig. 1.11

coin and each packet is represented by a black token, just like a pending (or not pending) signal, or the possibility to insert a coin. The current value of the counter is represented by the corresponding amount of black tokens.

This kind of abstraction of concrete objects simplifies the formal analysis of the model.

3.3 Mutual Exclusion

When it comes to distributed algorithms, the synchronization of actions is often crucial: an action can occur if certain conditions are met or certain local states are reached. It is reasonable to model such a condition or state as a place. This place contains a (black) token if and only if the respective condition is met or the respective local state is reached. Models of mutual exclusion are typical examples of this.

In the field of distributed systems, the problem of mutual exclusion manifests itself in a variety of contexts. The simplest case deals with the interaction of two processes and a (scarce) resource that each process occasionally uses. Examples are software processes occasionally using a printer, or vehicles occasionally passing through a traffic bottleneck. At the core of this lies the demand that the two processes never use the scarce resource at the same time.

Figure 3.2 shows the essential components: Each of the two processes *l* and *r* (denoting *left* and *right*) can cycle through three states, local, waiting and critical. In its local state, a pro-

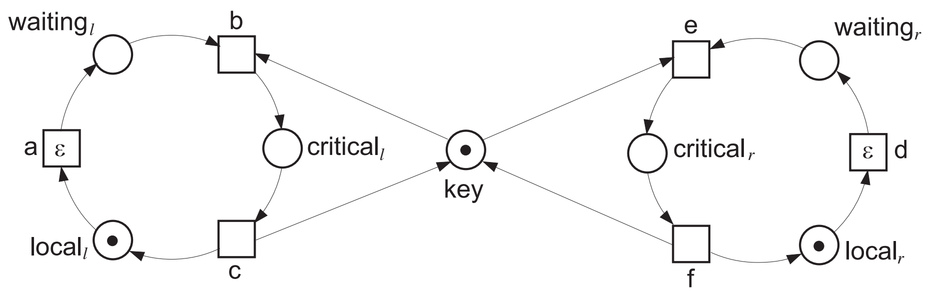


Figure 3.2: Mutual exclusion

cess works without the scarce resource. Via the step to waiting (transition a or d, respectively), the process announces its interest in the scarce resource.

The transitions a and d are cold. Hence, a process is not obliged to execute this step. From some point forward, it may only work locally. The step from waiting to critical (transition b or e, respectively) additionally requires a token in key. The purpose of this is obvious: in its critical state, a process, for instance *l*, uses the scarce resource. *l* only reaches this state (via b) by means of the key. However, then the other process (in this case *r*) cannot reach its critical state via e, because the key is not available anymore. The process *l* does not return the key until *l* leaves its critical state via c.

Between b and e, a conflict can arise an infinite number of times. No strategy has been modeled to solve this conflict. Because of this, all conceivable occurrence frequencies and sequences are possible. In an extreme case, the conflict will always be resolved in favor of the same process (for instance, in favor of *l*). Then the other process (in this case *r*) never reaches its critical state and instead remains waiting forever. In general, one wants to guarantee that every waiting process will eventually become critical. This problem is covered in Chapter 20.

Figure 3.3 shows a solution that is not very convincing: starting with *r*, both processes can alternately reach their critical state. By remaining in its local state, one process can thus prevent the other from repeatedly reaching its critical state.

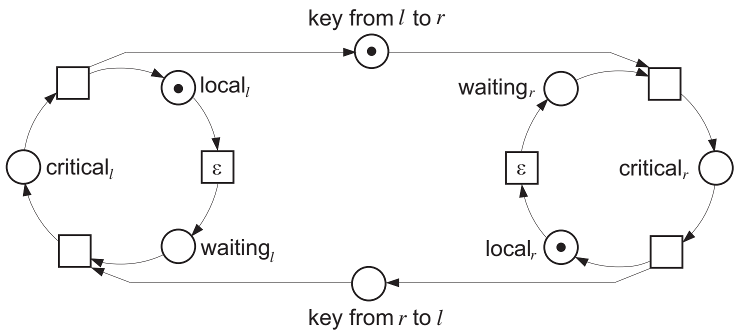


Figure 3.3: Processes reach their critical states alternately

3.4 The Crosstalk Algorithm

The *crosstalk algorithm* assumes two *agents* communicating with each other via a technical channel. The channel works reliably as long as it transmits only *one* message. However, as soon as both agents' messages collide on the channel, *crosstalk* occurs, which may result in corrupted messages reaching their respective recipients. Since both agents can send messages independently from one another, crosstalk cannot be prevented. However, the algorithm allows both agents to recognize crosstalk. In that case they could, for instance, repeat their messages in a predetermined order. We will now introduce an algorithm that works in *cycles*: during each cycle, either one agent successfully sends a message to the other, or both agents recognize crosstalk.

Chapter 5 will show that *scenarios* structure the algorithm and increase its comprehensibility. Chapter 12 will contain a proof that both agents correctly implement the cyclic behavior.

Messages from Left to Right

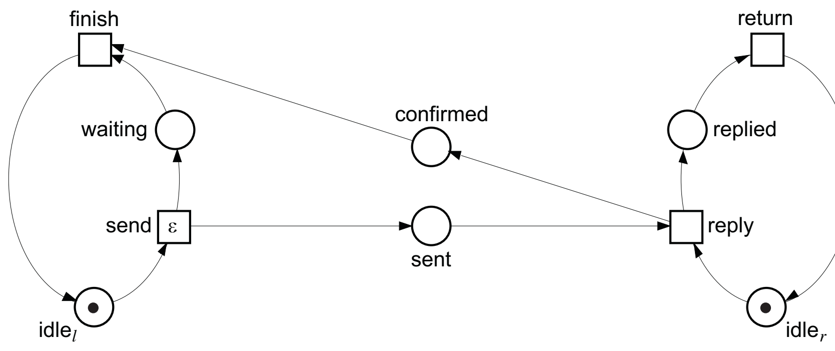


Figure 3.4: Sender and recipient

Figure 3.4 shows how the *left* agent can leave its initial state idle$_l$ via send. In doing so, it sends a message (sent) and enters a waiting state. After having received a confirmation (confirmed), it ends its cycle via finish.

The right agent leaves its initial state idle$_r$ via reply and in doing so confirms (confirmed) the message sent by the left agent. It ends its cycle via return.

Messages in Both Directions

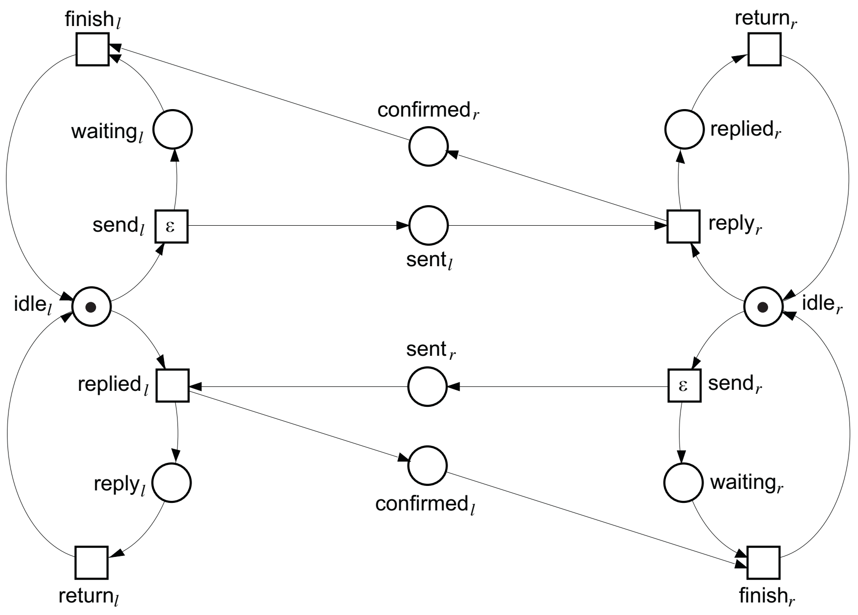


Figure 3.5: Symmetrical complement: deadlock possible

We now complement the system symmetrically so that the right agent can also send a message to the left agent. Figure 3.5 shows this symmetrical complement. However, the system enters a deadlock (both agents waiting) if *both* agents send a message during the same cycle.

Luckily, each agent can recognize a deadlock: it expects a confirmation (confirmed) but receives a message (sent). Therefore, each agent can resolve the deadlock via an additional transition crosstalk, as Fig. 3.6 shows.

Preventing Errors

The system in Fig. 3.6 is still faulty. It is, for instance, possible for the left agent to send a message (send$_l$), which is correctly

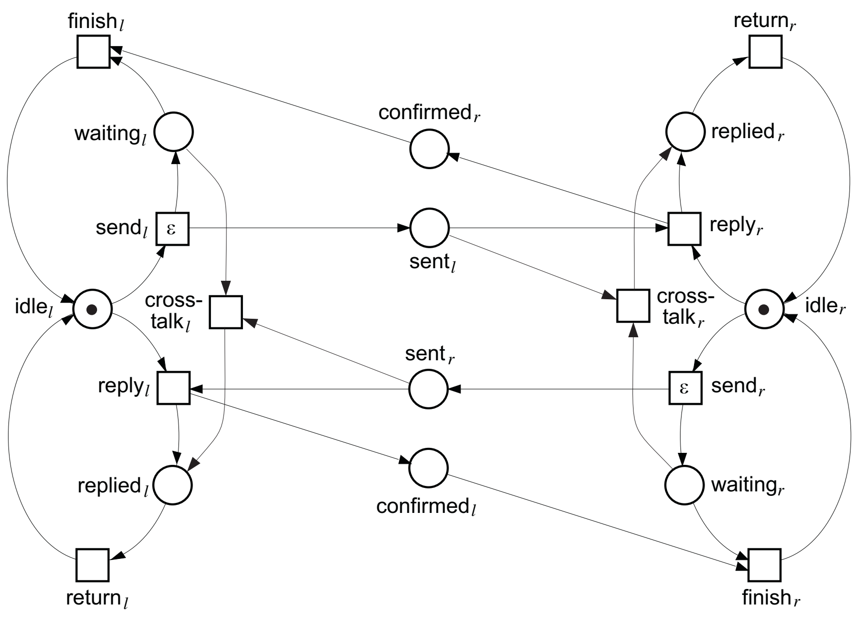


Figure 3.6: With crosstalk: errors possible

received and confirmed by the right agent via reply$_r$. The right agent then returns to its initial state and sends a message via send$_r$. Now the left agent finds a confirmation (confirmed$_r$) alongside a new message (sent$_r$), which it cannot identify as already belonging to the next turn. Instead, it falsely recognizes crosstalk.

Figure 3.7 solves the problem by adding another message type (finished). This system is actually correct – what that means exactly is covered in Chapter 12.

Figure 3.7 shows send$_l$ and send$_r$ as cold transitions: no agent is obliged to send a message. A sent message, however, has to be processed completely. If the system terminates, both agents have returned to their respective initial state idle.

3.5 1-Bounded Elementary System Nets

Except for the net in Fig. 3.1, all elementary system nets in this chapter can only reach markings in which each place holds at most one token. Also, it is often convenient to conceive of

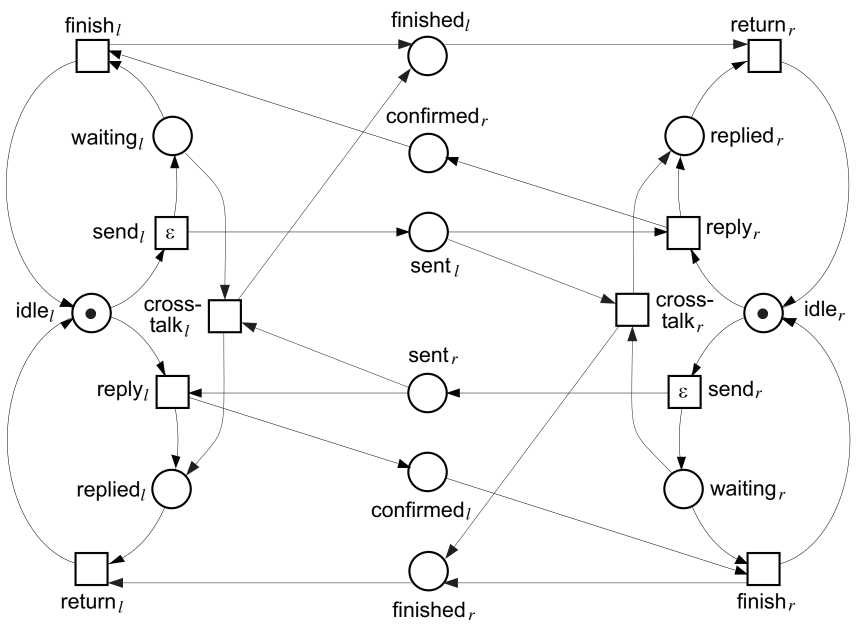


Figure 3.7: The crosstalk algorithm

a token as "moving" through a part of the net, for instance through the three local states of the left (or right) process in the system of mutual exclusion. Such system nets have special properties and analysis methods, which we will cover later on. These nets are so important that they deserve a special name: an elementary system net N is called *1-bounded* if for each reachable marking M and each place p of N:

1-bounded elementary system net

$$M(p) \leq 1.$$

The elementary system nets in Figures 3.2 through 3.7 are all 1-bounded; the system net in Fig. 3.1, obviously, is not.

A marking M of a 1-bounded elementary system net can be described as a string of marked places. For instance, ADE (or DAE, DEA, etc.) describes the initial marking of the system of mutual exclusion according to the notation in Fig. 3.8. Two steps begin with this marking: ADE $\xrightarrow{c}$ BDE and ADE $\xrightarrow{d}$ ADF.

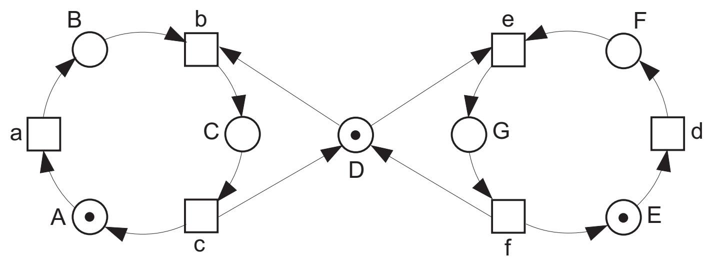


Figure 3.8: Mutual exclusion with symbolic denotations

Exercises

1. Add to the system net in Fig. 3.2 a place noncritical$_l$ that holds a token if and only if process l is not in the state critical$_l$.

2. Construct the marking graphs for the system of mutual exclusion and the crosstalk algorithm using the denotations given in Figs. 3.8 and 3.9.

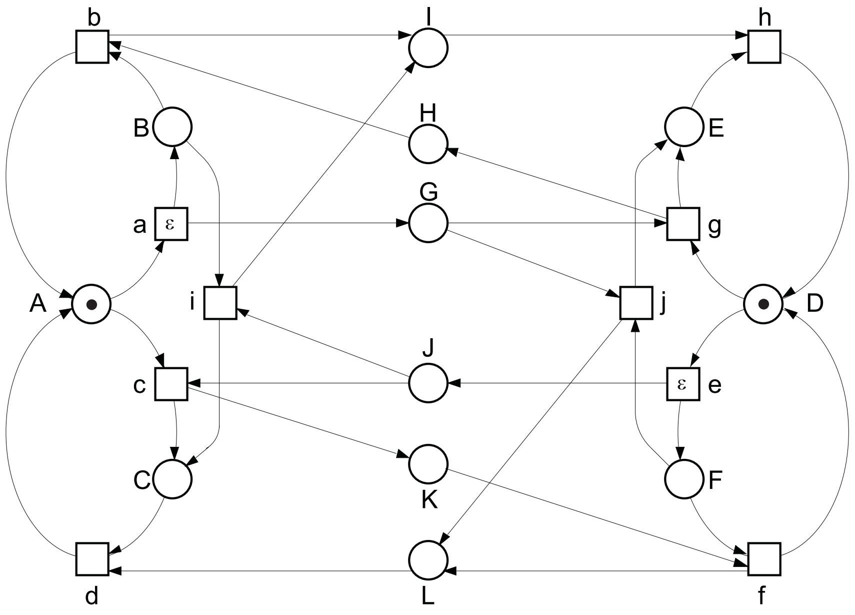


Figure 3.9: The crosstalk algortihm from Fig. 3.8 with symbolic denotations

Further Reading

In many, especially older, publications, elementary system nets – treated as “the Petri nets” – constitute the central formalism. Much attention is paid to places that accumulate unboundedly many tokens. After the *Third Advanced Course on Petri Nets* in 1998, a team of authors compiled a set of lectures on the rich theory of elementary system nets [70]. A uniform description is given by Priese and Wimmel [61]. In practice, places with unboundedly many tokens are needed only rarely.

Visual Formalisms

Someone who wants to illustrate abstract constructs and relations often uses sketches, diagrams or other graphical notations. Even in the early days of computer science, finite state machines were already modeled as graphs, and programs as flow charts. In the early 1960s, Petri's proposal to represent formal models primarily graphically rather than textually was nevertheless unusual.

Since the advent of state charts in the mid-1980s and of UML in the 1990s, visual formalisms have become established for other modeling techniques as well.

The graphical primitives of Petri nets, that is, the depiction of passive and active components as round and rectangular shapes, as well as the representation of causal relations and dynamically changeable objects as arrows and "tokens", have stood the test of time. They have, in part, been adopted by the UML community.

Sequential and Distributed Runs

Chapter 4

This chapter covers the question of how to formulate individual *runs* (e.g., *calculations*, *behaviors*) of distributed, reactive systems, and what insights into a system such runs can provide.

At first, we will examine the very intuitive term of a *sequential run* of a system net. *Distributed runs* require slightly more effort in understanding, but also describe the behavior more accurately. They form the basis for the concept of *scenarios*, which is covered in the next chapter.

4.1 Sequential Runs

Figure 4.1 shows a technical example of an elementary system net N. The initial marking AC enables the three transitions a, c and d. Hence, there are three steps starting in AC:

$$\mathsf{AC} \xrightarrow{\mathsf{a}} \mathsf{BC},\ \mathsf{AC} \xrightarrow{\mathsf{c}} \mathsf{D} \text{ and } \mathsf{AC} \xrightarrow{\mathsf{d}} \mathsf{AE}.$$

We can formulate a run of the system net N as a sequence of steps, starting with the initial marking M_0. Typical runs for N in Fig. 4.1 are

$$\mathsf{AC} \xrightarrow{\mathsf{a}} \mathsf{BC} \xrightarrow{\mathsf{b}} \mathsf{AC} \xrightarrow{\mathsf{c}} \mathsf{D} \qquad (1)$$

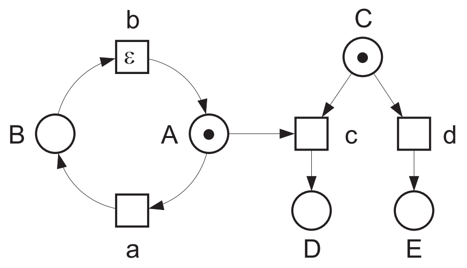


Figure 4.1: System net N

and

$$\mathsf{AC} \xrightarrow{\mathsf{d}} \mathsf{AE} \xrightarrow{\mathsf{a}} \mathsf{BE}. \qquad (2)$$

finite sequential run

The order of d and a in (2) is arbitrary. We could also have chosen

$$\mathsf{AC} \xrightarrow{\mathsf{a}} \mathsf{BC} \xrightarrow{\mathsf{d}} \mathsf{BE}.$$

We define a *sequential run* of a system net N as a sequence

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots$$

of steps of N, starting with the initial marking M_0 of N. A run can be finite or infinite. In Fig. 4.1, the system net N has infinitely many infinite sequential runs, for instance

$$\mathsf{AC} \xrightarrow{\mathsf{d}} \mathsf{AE} \xrightarrow{\mathsf{a}} \mathsf{BE} \xrightarrow{\mathsf{b}} \mathsf{AE} \xrightarrow{\mathsf{a}} \cdots \;. \qquad (3)$$

$$\begin{tikzcd} M_0 \arrow[d, "\mathsf{c}"] \\ M_1 \arrow[d, "\mathsf{a;\ x = 5}"] \\ M_2 \arrow[d, "\mathsf{b;\ y = z = } \square"] \\ M_3 \arrow[d, "\mathsf{d;\ y = } \square"] \\ \vdots \end{tikzcd}$$

sequential run of the cookie vending machine in Fig. 2.1

In general, we are interested in *complete* runs: A finite run

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} M_n$$

is *complete* if M_n is a final marking, that is, if M_n does not enable any hot transitions. Thus, the runs (1) and (2) are complete. Every run ending in BE is indeed complete, but can nevertheless be extended.

infinite sequential run

An infinite run is complete if at no point a step with an additional transition can be inserted. The run (3) is complete. The run

$$\mathsf{AC} \xrightarrow{\mathsf{a}} \mathsf{BC} \xrightarrow{\mathsf{b}} \mathsf{AC} \xrightarrow{\mathsf{a}} \ldots$$

is incomplete: a step with an additional transition d can be inserted at any point.

As explained in Sect. 2.8, it is possible to compile the steps of a system net into a marking graph. Figure 4.2 gives the marking graph of the system net N shown in Fig. 4.1. An additional arrow indicates the initial marking AD. The dashed circles indicate the final markings D and BE. Each sequential run of a system net N is a path through the marking graph G of N. Vice versa, each path through G that starts with the initial marking M_0 of N is an sequential run of N.

$$\begin{tikzcd}
{} \arrow[r] & AC \arrow[rr, "c"] \arrow[rrd, "d"] \arrow[d, bend right, "a"'] & & \boxed{D} \\
& BC \arrow[u, bend right, "b"'] \arrow[rd, "d"] & & AE \arrow[ld, bend right, "a"'] \\
& & \boxed{BE} \arrow[ru, bend right, "b"'] &
\end{tikzcd}$$

Figure 4.2: Marking graph of Fig. 4.1

4.2 Tokens as Labeled Places

Next to the representation of a run as a sequence of steps, there exists the representation as a labeled net with a special structure, called *distributed* run. These runs are based on the idea of representing each occurrence of a token in a place as an individual, labeled place. For a place p of an elementary system net, this means:

$$\textcircled{p} \quad \text{represents} \quad \textcircled{\bullet}\,p.$$

The labeled places $\textcircled{A}$ and $\textcircled{C}$, for instance, represent tokens in the pre-set of the transition c in Fig. 4.1. For the initial marking of the cookie vending machine shown in Fig. 3.1, we use 12 places: five are respectively labeled with storage and counter, one with insertion possible, and one with no signal. Likewise, for a generic system net, tokens $u_1, \ldots, u_n$ in a place p are represented by n correspondingly labeled places:

$$(p\ u_1), \ldots, (p\ u_n) \quad \text{represent} \quad (u_1, \ldots, u_n)\ p$$

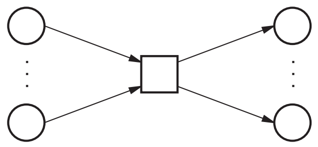
For the initial marking of the cookie vending machine shown in Fig. 2.1, for instance, a total of ten places is used: five of which take the form , two the form and one each the form (D •), (G •) and (E 7).

4.3 Actions

A distributed run consists of *actions*. An action describes the occurrence of a transition, especially its effect on the tokens involved. Technically, an action A is a loop-free, labeled net

action

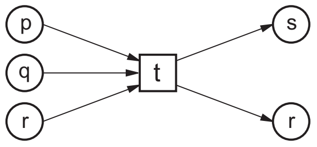
with exactly one transition. The net structure of A thus takes the form:



A *represents* a transition t of an elementary system net if:

- the transition a of A is labeled with t,
- ${}^\bullet a$ represents the tokens in ${}^\bullet t$,
- $a^\bullet$ represents the tokens in $t^\bullet$.

Exemplary and in graphical notation:



represents

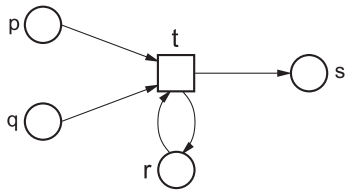


Figure 4.3 shows all actions of the mutual exclusion system shown in Fig. 3.8.

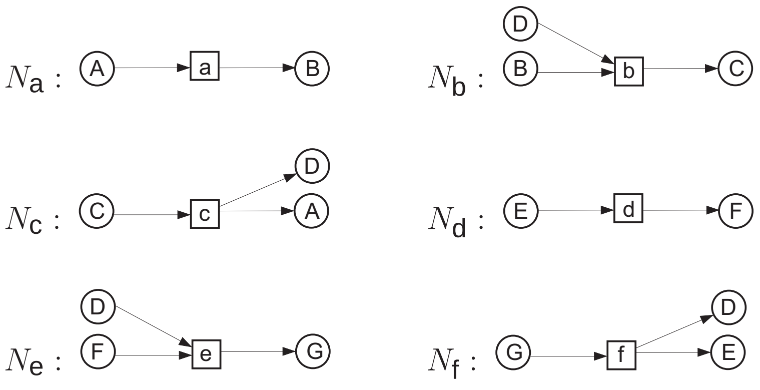
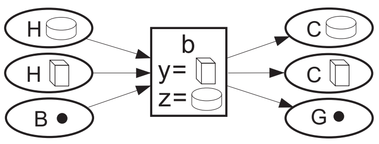


Figure 4.3: The six actions of the mutual exclusion system

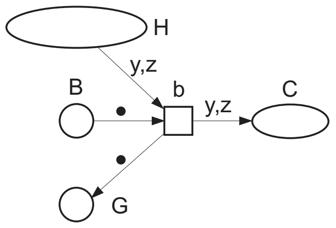
For a generic system net N, an action A *represents* a transition of N in the mode β if:

- the transition a of A is labeled with (t, β),
- ${}^\bullet a$ represents the tokens $\beta(p, t)$ for each $p \in {}^\bullet t$,
- $a^\bullet$ represents the tokens $\beta(t, q)$ for each $q \in t^\bullet$.

As an example, consider the transition b in Fig. 2.1 in the mode β with $\beta(\mathsf{y}) =$ ▯ and $\beta(\mathsf{z}) =$ ⬭. Then the following holds:



represents



in the mode β.

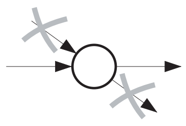
4.4 Distributed Runs

A distributed run consists of actions that are assembled in an acyclic fashion. Thus, the basis for a distributed run is a *causal net*.

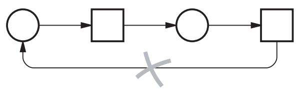
causal net

A causal net $K = (P, T, F)$ has the following characteristics:

- no place branches: at each place, at most *one* arc starts or ends, respectively;



- no sequence of arcs forms a loop: for each sequence of the form
$$k_0 F k_1 \ldots k_{n-1} F k_n$$
thus: $k_0 \neq k_n$;

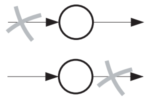


- each sequence of arcs has a first element: thus, there exists no string that "starts in infinity" and takes the form
$$\ldots k F k'.$$

places in the

outset:

end:



The places without an incoming arc form the *outset* of K. The places without an outgoing arc form the *end*. We denote the outset and end by ${}^{\circ}K$ and K°, respectively.

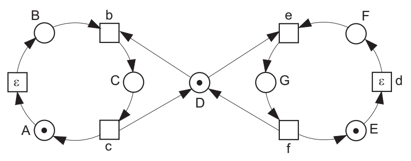
In contrast to a system net, a causal net can very well have infinitely many elements.

distributed run

A *distributed run* of a system net N is a labeled causal net K in which each transition t, together with ${}^{\bullet}t$ and $t^{\bullet}$, forms an action of N. K thus describes an uninterrupted part of the behavior of N.

Figure 4.4 shows a distributed run of the mutual exclusion system, employing each of the actions shown in Fig. 4.3 exactly once.

The upper chain of arcs consists of the actions N_{a}, N_{b} and N_{c} (see Fig. 4.3) and describes a cycle of the left process from its initial state A to B and C and back to A. Likewise, the lower chain of arcs describes a cycle of the right process from E to F and G and back to E.



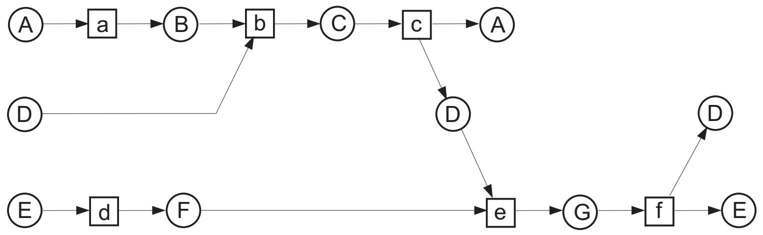


Figure 4.4: Distributed run of the mutual exclusion system

In general, a chain of arcs from an element x to an element y intuitively describes that x causally precedes y. The (three) occurrences of D in Fig. 4.4 are in accordance with this: the key is first used by the left process (via b and c), then by the right one (via e and f) and is again available at the end of the run. Thus, the actions N_{a}, N_{b}, N_{c} and N_{d} causally precede N_{e}.

On the other hand, N_{a} and N_{d} are not linked by a chain of arcs: they are *causally independent* from each other. This also holds for N_{c} and N_{d}.

An action can, of course, occur repeatedly in a distributed run. Figure 4.5 expands Fig. 4.4 by another cycle of the left process, using additional instances of the actions N_{a}, N_{b} and N_{c}.

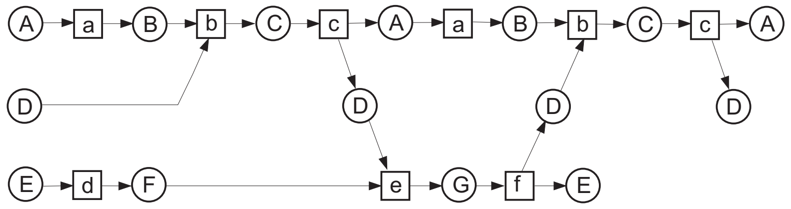


Figure 4.5: Expansion of the distributed run in Fig. 4.4

In general, a distributed run can be infinitely long – just as a sequential run. Figure 4.6 outlines a run of the mutual exclusion system in which, at first, the left process becomes critical. Then it remains in its local state A forever, and the right process executes its cycle infinitely often.

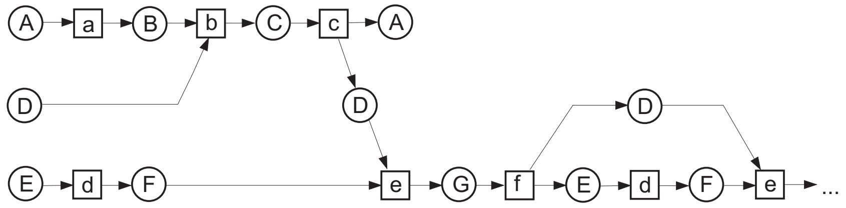


Figure 4.6: First l once, then r infinitely often

A distributed run K of a system net N is *complete* if and only if its outset ${}^\circ K$ represents the initial state of N and its end K° does not enable any hot transitions. For convenience, we will omit the attribute "complete." Unless noted otherwise, for the rest of this book, with "distributed run" we always mean "*complete* distributed run."

complete distributed run

The end N° of a finite distributed run of a system net N represents a final state of N. The end of an infinite run, in general, does not represent a reachable state. The end of the run in Fig. 4.6, for instance, consists of only a single place (labeled A).

Figure 4.7 shows a partial distributed run of the cookie vending machine shown in Fig. 2.1. The run describes exactly one

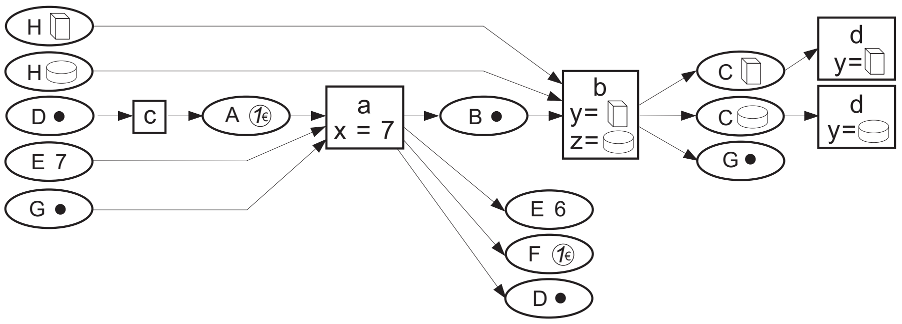


Figure 4.7: A partial distributed run of the cookie vending machine shown in Fig. 2.1

sale, from the insertion of the coin until the withdrawal of two cookie packets.

Figure 4.8 outlines the two finite and the – only – infinite distributed sequence of the elementary system net shown in Fig. 4.1.

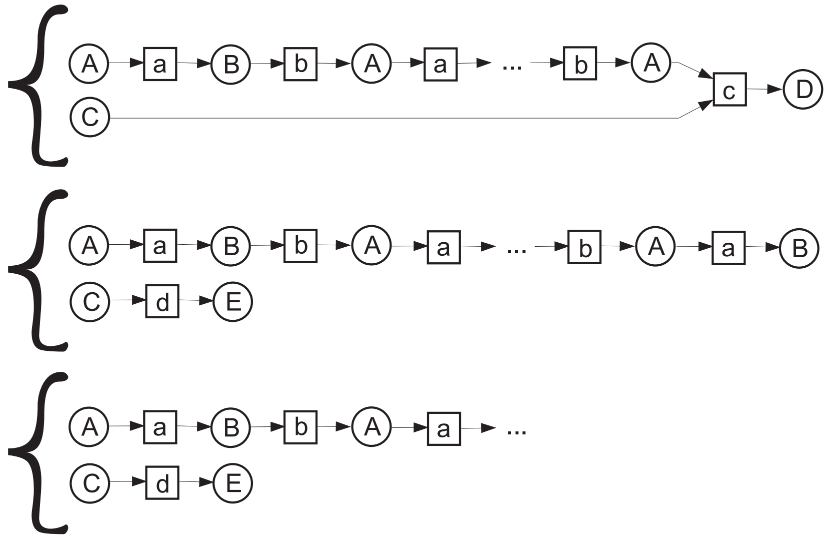
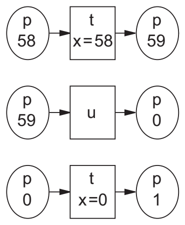


Figure 4.8: The many finite and the one infinite run to Fig. 4.1

4.5 Example: A Bell Clock

Figure 4.9 shows a minute clock: A counter is repeatedly incremented by 1 and reset to 0 after 60 increments. The place p always holds exactly one token, a natural number between 0 and 59. Starting with 58, the transition t always increments this number by 1, until it is set from 59 to 0.

actions of the minute clock



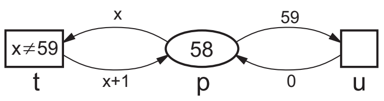


Figure 4.9: The minute clock

For a system net with such a simple structure, the distributed runs resemble the sequential runs. The net in Fig. 4.9 has exactly one distributed run:

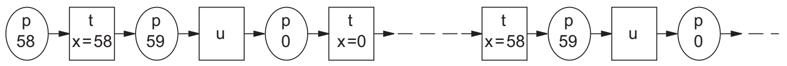


Figure 4.10 adds bells to the minute clock, which will start chiming at the beginning of every new hour (transition u). The end of the bell chimes is not linked to the clock. Figure 4.10 only specifies that the chimes have stopped (transition v) before the next full hour. The transitions t and v occur independently from each other. This is also shown by the bell clock's – only – distributed run in Fig. 4.11.

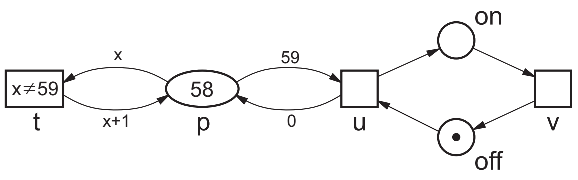


Figure 4.10: The bell clock

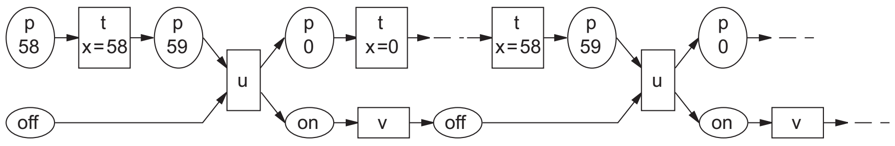


Figure 4.11: Run of the bell clock

4.6 The Kindergarten Game

The *kindergarten game* is the distributed version of a program used by Dijkstra [20] to demonstrate the use of formal verification techniques. We use this distributed variant as an example to show the uses of Petri nets and distributed runs (and verify them in Sect. 13.10). Dijkstra assumes *one* agent playing with given objects (pebbles). We assume the pebbles themselves to be agents and therefore replace them with children, for descriptive purposes.

The kindergarten game starts out with an arbitrary number of children in a playing area. Each child is dressed in either black or white. At any one time, two children can spontaneously leave the playing area together. After that, one child returns, according to the following rules:

- If the children are dressed in different colors, the one dressed in white returns;
- If the children are dressed in the same color, a child dressed in black returns (if both are dressed in white, one child changes).

We want to develop an appropriate model for this game. To do this, we construct a place children, with the group of children as initial marking. If we depict a child dressed in black or white as $\bullet$ or $\circ$, respectively, and abbreviate (children, $\bullet$) and (children, $\circ$) as $\bullet$ and $\circ$, respectively, then Fig. 4.12 shows the three possible actions of the game.

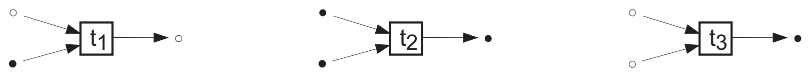


Figure 4.12: The three actions of the kindergarten game

Figure 4.13 models the entire game. As initial marking, a finite number of children, each dressed in either black or white, is assumed. Each transition in Fig. 4.13 is hot. Thus, a run terminates only if no transition is enabled, which means that only one child remains. The three runs in Fig. 4.14 all start with the initial marking $M_0(\text{children}) = [\circ, \circ, \bullet, \bullet, \bullet]$.

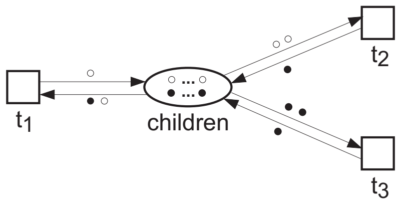


Figure 4.13: Model of the kindergarten game

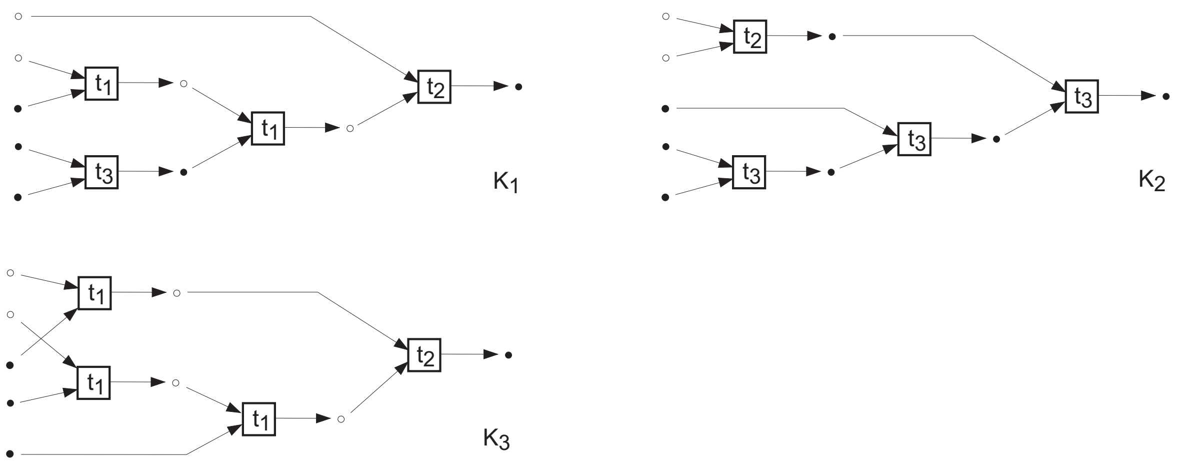


Figure 4.14: Three distributed runs of the kindergarten game

To what extent the color of the last remaining child's clothes depends on the initial marking and the structure of the respective run is covered in Sect. 13.10.

4.7 Causal Order

In contrast to sequential runs, distributed runs show the *causal* relations of actions. Fig. 4.15 exemplifies this with two system nets, both of which have the same sequential runs:

$$\text{ACE} \xrightarrow{a} \text{BCE} \xrightarrow{b} \text{BDE} \quad \text{and}$$
$$\text{ACE} \xrightarrow{b} \text{ADE} \xrightarrow{a} \text{BDE}.$$

Their distributed runs, however, are distinct: N_l has a single distributed run, shown in Fig. 4.16. N_r, on the other hand, has two distributed runs, shown in Fig. 4.17.

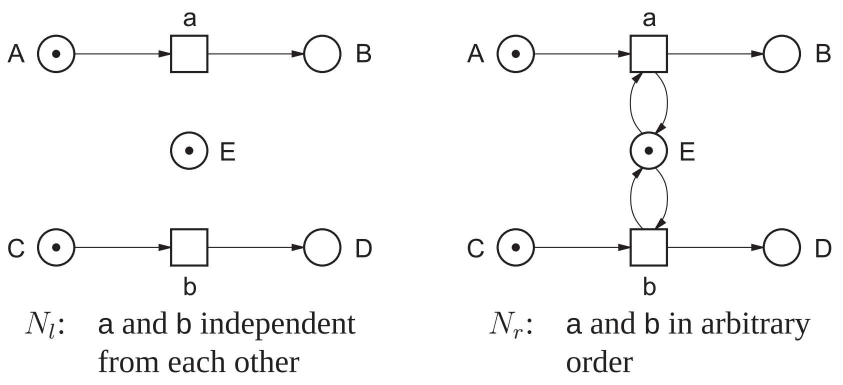

a
A
B
E
C
D
b
N_l: a and b independent from each other
a
A
B
E
C
D
b
N_r: a and b in arbitrary order

Figure 4.15: Independence and arbitrary order

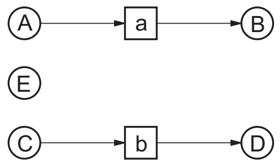


Figure 4.16: The distributed run of N_l

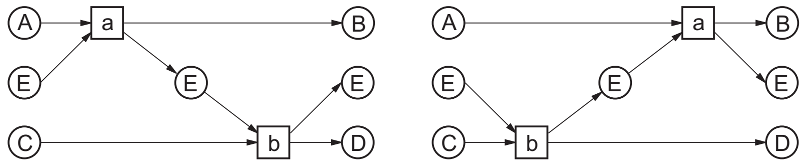
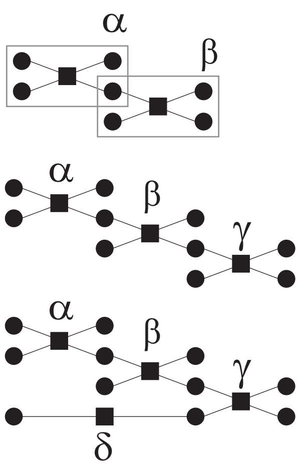


Figure 4.17: The distributed runs of N_r



causal order

In N_r, the place E can be seen as a model of a resource that is used, but not used *up*. This forces the two actions of a and b into one of two possible orders.

If an action α generates tokens that are used by another action β, then α *causally precedes* β. This "causally precedes" relation is, of course, transitive: if α precedes β and β precedes γ, then α also precedes γ. Furthermore, *causally precedes* is irreflexive: no action causally precede itself. Thus, *causally precedes* is a strict partial order.

In general, this order is indeed not total: An action δ may occur independently from β. However, this does not imply that δ and β occur simultaneously. Simultaneity is transitive; independence is not.

The relation between the sequential and the distributed runs of a system net N becomes evident if, in every distributed run K of N, the corresponding tokens are put in the places in ${}^\circ K$. Thus, K itself becomes a system net with an initial marking.

In this case, every sequential run of K is also a sequential run of N. The run K_1 in Fig. 4.14 is an example of this. It generates the two sequential runs in Fig. 4.18.

Vice versa, every sequential run of N is generated by at least one distributed run K of N.

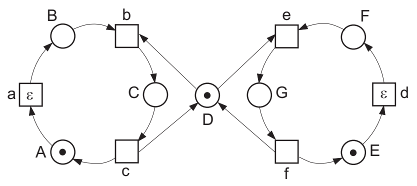
$$[\bullet,\bullet,\bullet,\circ,\circ] \xrightarrow{t_1} [\bullet,\bullet,\circ,\circ] \xrightarrow{t_3} [\bullet,\circ,\circ] \xrightarrow{t_1} [\circ,\circ] \xrightarrow{t_2} [\bullet]$$

$$[\bullet,\bullet,\bullet,\circ,\circ] \xrightarrow{t_3} [\bullet,\bullet,\circ,\circ] \xrightarrow{t_1} [\bullet,\circ,\circ] \xrightarrow{t_1} [\circ,\circ] \xrightarrow{t_2} [\bullet]$$

Figure 4.18: The two sequential runs generated by K_1

4.8 The Composition of Distributed Runs

Let K and L be two distributed runs. Their *composition* $K \cdot L$ is formed by identifying the end K° of K with the outset ${}^\circ L$ of L. To do this, K° and ${}^\circ L$ have to represent the same marking. $K \cdot L$ contains all the elements of K and L and retains their order. Figure 4.19 shows two composable runs K and L of the mutual exclusion system. An initial segment of their composition $K \cdot L$ is shown in Fig. 4.5.



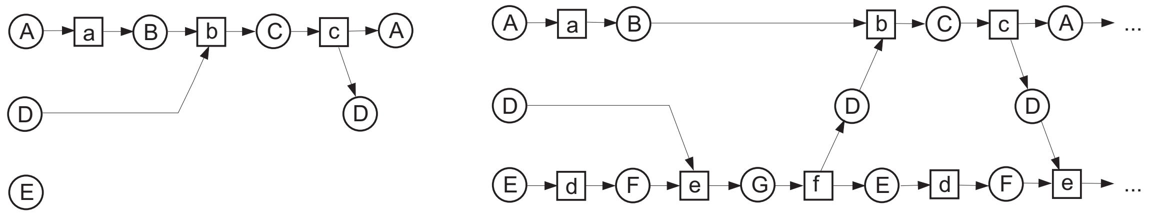


Figure 4.19: Two composable runs

If K and L can be composed, the run $K \cdot L$ is defined as composition

$$(P_K \cup P_L, T_K \cup T_L, F_K \cup F_L).$$

Exercises

1. Consider the system net in Fig. 4.20:

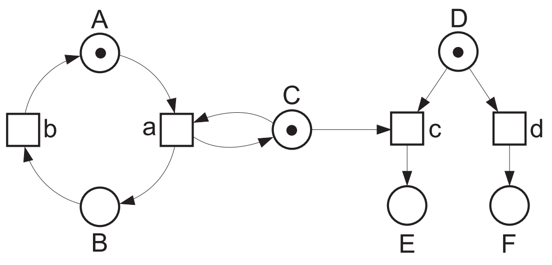


Figure 4.20: System net N

   (a) Determine the number of
      (1) infinite sequential runs,
      (2) infinite distributed runs,
      (3) finite sequential runs,
      (4) finite distributed runs.

   (b) Which of the answers to (a) change if one of the four transitions is assumed to be cold?

   (c) Compare the runs of the system nets in Fig. 4.20 and Fig. 4.1.

2. (a) Specify the actions of the system net N in Fig. 3.3.

   (b) Characterize all distributed runs of N.

3. Consider the bell clock in Fig. 4.10:

   (a) Expand the bell clock by a place that shows the current hour between 1 and 12. Modify the chimes so that on each n-th hour, the bell chimes n times.

   (b) Modify the bell clock so that when the tokens 15, 30, 45 and 0 are reached in place p, the bell chimes once, twice, three times and four times, respectively.

   * (c) Now combine the two system nets from (a) and (b) so that on each nth hour, the bell first chimes four times and then, after a pause, n times.

   (d) Construct finite initial segments of the runs of all subtasks.

4. Prove the following:
   If J can be composed with K, and K with L, then $(J \cdot K) \cdot L = J \cdot (K \cdot L)$.

Further Reading

In the literature, sequential runs are by far the predominant formalism. They are technically simple, sufficient to model many situations, and satisfy the common intuition that events occur along a global time scale, or are mapped onto such a scale by an observer. From the beginning, the concept of a distributed run, that is, a partially ordered set of local states and transitions, has been an important part of the theory of Petri nets [36], [58]. In 1981, Nielsen, Plotkin and Winskel [54] put distributed runs in the context of the mathematical structures of other system models. In 1988, Best and Fernandez discussed a multitude of relations between system nets and distributed runs [10]. Two aspects of distributed runs are particularly important today: their use in scenarios (Chapter 5) and their contribution to the control of the state space explosion during verification through model checking [49], [77]. To test interesting characteristics of a system, this process compiles sufficiently long initial segments of all its distributed runs (that start with the inital marking) into a tree-like structure. This structure is then efficiently analyzed. Esparza and Heljanko [23] have shown numerous proof techniques based on temporal logic that make use of this structure.

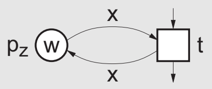
Read and Write vs. Give and Take

Conventional representations of algorithms, especially conventional programming languages, use memory cells, which store their respective current values as state components. Dynamic behavior is described as a value assignment of the form:

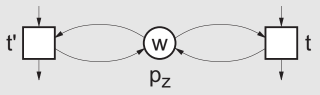
$$x_0 := f(x_0, \ldots, x_n).$$

Depending on the current values in the memory cells $x_0, \ldots, x_n$, the cell x_0 receives a new value. The previous value in x_0 is then not accessible anymore. "Read and (over)write" are thus the basic operations of programs. For many algorithms, this is intuitive and appropriate, for example, in Euclid's algorithm for computing the greatest common divisor of two natural numbers. However, different algorithms have different basic operations, for example, the sending and receiving of messages or objects. Therefore, Petri nets use a different approach: a state component is – intuitively – an object in a certain location. Dynamic behavior moves objects from one location to another. Some objects may be created (coming from an unmodeled environment) or they may disappear (into it) during the process. Thus, "give and take" are the basic operations here.

This prompts us to try to translate programs into Petri nets and vice versa: each variable z in a program corresponds to a place p_z in a system net N, and the current value w in z corresponds to a token in p_z. The reading of z is simulated with a transition t and a loop between p_z and t:

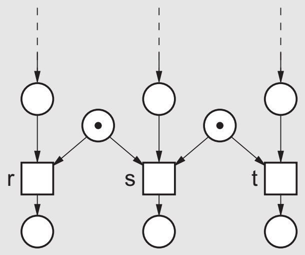


In a concurrent program with two processes, a second process in N would generate a second loop between p_z and a second transition t'.



In each run of N, the transitions t and t' can occur arbitrarily often, but always one after another and never independently. For example, t' may occur infinitely often and block t. This contradicts the assumption that processes can read a variable independently without interfering with each other. If the updating of the variable z is also modeled as a loop between a transition t" and the place p_z, the updating transition t" can be blocked by the reading transition t, in contradiction to the common assumption of concurrent programming in which the updating of a variable is never blocked by the reading of the variable. This observation has far-reaching consequences, some of which will become apparent in the case study of the mutual exclusion system in Chap. 20.

The translation of a system net N into a program P synchronizes the transitions of N into one or more control flows. The structure



of N generates (among others) a distributed run in which r and t occur independently from each other. A corresponding program would have to organize the nondeterminism between r and s and between t and s and thus force r and s under a central control.

Finally, programming languages and many other operational modeling techniques allow for multiple *control flows*. They are, if necessary, dynamically created or terminated, but rarely reach the necessary flexibility to model, for example, the solution to the kindergarten game in Fig. 4.13.

Scenarios — Chapter 5

A user of a technical or organizational system usually does not need each and every possible behavior of the system. The work is often limited to a few activities that are repeatedly executed. Therefore, it is convenient to consider typical *scenarios* of a system. A scenario consists of a finite number of elementary actions and terminates in the same state in which it started. Because of this, multiple *instances* of a scenario can occur multiple times in the same run. A scenario often describes an interaction pattern between a process and its environment, or between two processes.

Typical scenarios of the systems considered thus far are:

- selling a packet of cookies,
- visiting one's critical state once,
- sending a message.

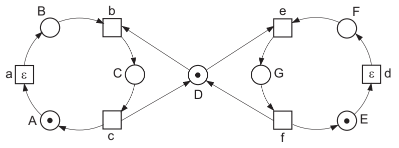
Technically, a scenario is constructed as a finite distributed run, whose final marking equals its initial marking. A run of a distributed, reactive system is often composed of many instances of only a few scenarios. If *every* run can essentially be constructed like this, the system is *scenario-based*. Understanding the scenarios of a system is often the easiest way to understand the entire system. We will illustrate this using the examples of the mutual exclusion system, the crosstalk algorithm and the cookie vending machine.

5.1 Defining Scenarios

scenario

A *scenario* of a system net N is a finite partial distributed run K of N in which the outset, ${}^\circ K$, and the end, K°, represent the same marking.

Figure 5.1 shows two scenarios of the mutual exclusion system: one for the left and one for the right process. Each process passes through the cycle of local, to waiting and critical, back to local. The kindergarten game (Sect. 4.6) is an example of a system net without scenarios.



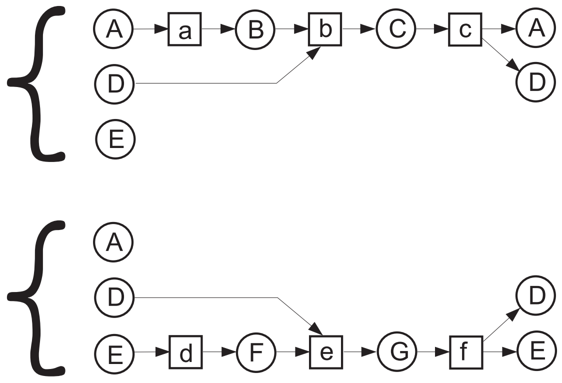


Figure 5.1: The two scenarios of the mutual exclusion system

From the definition of the composition of distributed runs in Sect. 4.8 it follows that arbitrarily many instances $K_0, \ldots, K_n$ of different scenarios with the same initial and final conditions can be assembled into a finite or infinite partial distributed run

$$K_0 \cdot \ldots \cdot K_n \quad \text{or} \quad K_0 \cdot K_1 \cdot \ldots,$$

respectively. For example, each finite distributed run of the mutual exclusion system shown in Fig. 3.2 can be assembled from instances of the two scenarios shown in Fig. 5.1.

scenario-based system net

A system net N is *scenario-based* if there exists a finite set A of scenarios such that each complete finite or infinite distributed run of N can be written as

$$K_1 \cdot K_2 \cdot \ldots \cdot K_n \quad \text{or} \quad K_1 \cdot K_2 \cdot \ldots,$$

respectively, where each K_i is a scenario in A. Multiple instances of the same scenario may, of course, occur in the same run. Occasionally, a run may also have an irregular initial or final segment K_0 or K_n, respectively.

The mutual exclusion system is obviously scenario-based: the two scenarios shown in Fig. 5.1 suffice. An infinite run may also contain an instance of the action a or d.

5.2 The Scenarios of the Crosstalk Algorithm

The two processes l and r of the crosstalk algorithm from Chap. 3 (Fig. 3.7) cooperate in three scenarios:

- l sends, r receives;
- r sends, l receives;
- both send and receive, and crosstalk occurs.

One may assume a more abstract view and summarize the first two scenarios as “one sends, one receives”.

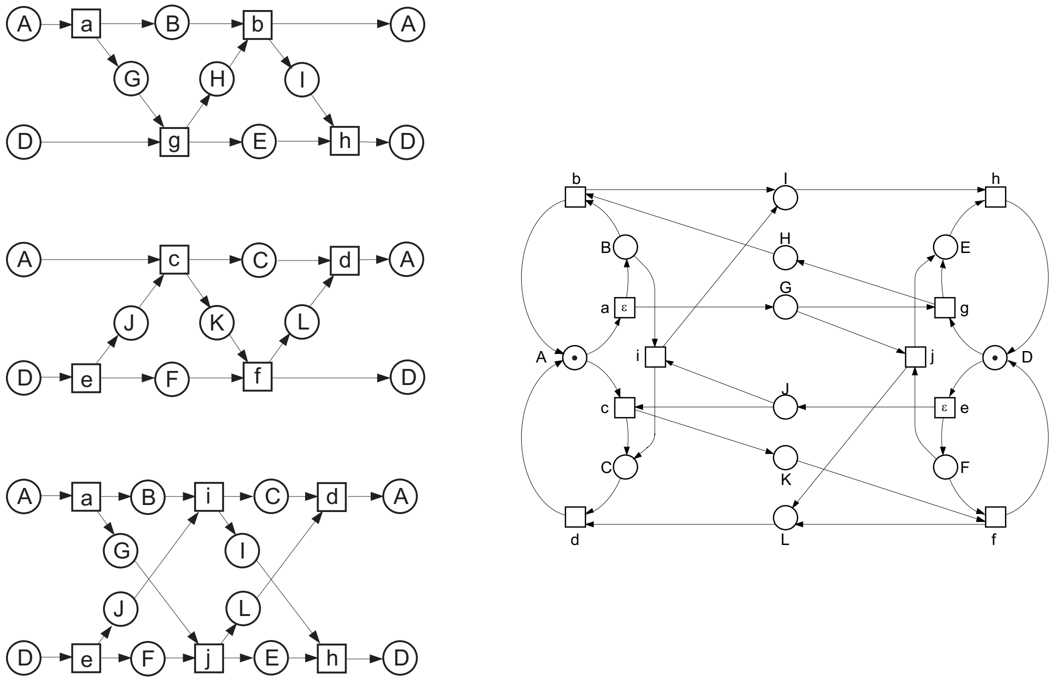


Figure 5.2: The three scenarios of the crosstalk algorithm

Figure 5.2 shows the three scenarios of the algorithm with the symbolic denotations from Fig. 3.8. The crosstalk algorithm is scenario-based: each distributed run consists of a finite or infinite number of instances of the three scenarios shown in Fig. 5.2.

5.3 The Scenarios of the Cookie Vending Machine

The cookie vending machine from Chap. 1 and 2 has only one scenario: the return of an inserted coin. Figure 5.3 shows this scenario for the version of the machine shown in Fig. 2.1.

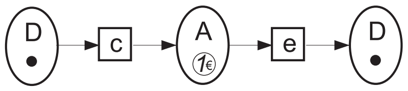


Figure 5.3: Scenario of the cookie vending machine

Intuitively, one might also consider selling a cookie packet as a scenario. In its outset, a coin can be inserted, no signal is pending, and the compartment is empty. The partial run in Fig. 4.7 shows this behavior of the cookie vending machine. To form a complete scenario, the tokens in the counter, the cash box and the storage would have to match in the outset and at the end. To also cover such cases, we will weaken the definition of a scenario of a system net N:

A scenario for a marking M of N is a finite partial distributed run K of N in which a part of $^\circ K$ and a part of K° represent the marking M. Figure 4.7 thus shows a scenario of the cookie vending machine for a marking M with $M(D) = M(G) = [\bullet]$. Figure 5.4 shows another such scenario. Both can be composed to form the partial run shown in Fig. 5.5.

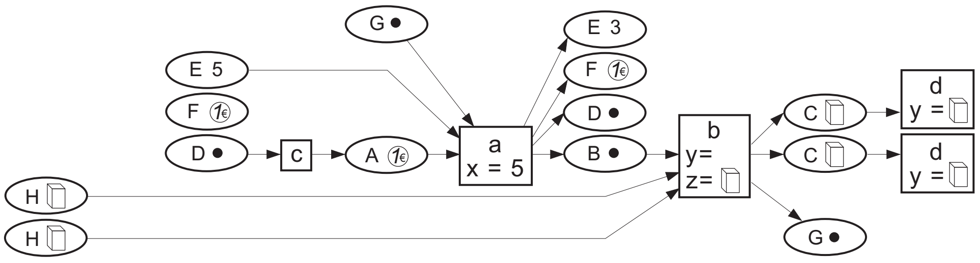


Figure 5.4: Scenario for the marking M with $M(\mathsf{D}) = M(\mathsf{G}) = [\bullet]$

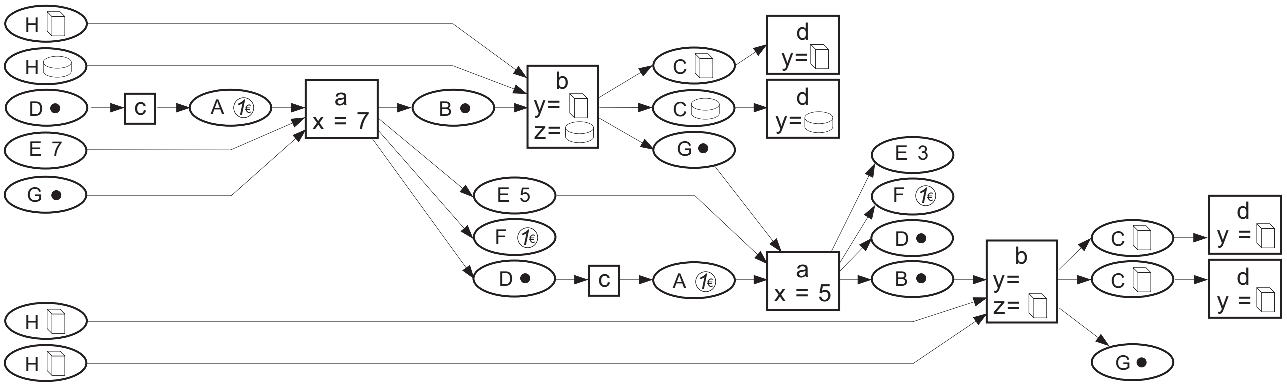


Figure 5.5: Distributed run of the cookie vending machine, composed of the scenarios for the marking $M(\mathsf{D}) = M(\mathsf{G}) = [\bullet]$ shown in Figs. 4.7 and 5.4

Exercises

1. (a) How many scenarios does the algorithm in Fig. 3.3 have? Give one.
   (b) How many finite sequential runs does that scenario describe?

2. Does the system net in Fig. 5.6 have scenarios? Is it scenario-based?

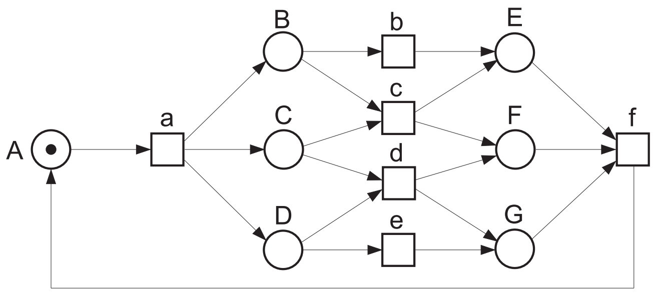


Figure 5.6: System net

3. (a) Construct a scenario for the system net in Fig. 5.7.

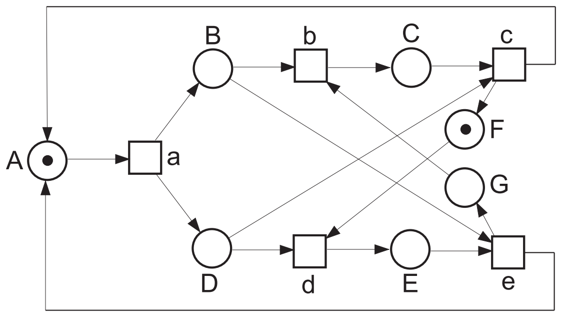


Figure 5.7: System net

   (b) Is the system net scenario-based?

Further Reading

Thinking in *scenarios* can significantly simplify and deepen our understanding of a complex system. In connection with Petri nets, Desel [16], in particular, has proposed the use of scenarios. *Message sequence charts* describe scenarios very explicitly. They form the basis of a modeling and simulation technique in [35].

Scenario-Based System Nets

The term “scenario” is not used very consistently in the field of software engineering. In general, the term “scenario” is used to describe a coherent section of behavior that is contiguous from the observer’s perspective. Typically, a scenario is used interactively to reach a goal. From the perspective of the system architecture, a scenario generally covers several components and only uses a part of each component. A scenario is usually *reactive*, because it needs input from the environment during its process. In the mutual exclusion system and the crosstalk algorithm, all scenarios have this characteristic. This also holds for the example of the cookie vending machine if the machine and each user are considered to be components.

Because scenarios describe the behavior that a user has envisioned for a specific system, it would be most useful if one could derive a system from a given set of scenarios. In the above systems of the crosstalk algorithm, the cookie vending machine and the mutual exclusion system can, in fact, be derived from the corresponding scenarios by identifying elements with the same denotations. Harel greatly extends this idea for Live Sequence Charts [35] and integrates it into a tool.

Further Notations for for Elementary System Nets

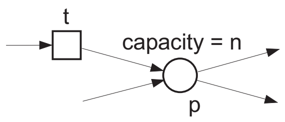
Chapter 6

In the literature, elementary system nets are often extended by additional notations, especially *capacities* (“no transition can add an $(n+1)$th token to a place p”) and arc weights (“n tokens simultaneously flow through an arc”). We will discuss both of these extensions and show that they do not increase the expressiveness of elementary system nets: they can be simulated and are thus merely syntactic sugar. Occasionally, it is possible to use capacities and arc weights to construct very intuitive and clear models. In such cases, they should definitely be used.

However, a transition that is supposed to “test” the number of tokens in a place, or that should be given priority in case of a conflict, is a different matter: such things cannot be simulated within the bounds of elementary system nets.

6.1 Place Capacities

One occasionally wants to express that a place p can hold at most n tokens, and that a transition t that would add an $(n+1)$th token should thus not be enabled. In the graphical representation of a system net N, the place p could be labeled accordingly



and the conditions for the enabling of t in a marking M be expanded by $M(p) \leq n-1$.

This does not increase the expressiveness of elementary system nets. To show this, let N be an elementary system net including a place p with a capacity of n. A marking M of N is

n-bounded marking

called *n-bounded* if

$$M(p) \leq n.$$

n-reachable marking

In particular, let the initial marking M_0 of N be n-bounded. We now construct an elementary system net N' such that the reachable bounded markings M of N and the markings M' of N' unambiguously correspond to each other, and that for all bounded markings M_1 and M_2 of N the following holds:

$$\begin{aligned} &M_1 \xrightarrow{t} M_2 \text{ is a step of } N \text{ iff} \\ &M_1' \xrightarrow{t} M_2' \text{ is a step of } N'. \end{aligned} \qquad (1)$$

N' consists of N, expanded by a place p'. For each transition t of N that is not connected to p by a loop, N' contains an additional arc of the form

- (t, p') if (p, t) is an arc of N,
- (p', t) if (t, p) is an arc of N.

n-complement of p

The initial marking M_0 of N is expanded in N' by

$$M_0(p') = n - M_0(p).$$

N^n

The place p' is then the *complement* of p. Figure 6.1 shows this construction. Proposition (1) now follows directly from the step rule for transitions.

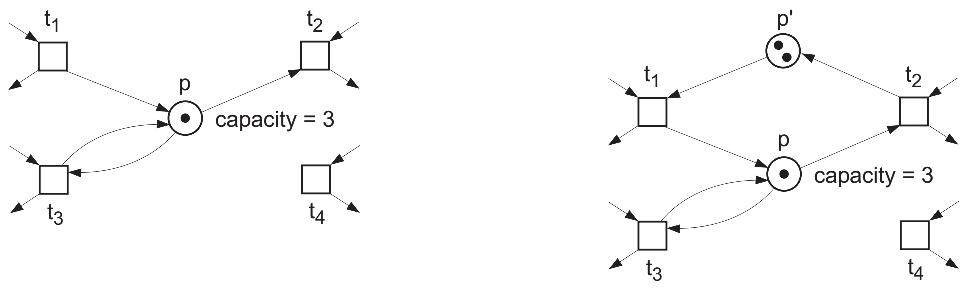


Figure 6.1: Complement p′ of p

6.2 Arc Weights

Occasionally, a model is needed that only contains black tokens (like elementary system nets), but still adds to or removes from a place more than one token during a single step (which is easily achieved for the generic system nets in Chap. 2). To graphically represent such a net, the corresponding arc is labeled with the corresponding factor (*arc weight*). Figure 6.2 shows an example: when all the packets have been sold, the money is taken from the cash box, the storage is refilled and the counter is reset to 5.

w-step

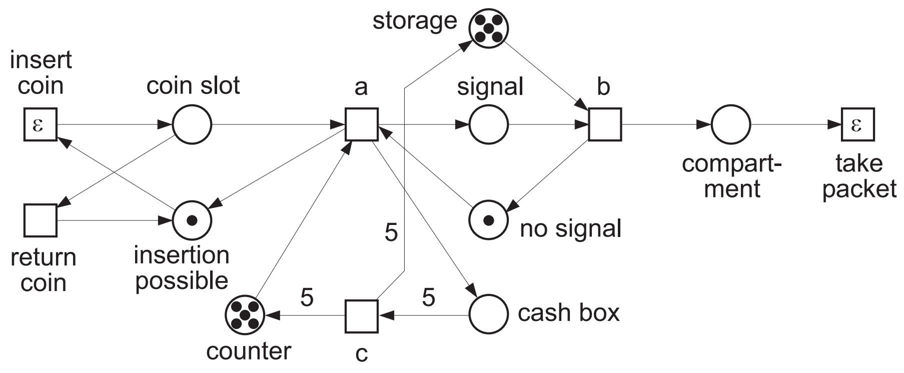


Figure 6.2: Expansion of Fig. 3.1 by refilling the storage with 5 packets

An elementary system net N with arc weights can, as with place capacities, also be simulated by a net N' without arc weights. However, N and N' are not related as closely anymore: N' contains additional places and transitions, and a single step of N is simulated by a sequence of steps of N'. To be precise, a sequential run

w-generalization

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots \xrightarrow{t_k} M_k$$

of N is called a *reduction* of a sequential run

n-reduction

$$M'_0 \xrightarrow{t'_1} M'_1 \xrightarrow{t'_2} \dots \xrightarrow{t'_{k'}} M'_{k'}$$

of N' if the sequence $t_1 \dots t_k$ is generated from the sequence $t'_1 \dots t'_{k'}$ by eliminating from $t'_1 \dots t'_{k'}$ all the transitions that are not in N. The net N' is now constructed such that:

The sequential runs of N are the reductions of the sequential runs of N'. (2)

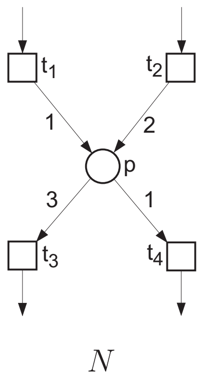
For the construction of N', each place p of N is replaced by a sequence of places and transitions. Figure 6.3 shows an example of this. There the step sequence

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} M_3$$

in N is a reduction of the step sequence

$$M'_0 \xrightarrow{t_1} M'_1 \xrightarrow{t_2} M'_2 \xrightarrow{u_1} M'_3 \xrightarrow{u_2} M'_4 \xrightarrow{u_3} M'_5 \xrightarrow{u_1} M'_6 \xrightarrow{t_3} M'_7$$

in N'. Other step sequences in N' may generate the same reduction.



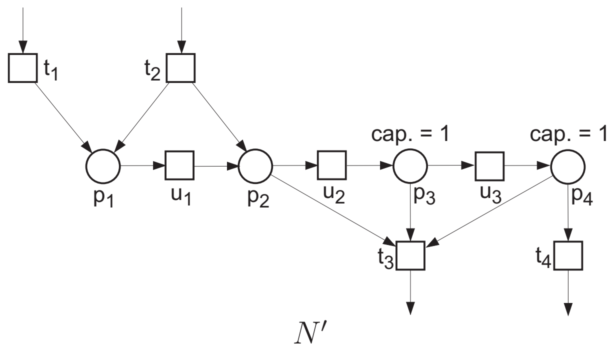
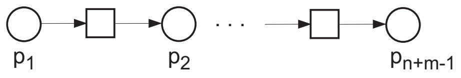


Figure 6.3: Technical example of the simulation of arc weights

The general procedure for constructing N' from N replaces each place p with a sequence



of places and transitions. Its length $n + m - 1$ results from the greatest weights n and m of arcs that end and start in p, respectively. In Fig. 6.3 thus $n = 2$ and $m = 3$, which results in $2+3-1=4$. Furthermore, for all places $p_{n+1}, \ldots, p_{n+m-1}$ a capacity of 1 is required. In Fig. 6.3 those are the places p_3 and p_4.

Each arc (t, p) of N that ends in p with an arc weight of k is replaced in N' with k arcs of the form $(t, p_1), \ldots, (t, p_k)$. Likewise, each arc (p, u) of N that starts in p with an arc weight of l is replaced in N' with l arcs of the form $(p_{n+m-l}, u), \ldots, (p_{n+m-1}, u)$. The initial marking $M_0(p)$ of N is assigned to $M_0(p_1)$ of N'.

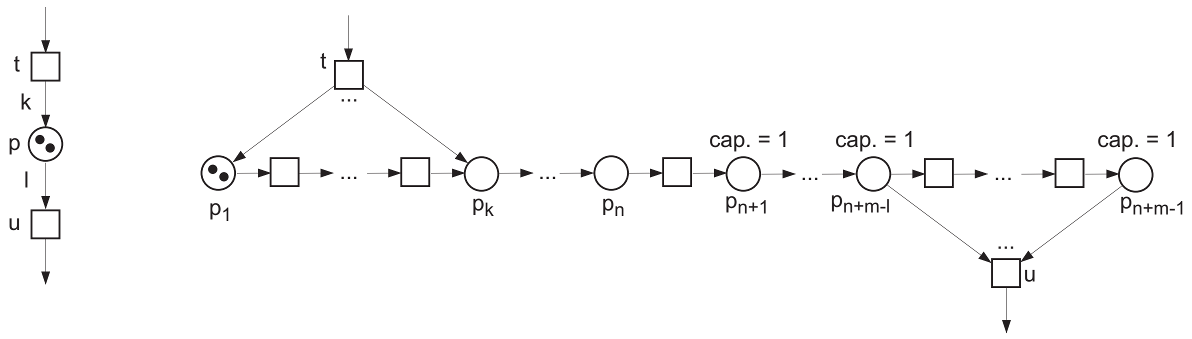


Figure 6.4: Simulation of arc weights

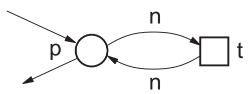
Figure 6.4 outlines this construction. It is easy to show that this construction satisfies (2). In the special case of $n = m = 1$, the construction leaves the place p and its surroundings unchanged.

6.3 Real Extensions

A transition t of an elementary system net N is a $\geq n$*-tester of a place* p if for each marking M the following holds: M enables t if

$$M(p) \geq n. \tag{3}$$

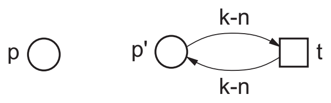
In the construction



(4)

t is obviously a $\geq n$-tester of p. If t is already a transition of N, this construction also implements the condition "p contains at least n tokens" for the occurrence of t.

The construction of a $\leq n$-tester of p analogous to (3) is not so simple. If a capacity $k \geq n$ is already known for p, the complement p' (cf. Section 6.1) can be tested for "$\geq k - n$" instead:



Because $M(p) + M(p') = k$ for each marking M, it follows that $M(p) \leq n$ iff $M(p') \geq k - n$.

If p is unbounded, no complement can be constructed and this method fails. For an unbounded place p no "tester" transition t can be constructed that tests for $M(p) = 0$, $M(p) \leq n$ or $M(p) = n$ analogous to (4). Closely related to the test for "$= 0$" is the increase of expressiveness by means of inhibitor arcs. An *inhibitor arc* (t, p) requires for the enabling of t that p does not contain any tokens. Reset arcs also increase the expressiveness of elementary system nets. A *reset arc* (t, p) removes all tokens from p at the occurrence of t.

Priorities are another example of real extensions: let t and u be two transitions of an elementary system net. The transition t takes *priority* over u if t always occurs whenever a marking M enables both t and u. As with the test for "0", priorities cannot be simulated by elementary system nets if t and u are adjacent to an unbounded place. Also, priorities are not compatible with the concept of distributed runs.

Exercises

* 1. Develop a procedure to simulate a system net with capacities and arc weights with an elementary system net. Apply your procedure to the system net in Fig. 6.5.

   Hint: At first, generalize the concept of the complement of a place from Section 6.1 to include arc weights. Then, analogously to the approach in Section 6.1, use this to construct a system net without capacities, but with arc weights. Simulate this system net using the method in Section 6.2. The result is a system net without arc weights, but with capacities. Eliminate the capacities following the procedure in Section 6.1.

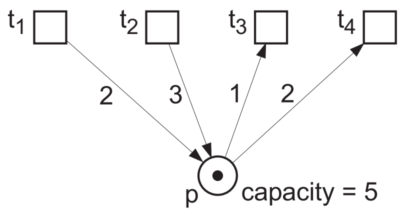


Figure 6.5: System net with capacity and arc weights

2. Convert the cookie vending machine in Fig. 6.2 into an elementary system net. As a simplification, you may limit the filling of the storage to three tokens.

Further Reading

The novice modeler sometimes perceives the rule for the occurrence of transitions as limiting and cumbersome. Various ways of altering, generalizing or extending it come to mind. Section 6.3 shows a few such extensions. However, such additional expressiveness comes at a price: the models quickly become difficult to comprehend, the combination of such concepts is prone to inconsistency, and many analysis techniques fail at first. However, some can be newly devised. An example of this is Busi's analysis of inhibitor arcs [11]. Among the many other variants and generalizations, three are particularly interesting:

Montanari and Rossi [52] propose *context places*. A context place can be accessed by multiple transitions simultaneously without hindrance. They can, for instance, be used to solve the problem described in the postscript to Chapter 4: *Read and Write vs. Give and Take*.

So-called *self-modifying* Petri nets, proposed by Valk in [75], employ arcs with *variable* arc weights. The actual weight of an arc then corresponds to the current number of tokens in one of the net's places.

The authors of [18] propose special *signal arcs* between transitions. With such an arc between t and u, the two transitions will occur *simultaneously* if both are enabled. If only t is enabled, then only t will occur. Such behaviour can be simulated with inhibitor arcs. A good historic overview over different extensions can be found in the *Advanced Courses on Petri Nets* [70, 18].

A range of publications pursue the idea of tokens with a special structure and of using this structure for the formulation of the occurrence rule and for the analysis of system characteristics. An example of this is the “nets as tokens” concept proposed by Valk [76].

The Synthesis Problem — Chapter 7

A system is often modeled by a description of its observable behavior, that is, global states and steps. However, to implement a system, it is often more practical to identify *local* state components and actions whose cause and effect are limited to a few state components. At first, we will discuss this problem using the example of the light/fan system, and then give the problem a precise form and solve it in the rest of this chapter. The techniques presented here will be used in the case study in Chapter 21 to systematically create an asynchronous hardware architecture.

7.1 Example: The Light/Fan System

The reader is probably familiar with the common connection between lighting and air ventilation in (windowless) bathrooms: If the light is switched on while the fan is off, the latter will start as well after some time. If the light is then switched off, the fan will continue running for some time. If the fan is off and the light is first switched on and then quickly switched off again, the fan will not start at all. If the fan is running and the light is switched off and then quickly switched on again, the fan will continue running without interruption.

Traditionally, systems are often modeled as *state automata*: a state automaton consists of *states* and *steps*. One state is the *initial state*. Every step transforms one state into another and thereby executes an action. Several steps may quite well execute one and the same *action*. Technically, a state automaton can be described as a graph, with states as nodes and steps as labeled edges.

Figure 7.1 shows the behavior of the light/fan system as a state automaton Z. It has four global states and four actions,

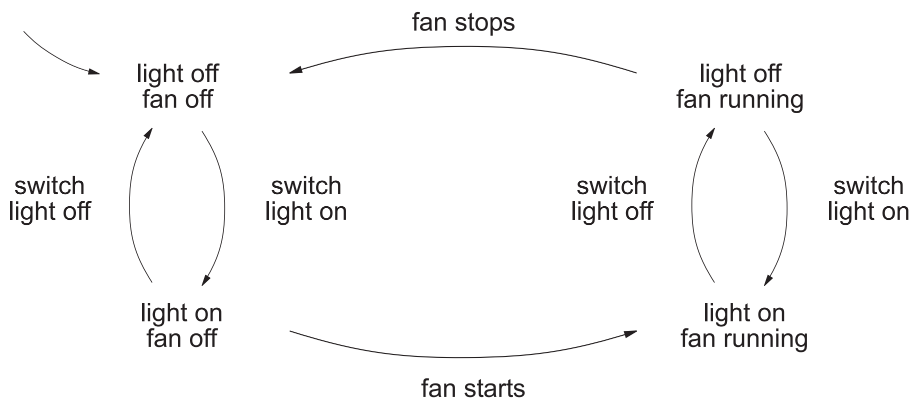


Figure 7.1: The light/fan system as a state automaton

two of which (switch light on and switch light off) can occur in two states each.

Figure 7.2 shows the system as an elementary system net N. It has four places describing the *local* states as well as four transitions, one for each action of the system.

The representation as a system net clearly describes the cause and effect of each action. switch light off, for example, can only occur if the light is on. The current state of the fan is irrelevant for this action. The fan itself, however, only starts if it is not running and the light is on at the same time.

In Fig. 7.2, we have modeled the switching of the light as *cold* transitions, since nobody is forced to use the light switch. The fan is a different matter. It has to react appropriately. The

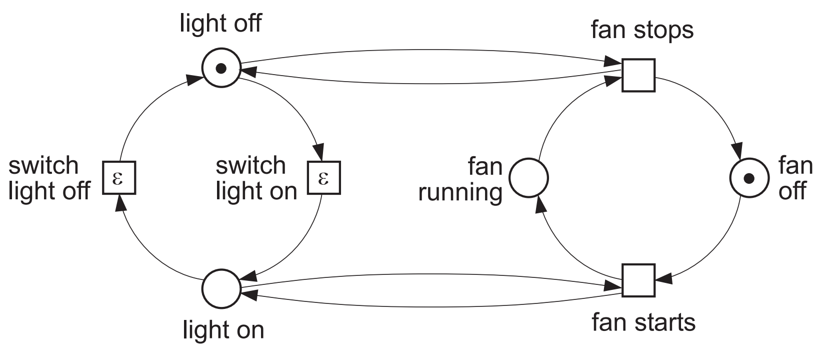


Figure 7.2: The light/fan system as an elementary system net

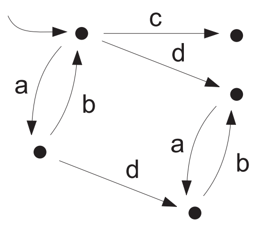


Figure 7.3: Abstract state automaton Z_1

difference between hot and cold transitions is not modeled in Fig. 7.1, and it is irrelevant for the rest of this chapter.

It is easy to determine that the system net N has the exact same behavior as the state automaton Z: one constructs the marking graph of N (Section 2.8) and asserts that it is identical to Z. The *synthesis problem of* Z is the search for a technique to derive N from Z.

7.2 The General Question of the Synthesis Problem

Each state of the light/fan state automaton in Fig. 7.1 is labeled with two conditions. These conditions form places in Fig. 7.2. In general, however, there is no information on the states of a state automaton Z. It is *abstract*, as in Fig. 7.3. Every edge of such an automaton is labeled with an *action*. The same action may occur on more than one edge.

The *synthesis problem* of any (abstract) state automaton Z is the search for a *distributed system* V that behaves exactly like Z. Instead of global states, only *local* states may appear in V. Every action appearing in Z has to be described completely and unambiguously through its effect on a few local states in V. To constructively solve this problem, we restrict the candidates for V to 1-bounded elementary system nets. In doing so, V behaves like Z if the marking graph G of V is isomorphic to Z. G is isomorphic to Z if every node k of G maps to exactly one node k' of Z such that:

- the initial marking of G is mapped to the initial state of Z,
- $h \xrightarrow{t} k$ is a step in G iff $h' \xrightarrow{t} k'$ is a step in Z.

To summarize: a system net N solves the synthesis problem of a given (abstract) state automaton Z if the marking graph G of N is isomorphic to Z.

There exist state automata Z whose synthesis problem cannot be solved by any 1-bounded elementary system net. An example is the state automaton R in Fig. 7.4.

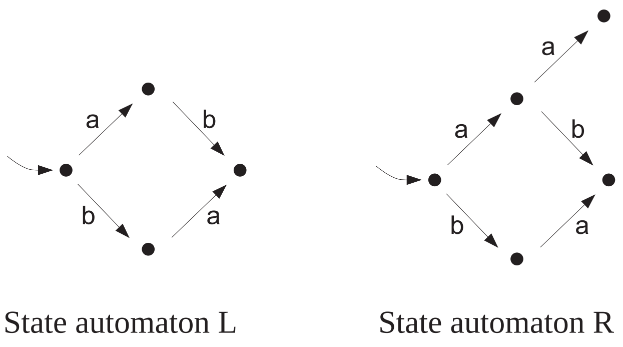


Figure 7.4: The synthesis problem of L is solvable; that of R is not solvable

7.3 Regions of State Automata

In the following, we will develop an algorithm that takes a state automaton Z and decides whether there exists a 1-bounded elementary system net N that solves the synthesis problem of Z. If it is solvable, a solution that is "as small as possible" is constructed. The construction does not take into account the difference between hot and cold transitions. State automata have an initial state but no final states.

To do this, for a given state automaton Z, a system N is constructed. If the marking graph G of N is isomorphic to Z, then N is the sought-after solution. Otherwise, a theorem guarantees that no solution exists.

First of all, we have to decide which characteristics of Z can be used for the construction of N. Apart from its structure as a directed graph, Z is characterized particularly by edges with the same labelings. After all, from the set of edges that are labeled with a symbol t, a *single* transition t has to be derived for N. Therefore, the starting nodes of those edges – if applicable, together with other nodes of Z – represent a place in ${}^\bullet t$.

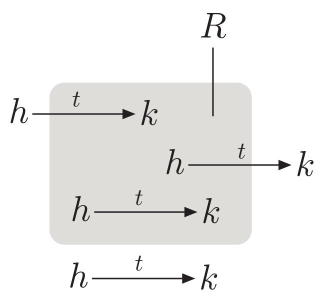
Put more generally, a set of nodes of Z can represent a place in N. To find such sets of nodes, we define the following terms:

Let Z be a state automaton, let R be a nonempty subset of its nodes and let

$$\pi : h \xrightarrow{t} k$$

be an edge of Z ("t-edge"). We define:

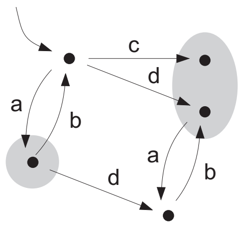
R *receives* π,	if $h \notin R$ and $k \in R$
R *dispatches* π,	if $h \in R$ and $k \notin R$
R *contains* π,	if $h \in R$ and $k \in R$



R is a *region of* Z if for each edge labeling t of Z:

- R receives either each or no t-edge and
- R dispatches either each or no t-edge.

region



no regions

minimal region

A region R of Z is *minimal* if no proper subset of R is a region of Z.

Figure 7.5 shows some of the regions of the state automaton Z_1 from Fig. 7.3. Figure 7.6 shows the minimal regions of Z_1.

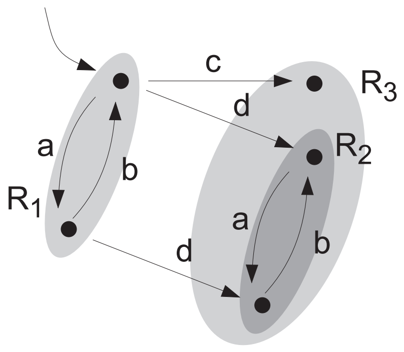
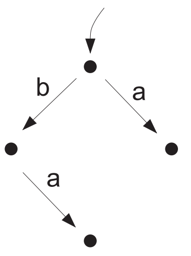


Figure 7.5: Three regions of the state automaton Z_1: R_1 and R_2 are minimal; R_3 is not.

7.4 The System Net of a State Automaton

Using the following procedure, we construct from a given state automaton Z an elementary system net N, the *system net of* Z:



state automaton Z_0

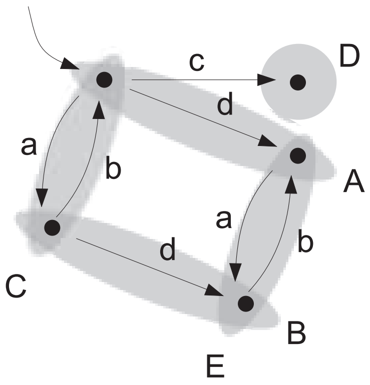
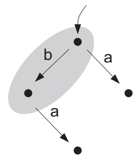
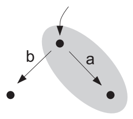


Figure 7.6: The minimal regions of Z_1

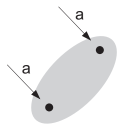
region A:



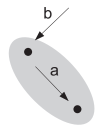
region B:



region C:

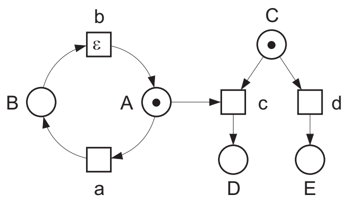


region D:



- each minimal region p of Z is a place of N;
- each edge labeling t occurring in Z is a transition of N;
- if a region p receives the t-edges of Z, then (t, p) is an arc of N;
- if a region p dispatches the t-edges of Z, then (p, t) is an arc of N;
- if a region p contains all the t-edges of Z, then (t, p) and (p, t) are arcs of N
- if the initial state of Z lies within a region p of Z, the place p of N holds a token.

Figure 4.1 shows the system net of the state automaton Z_1 shown in Fig. 7.3, with its minimal regions $\mathsf{A}, \ldots, \mathsf{E}$ shown in Fig.7.6.



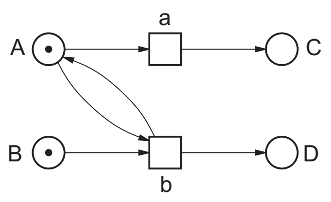
system net of a state automaton

7.5 The Solution to the Synthesis Problem

The following *Synthesis Theorem* is the basis for the solution to the synthesis problem:

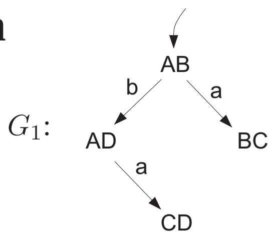
Theorem 1 (Synthesis Theorem)**.** *If the synthesis problem of a state automaton Z can be solved by a* 1*-bounded elementary system net, then the system net of Z is a solution.*

With this, it is possible to solve the general synthesis problem: one constructs the system net N of a given state automaton Z and from this the marking graph G of N. If Z and G are isomorphic, N obviously solves the synthesis problem of Z. Otherwise, the synthesis problem of Z cannot be solved by any 1-bounded elementary system net.



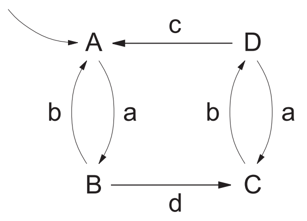
system net N_0 of Z_0

7.6 The Synthesis Problem of the Light/Fan State Automaton



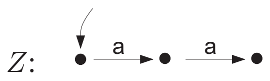
marking graph of N_0

To simplify the argument, we use an abridged version of the denotations of the light/fan system, as shown in Fig. 7.7.

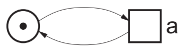


minimal regions:
$\{\mathsf{A}, \mathsf{B}\}, \{\mathsf{A}, \mathsf{D}\}, \{\mathsf{B}, \mathsf{C}\}, \{\mathsf{C}, \mathsf{D}\}$

Figure 7.7: Abridgment of Fig. 7.1



the system net of Z:



does not solve the synthesis problem of Z

This state automaton has four minimal regions. Following to the procedure explained above, the system net in Fig. 7.8 is constructed.

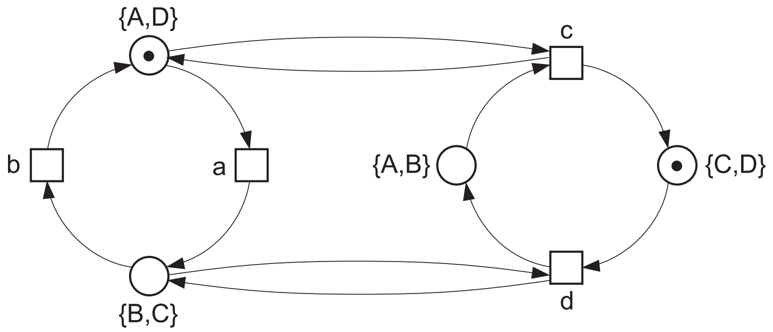


Figure 7.8: Solution to the synthesis problem of the state automaton in Fig. 7.7

The construction of its marking graph is left to the reader, as well as ascertaining that the graph is isomorphic to Fig. 7.7.

With this, Fig. 7.8 solves the synthesis problem of the state automaton in Fig. 7.7. Changing the abridged denotations in Fig. 7.8 back to their longer versions, in fact, results in the system net shown in Fig. 7.2.

Exercises

1. Show that in Fig. 7.4 the synthesis problem for L is solvable, but the one for R is not.

2. Solve the synthesis problem for the state automaton in Fig. 7.9.

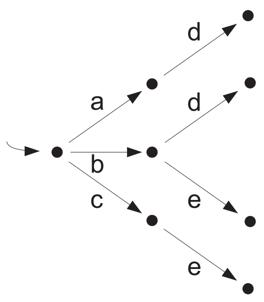


Figure 7.9: State automaton

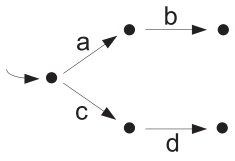


Figure 7.10: State automaton with easy synthesis problem

3. Why is the solution to the synthesis problem for the state automaton in Fig. 7.10 especially easy?

4. Is the elementary system net in Fig. 7.11 a solution to the synthesis problem of a state automaton?

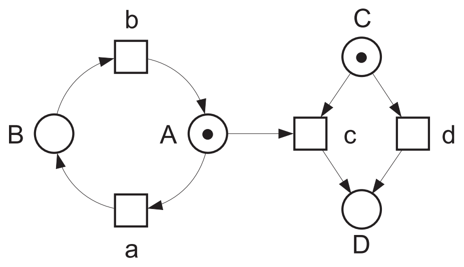


Figure 7.11: System net

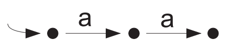


Figure 7.12: State automaton

5. The synthesis problem for the state automaton in Fig. 7.12 cannot be solved by any 1-bounded elementary system net. This is shown in the margin of Sect. 7.5. Construct an elementary system net whose marking graph is isomorphic to the state automaton in Fig. 7.12.

Further Reading

The solution to the synthesis problem belongs to those success stories in the field of Petri nets that emerged only slowly. Now, however, it belongs – in various versions – to the standard repertoire of most Petri net analysis tools.

As early as the late 1970s, Carl Adam Petri knew how to reverse calculate any 1-bounded elementary system net N from its marking graph G: a state set A of G forms a place in N if either all or no edges of G that have the same label either start or end in A. He did not consider this to be particularly interesting, however, because n places of N would result in 2^n nodes in G and thus 2^{2^n} state sets – an unmanageable situation.

In the mid-1980s, Ehrenfeucht and Rozenberg [21] solved the synthesis problem of loop-free, 1-bounded elementary system nets as an example of use for their "2-Theory". To do this, they use *all* regions of a given state automaton for the construction of a system net. It was not until 1993 that Bernardinello [8] confirmed the assumption that the minimal regions suffice. Only with this restriction does the region theory become usable in practice. By using only the minimal regions, nets of manageable sizes are constructed.

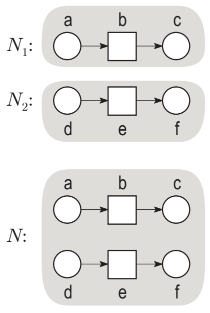
Many authors took part in generalizing the 1-boundedness in Theorem 1 to generic elementary system nets and in including even inhibitor edges. With this, the synthesis problem of much more general graphs becomes solvable. The most important authors include Badouel, Bergenthum, Busi, Cortadella, Darondeau, Desel, Gunther, Hoagens, Juhás, Kishinevsky, Kleijn, Kontratyev, Lavagno, Lorenz, Mauser, Mukund, Pietkiewicz-Koutny, Pinna, Thiagarajan, van der Aalst and Yakovlev. An overview of these developments can be found in [45]. Also very interesting for practical purposes is a reduction of the criteria for the solution to the synthesis problem: A system net N solves a reduced synthesis problem of a state automaton Z if the marking graph G of N is not necessarily isomorphic to Z, but if G and Z are bisimular or fulfill some other simulation relation. In [12], for instance, only a bisimulation is required between the state automaton Z and the marking graph of the synthesized net N. Additionally, the transitions of N may be labeled. Then, one action of Z can be implemented by several transitions of N. Lastly, it is possible to require special characteristics of N, for instance, the free-choice characteristic from Chapter 16. [78] solves the synthesis problem of other, liberal simulation relations and also takes distributed runs into account. Numerous software tools for the analysis of Petri nets offer a synthesis module, for instance, the tool Petrify [13].

Put intuitively, the synthesis problem seeks to construct a distributed system from the observation of its sequential behavior. The fact that this is possible marks Petri nets as a "very natural modeling technique" for distributed systems.

Composition of Nets — Chapter 8

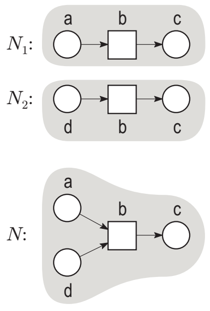
It is a general principle of software engineering to assemble small systems into larger ones. This process is called *composition*. Here we will consider the case of nets with *interfaces*. Two such nets can be joined together at their interfaces. Interfaces used for asynchronous, directed communication are an important special case. Such *open nets* are particularly natural: each Petri net can be broken down unambiguously into its set of smallest open nets.

8.1 Nets with Interfaces



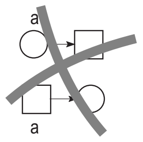
Two nets $N_1 = (P_1, T_1, F_1)$ and $N_2 = (P_2, T_2, F_2)$ can always also be conceived of as a single net N. Each place, each transition and each arc of N is a place, a transition or an arc of either N_1 or N_2:

$$N =_{def} (P_1 \cup P_2, T_1 \cup T_2, F_1 \cup F_2). \tag{1}$$



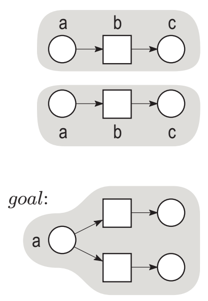
Of interest are those elements that belong to both N_1 and N_2: A place $p \in P_1 \cup P_2$ consistently appears only once in N. The same is true for shared transitions and arcs.

The case of an element appearing in N_1 as a place and in N_2 as a transition or arc does not appear in any sensible modeling task. We will not consider such cases here.



A composition of the form (1) is not sufficiently flexible. An example of this is a net N for which there exist two instances (copies) that should be composed in such a way that only a few specified elements are merged (i.e., *identified*) and the others are kept separate.

To express this, we distinguish between *interface* elements and *inner* elements of a net N. The difference becomes im-

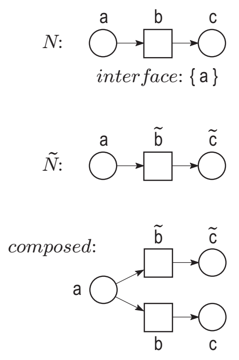


portant during the composition of N with a net N' if N and N' share one or more elements: if an element e belongs to the interfaces of both N and N', then during the composition, e should be identified according to (1). Otherwise, contrary to (1), the two examples of e should be kept separate. Technically, this is achieved by means of a new element $\tilde{e}$ that does not belong to either N or N'. At first, the net $\tilde{N}$ is constructed from N by

interface

$$\text{substituting } \tilde{e} \text{ for } e \tag{2}$$

in N. For the composition of N and N', the nets $\tilde{N}$ and N are now joined according to (1).



A net $N = (P, T, F)$ together with an interface

$$I \subseteq P \cup T$$

is an *interface net*.

Graphically, the interface of an interface net is often placed on or outside of a border around N. Figure 8.1 shows three interface nets modeling the behavior of a customer, a salesperson and his stock. Their inner elements model appropriate control flows.

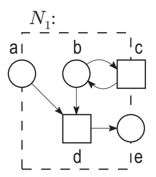
interface net

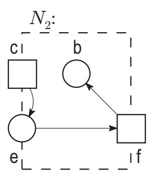
composition of interface nets

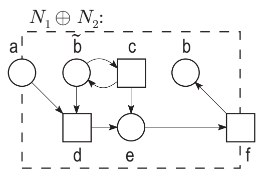
The composition of two interface nets $N_1 = (P_1, T_1, F_1)$ and $N_2 = (P_2, T_2, F_2)$ with the interfaces I_1 and I_2 is again an interface net, $N_1 \oplus N_2$. In accordance with (2), we assume here that no inner element of N_1 or N_2 appears in both nets. Then (1) describes the net structure of $N_1 \oplus N_2$. The shared interface elements of N_1 and N_2 become inner elements of $N_1 \oplus N_2$; the rest comprise the interface I of $N_1 \oplus N_2$:

$$I =_{def} (I_1 \cup I_2) \backslash (I_1 \cap I_2).$$

Figure 8.2 shows the composition of two of the interface nets shown in Fig. 8.1.







During the composition of two nets, those two nets are interchangeable ($\oplus$ is commutative). If, in the case of multiple interface nets, each shared element appears at most in two interfaces, the order in which the nets are composed is also interchangeable ($\oplus$ is occasionally associative).

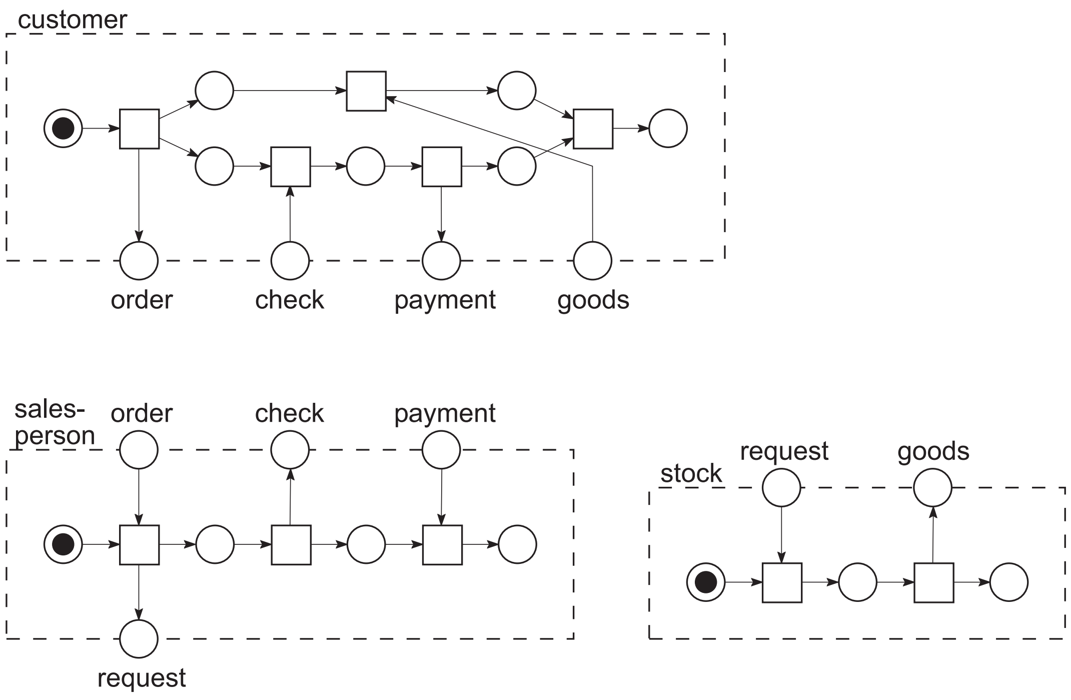


Figure 8.1: Three interface nets

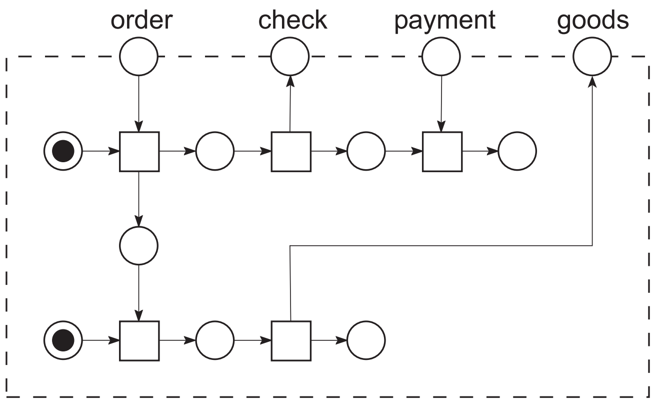


Figure 8.2: Composition salesperson $\oplus$ stock

Theorem 2 (Composition Theorem for Interface Nets)**.** *For $i = 1, 2, 3$, let N_i be interface nets with the interfaces I_i.*

(a) $N_1 \oplus N_2 = N_2 \oplus N_1$.

(b) *If $I_1 \cap I_2 \cap I_3 = \emptyset$, then $(N_1 \oplus N_2) \oplus N_3 = N_1 \oplus (N_2 \oplus N_3)$.*

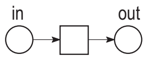
Both propositions follow from the known laws of set operations.

associative set of interface nets

A set M of interface nets is called *associative* if no element appears in the interfaces of three or more of the nets in M. The term "associative" follows from Theorem 2(b). The three interface nets in Fig. 8.1 form an associative set.

8.2 Communicating Nets

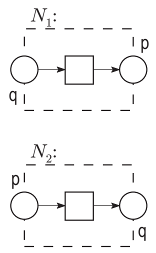
Shared places in the interfaces of two nets are often used for asynchronous, directed communication. A place p in the interface of a net N is an *in-place of* N if no arc ends in p, that is, if ${}^\bullet p = \emptyset$. Likewise, p is an *out-place of* N if $p^\bullet = \emptyset$. OUT$_N$ and IN$_N$ respectively denote the sets of the *out*- and *in-places* of N.



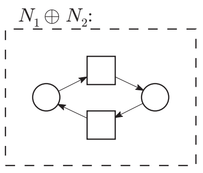
communicating interface nets

Two interface nets N_1 and N_2 *communicate via an interface place* p if N_1 puts tokens into p and N_2 takes tokens from p, or vice versa: if N_2 puts tokens into p and N_1 takes tokens from p, that is, if

$$p \in (\text{OUT}_{N_1} \cap \text{IN}_{N_2}) \cup (\text{OUT}_{N_2} \cap \text{IN}_{N_1}).$$



Communication via p and q



N_1 and N_2 *communicate* if for the interfaces I_1 and I_2 of N_1 and N_2 each $x \in I_1 \cap I_2$ is a place via which N_1 and N_2 communicate. The three nets in Fig. 8.1 communicate pairwise with each other.

In practice, sets of pairwise communicating interface nets are common: each interface place is shared by at most two interface nets. For such sets, the following holds.

Theorem 3 (Associativity Theorem)**.** *Let M be a set of pairwise communicating interface nets. Then M is associative.*

To prove this, one first shows by contradiction that each shared element x appears at most in two interfaces I_i of nets N_i in M: if $x \in I_1 \cap I_2 \cap I_3$, then x is an *in*-place or an *out*-place of at least two of the three nets N_1, N_2, N_3. Those two nets do not communicate! Thus the proposition follows from Theorem 2.

8.3 Unambiguous Decomposition into Open Nets

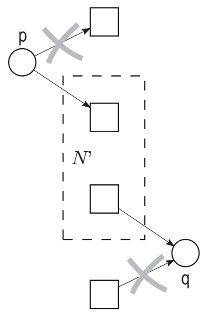
Communicating nets often only have *in*- and *out*-places in their interfaces, that is, no transitions and no places with transitions in both their pre- *and* post-sets. Such nets are often called *open*. The nets in Figures 8.1 and 8.2 are all open. Open nets are a very natural construction: each arbitrary net can be decomposed into a unique set of smallest, pairwise communicating open nets.

For a net $N = (P, T, F)$ and subsets $P' \subseteq P$, $T' \subseteq T$ and $F' \subseteq F$ of places, transitions and arcs of N, the net

$$N' = (P', T', F')$$

is a *subnet of* N if N' inherits all relevant arcs of N, that is, if for each arc $(x, y) \in F$ with $x, y \in P' \cup T'$:

$$(x, y) \in F'.$$



Now let IN be the set of *in*-places and OUT the set of *out*-places of N' such that in N for each $p \in$ IN and each $q \in$ OUT:

$$p^\bullet \subseteq T' \text{ and } {}^\bullet q \subseteq T'.$$

Then N' together with the interface

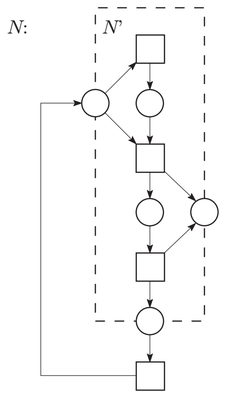
$$I =_{def} \text{IN} \cup \text{OUT}$$

is an *open subnet of* N. If N' itself does not have any proper open subnets, then N' is a *minimal open subnet of* N.

open subnet

minimal open subnet

The two open nets salesperson and stock in Fig. 8.1 are open subnets of the net in Fig. 8.2. Each minimal open subnet



N is a minimal open subnet.

of the examples in Figures 8.1 and 8.2 consists of a single transition t together with its pre- and its post-set ${}^\bullet t \cup t^\bullet$ as interface.

The minimal open subnets of N form an associative set and can thus be composed in arbitrary order. Above all, however, their composition again results in N itself:

Theorem 4 (Theorem on Minimal Open Subnets)**.** *Let N be a net and let $N_1, \ldots, N_k$ be the minimal open subnets of N. Then:*

(a) $\{N_1, \ldots, N_k\}$ *is associative.*

(b) $N = N_1 \oplus \ldots \oplus N_k$.

Property (a) follows from the Associativity Theorem and the observation that the N_i are communicating pairwise. The proof of (b) utilizes an equivalence relation $\sim$ on the transitions of N: let $\sim$ be the strongest equivalence such that $t \sim u$ if $t^\bullet \cap u^\bullet = \emptyset$ or ${}^\bullet t \cap {}^\bullet u = \emptyset$. Each equivalence class K of $\sim$ together with the places in ${}^\bullet t \cup t^\bullet$ of all transitions $t \in K$ form a net N_K. Together with the *in*- and *out*-places of N_K as interface I, the net N_K is a minimal open subnet of N.

Exercises

1. Model the cookie vending machine shown in Fig. 3.1 as well as one of its customers as open nets.
   Hint: The three transitions insert coin, take packet and return coin in Fig.3.1 describe (in parts) the behavior of the customer. Thus, the interface between the vending machine and the customer consists of transitions there. The behavior of the customer is not explicitly modeled. Construct the interface between the vending machine and the customer as places for the coin slot, the cookie compartment and a slot for returned coins.

2. Based on Fig. 8.1, construct the nets

   (a) customer $\oplus$ salesperson,

   (b) customer $\oplus$ stock,

   (c) customer $\oplus$ salesperson $\oplus$ stock.

3. Prove Theorem 2.

Further Reading

In this chapter, we have defined the composition of two nets via shared elements. Alternatively, it is possible to assign labels to elements and to define composition via equally labeled elements. Most composition operations given in the literature are ultimately based on one of these two principles. This includes the mathematically challenging pushout construction found in category theory. Open nets are typically used to model service-oriented architectures [2]. Interfaces with transitions are covered in [9]. An overview on different composition operations is given in [34]. Associativity is important for the composition of many variants of nets [68].

Part II

Analysis Methods

Part I showed how to model a system as a Petri net. Part II is now concerned with the *analysis* of such models, that is, determining important properties. Such an important property often makes a statement about each and every reachable marking M of a system net. For instance, in each reachable marking M of the cookie vending machine in Fig. 2.1, the following holds:

$$M \text{ marks either } \mathsf{A} \text{ or } \mathsf{D}$$

and

$$M \text{ consists of at most 9 tokens.}$$

Such a system property is a *state property* of the system net. There are a series of specific analysis techniques for state properties. This topic is discussed in Chapters 9–13.

There are some very common properties that are often asked about in a system net N: does N terminate? Can N reach a marking that no longer enables any transitions? Can each transition always become enabled again? From each reachable marking, is it possible to reach the initial marking again? Such questions generally cannot be answered, or at least not very efficiently. Some of them can be characterized in terms of graph theory by means of a marking graph (Sect. 2.8) or approximated by means of a derived graph, the covering graph. Chap. 14 discusses these graphs.

Two questions are particularly important for a system net N and a marking M:

- Is M reachable from the initial marking?
- Is M definitely reached from the initial marking?

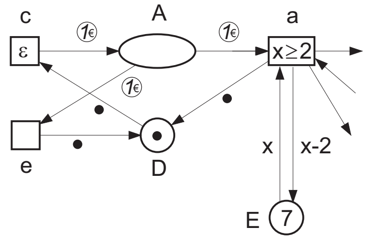
These two questions are covered in Chaps. 15 and 16. Chapters 17, 18 and 19 introduce special net classes and describe how their structures can be utilized in their analysis.

State Properties Chapter 9

Important properties of a system pertain to the system's reachable states. In a system net N, those are the reachable markings. We represent such a *state property* E of N with an expression a that contains as variables the places of N. If, for a marking M, each place p in a is then replaced with $M(p)$, the expression can be evaluated to either *true* ("E holds in M") or *false* ("E does not hold in M").

State properties are often expressed as linear equations or inequalities. This chapter covers the *form* and the validity of such equations and inequalities. How to *prove* their validity is covered in the following chapters. As an example, we start out with properties of the cookie vending machine. The form of linear equations and inequalities is very simple and is explained in the second section. This is followed by further examples, and finally by ideas about variants of linear equations and inequalities.

9.1 Equations and Inequalities of the Cookie Vending Machine



For the model of the cookie vending machine N in Fig. 2.1 and its respective initial marking M_0, the following holds:

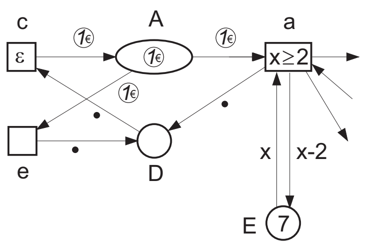
$$M_0(\mathsf{A}) + M_0(\mathsf{D}) + M_0(\mathsf{E}) = [\,] + [\bullet] + [7] = [\bullet, 7]. \qquad (1)$$

In particular, the *quantity* of tokens can be derived from this:

$$|M_0(\mathsf{A})| + |M_0(\mathsf{D})| + |M_0(\mathsf{E})| = |[\bullet, 7]| = 2. \qquad (2)$$

Thus, the places A, D and E initially hold a total of two tokens. After the step $M_0 \xrightarrow{\mathsf{c}} M_1$:

$$M_1(\mathsf{A}) + M_1(\mathsf{D}) + M_1(\mathsf{E}) = [\text{1€}] + [\,] + [7] = [\text{1€}, 7]. \qquad (3)$$



The places now hold different tokens, but the quantity of tokens remains the same:

$$|M_1(\mathsf{A})| + |M_1(\mathsf{D})| + |M_1(\mathsf{E})| = |[\textcircled{1}, 7]| = 2. \tag{4}$$

Now Eqs. (2) and (4) can be combined. For each of the two markings $M = M_0$ and $M = M_1$, the following holds:

$$|M(\mathsf{A})| + |M(\mathsf{D})| + |M(\mathsf{E})| = 2. \tag{5}$$

In fact, Eq. (5) holds not only for the markings $M = M_0$ and $M = M_1$, but for *each reachable marking* M of N. The places A, D and E thus always hold two tokens. Equation (5) *holds in* N, and we write as shorthand:

$$|\mathsf{A}| + |\mathsf{D}| + |\mathsf{E}| = 2. \tag{6}$$

Upon closer inspection, it becomes evident that A and D together *always* hold one token and E another. Thus, for each reachable marking M:

$$|M(\mathsf{A})| + |M(\mathsf{D})| = 1 \text{ and } |M(\mathsf{E})| = 1.$$

As shorthand, we write:

$$|\mathsf{A}| + |\mathsf{D}| = 1, \text{ and } |\mathsf{E}| = 1. \tag{7}$$

The relation between A and D can be described even more precisely: in each reachable marking M, either A holds a euro or D holds a black dot. To express this observation as an equation, we count the tokens in D; that is, we calculate $|M(\mathsf{D})|$, and then generate the respective amount of euro coins:

$$|M(\mathsf{D})| \cdot [\textcircled{1}]. \tag{8}$$

Because $M(\mathsf{D}) = [\bullet]$ or $M(\mathsf{D}) = [\ ]$, it follows that $|M(\mathsf{D})| = 1$ or $|M(\mathsf{D})| = 0$. The expression (8) thus either evaluates to $[\textcircled{1}]$ or $[\ ]$. If $M(\mathsf{A})$ (that is, $[\ ]$ or $[\textcircled{1}]$, respectively) is added to this, the result is $[\textcircled{1}]$. Thus, for each reachable marking M:

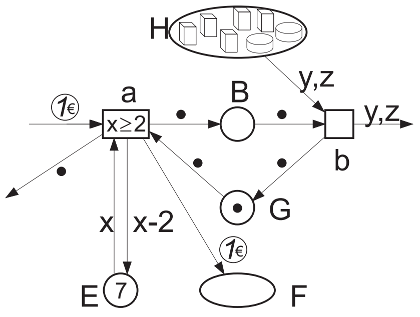
$$M(\mathsf{A}) + (|M(\mathsf{D})| \cdot [\textcircled{1}]) = [\textcircled{1}]$$

or simply

$$\mathsf{A} + (|\mathsf{D}| \cdot [\textcircled{1}]) = [\textcircled{1}].$$

Intuitively speaking, the token in D corresponds to a euro in A.

The relation between F and H is particularly interesting. In any reachable marking M, each euro in F should correspond to two packets in H, thus $2 \cdot |M(\mathsf{F})| + |M(\mathsf{H})| = 7$. However, this does not hold if a signal is pending in B. In this case, the signal neutralizes one euro in F. So actually $2 \cdot |M(\mathsf{F})| - 2 \cdot |M(\mathsf{B})| + |M(\mathsf{H})| = 7$, or simply



$$2|\mathsf{F}| - 2|\mathsf{B}| + |\mathsf{H}| = 7 \tag{9}$$

holds in N.[1] Furthermore, the equation

$$2|\mathsf{B}| + 2|\mathsf{G}| = 2$$

obviously holds in N and can be added to Eq. (9), which yields:

$$2|\mathsf{F}| + 2|\mathsf{G}| + |\mathsf{H}| = 9.$$

This equation can be interpreted as a "stream" of tokens, flowing from H via G to F.

Cardinality and multiplicity are apparently important for the formulation of valid linear equations. However, other functions are often needed as well. An example is the relation between the numeral value w of the token in E and the amount a of coins in F: the sum $w + 2a$ is always 7. To express this relation as an equation, let *cont* be a function that, for each singleton multiset $[n]$ with $n \in \mathbb{N}$, returns the value n. It is thus defined as

$$cont([n]) =_{\text{def}} n.$$

in the initial marking M_0 in Fig. 2.1:

$cont(M_0(\mathsf{E})) = cont([7]) = 7$
$2 \cdot |M_0(\mathsf{F})| = 2 \cdot |[\,]| = 2 \cdot 0 = 0$

Then:

$$cont(\mathsf{E}) + 2|\mathsf{F}| = 7. \tag{10}$$

$2 \cdot |M_0(\mathsf{F})| + |M_0(\mathsf{H})| =$
$2 \cdot |[\,]| + |[□, □, □, □, □, ⬭, ⬭]| =$
$2 \cdot 0 + 7 = 7 \leq 9$

In addition to equations, there are also some interesting *inequalities* holding in N, for instance

$$2|\mathsf{F}| + |\mathsf{H}| \leq 9 \text{ and} \tag{11}$$

$$cont(\mathsf{E}) \leq |\mathsf{H}|. \tag{12}$$

$cont(M_0(\mathsf{E})) = cont([7]) = 7 \leq 7 =$
$|[□, □, □, □, □, ⬭, ⬭]| = |M_0(\mathsf{H})|$

$cont(\mathsf{E}) + 2|\mathsf{F}| = 7$

[1] As is customary, the product is henceforth written without the dot.

9.2 Valid Equations

The most general form of a linear equation of a system net N is

equation

$$f_1(p_1) + \ldots + f_k(p_k) = a, \tag{13}$$

where $p_1, \ldots, p_k$ are the places of N. They are the *variables* of the equation. $f_1, \ldots, f_k$ are functions of the form

$$f_i : \mathcal{M} \to A, \tag{14}$$

where $\mathcal{M}$ is the set of all multisets that can occur as markings of places of N. The set A is an arbitrary set, whose elements can be added. In particular, a is an element of A in Eq. (13).

A function f_i is often the identity function. In this case, the tokens in p_i are themselves used in the equation. f_i is also often the cardinality function, in which case the number of tokens in p_i is used in the equation. A is often the set $\mathcal{M}$ of multisets, as in Eqs. (1) and (3), or the set of natural numbers or integers, as in Eqs. (6), (7) and (10). Other variations will be introduced later.

An equation of the form (13) *is valid in a marking M of N* if

$$f_1(M(p_1)) + \ldots + f_k(M(p_k)) = a. \tag{15}$$

An equation *is valid in N* if it holds in each reachable marking of N. If a subtraction is defined on the set A, then "$-$" may also occur in place of "$+$" in Eq. (13), and equations may be transposed in the usual way. In the model of the cookie vending machine in Fig. 2.1, for instance

$$|\mathsf{H}| - 2|\mathsf{B}| + 2|\mathsf{F}| = 7, \text{ and}$$

$$|\mathsf{H}| = cont(\mathsf{E}) + 2|\mathsf{B}|.$$

9.3 Example: Dining Philosophers

The system of five thinking and dining philosophers was formulated by E.W. Dijkstra in the mid-1960s as an illustration of a synchronization problem [19]. Five philosophers sit around a table. Initially, between each two neighboring philosophers lies a single chopstick. As an example showing the role of functions (and the multiple occurrence of a variable) in valid equations, Fig. 9.1 shows this system in its initial state: all five philosophers $p_1, \ldots, p_5$ are thinking, no philosopher is dining and all five chopsticks $g_1, \ldots, g_5$ are available. Obviously, a philosopher needs two chopsticks to start dining (transition t). So in order to do this, a philosopher p_i picks up the chopstick g_i to his left and the chopstick g_{i-1} to his right (with $g_0 =_{def} g_5$). Two neighboring philosophers can thus never dine at the same time. When the philosopher has finished dining, he puts his chopsticks back (transition u) and again devotes himself to thinking.

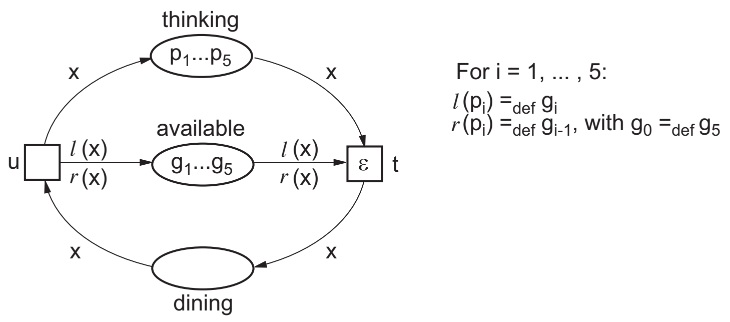


Figure 9.1: The five philosophers

The equation

$$\text{thinking} + \text{dining} = [p_1, \ldots, p_5]$$

is obvious: each philosopher is either thinking or dining. The equation

$$\text{available} + l(\text{dining}) + r(\text{dining}) = [g_1, \ldots, g_5] \qquad (16)$$

$l([p_1, p_3])$
$= [l(p_1), l(p_3)]$
$= [g_1, g_3]$

is more interesting: each dining philosopher corresponds to his two chopsticks. The functions l and r are only defined for single philosophers in Fig. 9.1. In Eq. (16), however, we apply them to *sets* of philosophers. We can generalize a function f from single elements to multisets as follows:

$$f([a_1, \ldots, a_n]) =_{\text{def}} [f(a_1), \ldots, f(a_n)].$$

A little less transparent is the intuitive meaning of the valid equation

$$l(\text{thinking}) + r(\text{thinking}) - \text{available} = [g_1, \ldots, g_5]. \qquad (17)$$

9.4 Valid Inequalities

Using functions $f_i : \mathcal{M} \to A$ as in Eq. (14), it is also possible to construct *inequalities* analogously to equations as shown in (13):

inequality

$$f_1(p_1) + \ldots + f_k(p_k) \leq a.$$

This only works, however, if an order $\leq$ is defined on the set A. Common examples of this are multisets (see Section 2.3) and the sets of natural numbers or integers. The relation "$\leq$" may also be replaced with "$<$", "$\geq$" or "$>$" (in principle, predicates that are even more general are possible). Equations (11) and (12) show examples of inequalities. The *validity* of such an inequality in a marking M of a system net N is defined analogously to Section 9.2.

valid inequality

From the definition of markings, it follows that for each system net N and each place p of N: $M(p) \geq [\,]$ for each marking M. Thus the *canonical inequality*

canonical inequality

$$p \geq [\,]$$

holds for each place p of N. From this, it also immediately follows that $|p| \geq 0$.

9.5 Equations and Inequalities of Elementary System Nets

For an elementary system net N, equations according to (13) often take the special form

$$n_1 \cdot p_1 + \cdots + n_k \cdot p_k = n_0,$$

where $n_0, \ldots, n_k$ are integers. Although a place p always holds a natural number $M(p)$ of tokens, the formalism has to operate

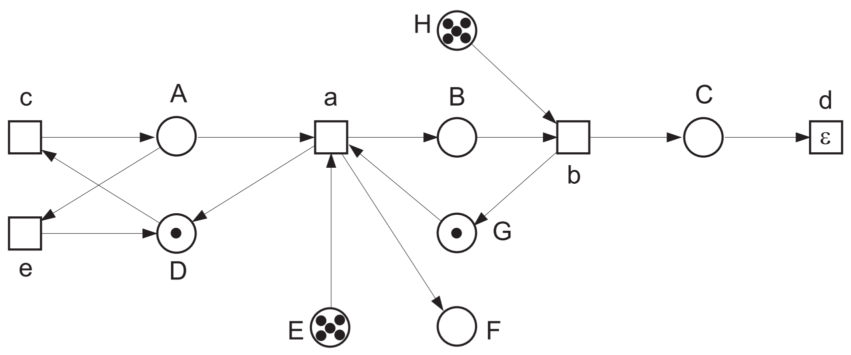


Figure 9.2: Abstract variant of the cookie vending machine

on integers to support subtraction. The factors n_i often take the value 1 and are not written. In the abstract variant of the cookie vending machine in Fig. 9.2, for instance, the following equations hold:[2]

$$\mathsf{A}+\mathsf{D}=1, \quad \mathsf{B}+\mathsf{G}=1, \quad \mathsf{E}+\mathsf{F}=5,$$
$$\mathsf{H}+\mathsf{G}+\mathsf{F}=6, \quad \mathsf{E}+\mathsf{B}-\mathsf{H}=0.$$

In addition, the following inequalities hold:

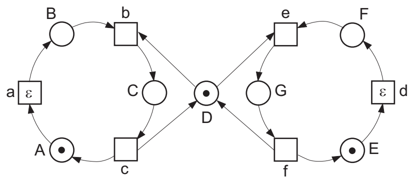
$$\mathsf{H}+\mathsf{C}\leq 5 \quad \text{and} \quad \mathsf{E}+\mathsf{B}+\mathsf{C}\leq 5.$$

In systems with communicating processes, the control flow of each individual process P can be represented as an equation of the form

$$p_1+\ldots+p_n=1, \tag{18}$$

meaning that the control flow of P runs through the places $p_1,\ldots,p_n$. An example of this is the system of mutual exclusion: the control flows of the processes l and r correspond to the equations

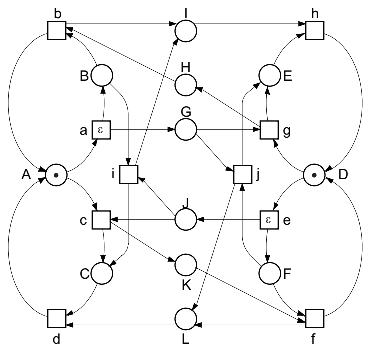
$$\mathsf{A}+\mathsf{B}+\mathsf{C}=1 \quad \text{and} \quad \mathsf{E}+\mathsf{F}+\mathsf{G}=1. \tag{19}$$



The property of mutual exclusion is described by the inequality

$$\mathsf{C}+\mathsf{G}\leq 1. \tag{20}$$

[2]For better readability, the cardinality bars are henceforth omitted in equations of elementary system nets.



The control flows of the two processes of the crosstalk algorithm are likewise described by the equations

$$\mathsf{A}+\mathsf{B}+\mathsf{C}=1 \quad \text{and} \quad \mathsf{D}+\mathsf{E}+\mathsf{F}=1.$$

An example of a factor $n_i > 1$ is the equation

$$2\mathsf{A}+\mathsf{B}+\mathsf{C}+\mathsf{D}+\mathsf{E}=2$$

in the elementary system net in Fig. 9.3.

A valid equation of the form (18) corresponds to a special subgraph of the respective Petri net. This subgraph consists of the places $p_1, \ldots, p_n$ used in the equation, and all adjacent transitions. In the mutual exclusion system with its valid equations (19), for instance, the places A, B and C, together with all adjacent transitions, form a circle (likewise, E, F and G). The valid equation

$$\mathsf{C}+\mathsf{D}+\mathsf{G}=1$$

with the branching place D forms a subgraph consisting of two circles. In general, the subgraph contains with each place p_i all its adjacent transitions and with each transition t exactly one place in ${}^\bullet t$ and one place in $t^\bullet$.

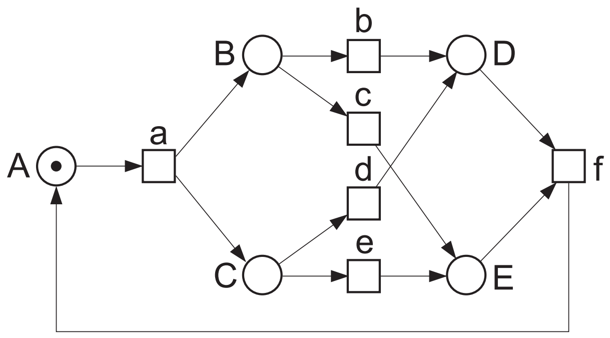


Figure 9.3: Technical example with $2\mathsf{A}+\mathsf{B}+\mathsf{C}+\mathsf{D}+\mathsf{E}=2$

9.6 Modulo Equations

The value range A of the functions f_i in Eq. (14) sometimes consists of the two values 0 and 1 with the *modulo-2 addition*, that is,

$$1 + 1 = 0.$$

Figure 9.4 shows an elementary system net N in which the modulo-2 equation

$$\mathsf{A} + \mathsf{B} + \mathsf{E} + \mathsf{F} = 0$$

holds, where $+$ is the modulo-2 addition. From this equation it follows, for instance, that no marking M with $M(\mathsf{A}) = 1$ and $M(\mathsf{B}) = M(\mathsf{E}) = M(\mathsf{F}) = 0$ is reachable in N.

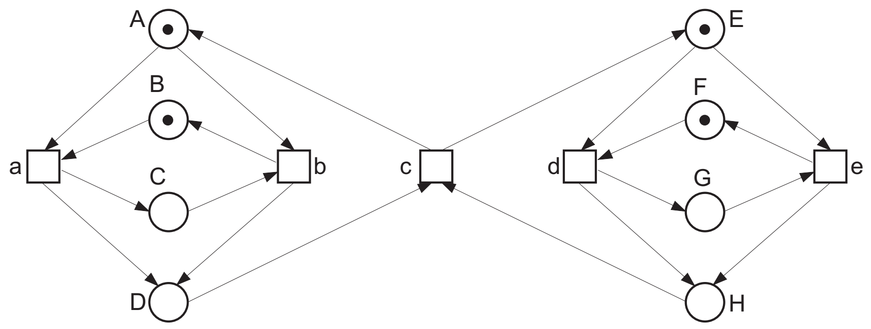


Figure 9.4: Elementary system net with the valid modulo-2 equation $\mathsf{A} + \mathsf{B} + \mathsf{E} + \mathsf{F} = 0$

As another example, in Section 13.10 we will use a modulo-2 equation to prove central properties of the kindergarten game from Section 4.6.

9.7 Propositional State Properties

It is often convenient to formulate state properties as logical expressions, as, for instance, for the cookie vending machine: "If a signal is pending, then the storage contains at least either a rectangular or a round packet." or "If the storage still contains all the packets of the initial marking, then signal and cash box hold equally many tokens (that is, at most one)." For the mutual exclusion system: "The left process is not critical or the key is not available." Or for the crosstalk algorithm (Section 3.4): "If the left process is waiting, then either sent_l or $\mathsf{confirmed}_r$ or $\mathsf{finished}_r$ contains a token."

To formulate such expressions, it is necessary to combine valid equations and inequalities (i.e., state properties) via logical operations. For two equations or inequalities α and β, we define the logical negation $\neg\alpha$ of α and the logical "and" $\alpha \wedge \beta$ of α and β in a marking M of a system net N as follows:

$$\neg\alpha \text{ holds in } M \text{ iff } \alpha \text{ does not hold in } M,$$

$$\alpha \wedge \beta \text{ holds in } M \text{ iff } \alpha \text{ and } \beta \text{ both hold in } M.$$

state property

With this, we can formulate *propositional state properties*. In the initial marking M_0 of Fig. 9.4, for instance, $(\mathsf{A} = 1) \wedge (\mathsf{B} = 1)$ as well as $\neg(\mathsf{C} = 1)$ hold. Other logical operations like $\alpha \vee \beta$ ("α or β") and $\alpha \to \beta$ ("if α, then β") can then, as usual, be derived as $\neg(\neg\alpha \wedge \neg\beta)$ and $\neg\alpha \vee \beta$, respectively.

A propositional expression α *holds in a system net* N if α holds in each reachable marking of N. The following expressions, for instance, hold

- in the cookie vending machine in Fig. 2.1:

$$\mathsf{B} \geq 1 \to (\mathsf{H} \geq [\square] \vee \mathsf{H} \geq [\ominus]) \text{ and} \tag{21}$$

$$\mathsf{H} = [\square, \square, \square, \square, \square, \ominus, \ominus] \to (|\mathsf{B}| = |\mathsf{F}| \wedge |\mathsf{B}| \leq 1), \tag{22}$$

- in the mutual exclusion system in Fig. 4.4:

$$\mathsf{C} = 0 \quad \vee \quad \mathsf{D} = 0, \tag{23}$$

- in the crosstalk algorithm in Fig. 3.8:

$$\mathsf{B} = 1 \to (\mathsf{G} = 1 \vee \mathsf{H} = 1 \vee \mathsf{L} = 1). \tag{24}$$

These examples support the observation that propositional state properties are often composed of very simple equations and inequalities. The following *dot notation* is therefore very convenient: for a system net N with a place p, an element $u \in U$ and a multiset $A \in \mathcal{M}$ of the universe U

$$p.u \text{ denotes } p \geq [u] \text{ and}$$

$$p.A \text{ denotes } p \geq A.$$

If a place p holds only black tokens, then

$$p \text{ denotes } p \geq [\bullet].$$

For a set $P = \{p_1, \ldots, p_n\}$ of places, we simply write

$$P \text{ or } p_1 \ldots p_n \text{ instead of } P = p_1 \wedge \ldots \wedge p_n. \qquad (25)$$

The above examples (21) through (24) then become

$$\mathsf{B} \rightarrow (\mathsf{H}.\square \vee \mathsf{H}.\ominus),$$

$$\mathsf{H}.[\square, \square, \square, \square, \square, \ominus, \ominus] \rightarrow (|\mathsf{B}| = |\mathsf{F}| \wedge |\mathsf{B}| \leq 1),$$

$$\neg\mathsf{C} \vee \neg\mathsf{D},$$

$$\mathsf{B} \rightarrow (\mathsf{G} \vee \mathsf{H} \vee \mathsf{L}).$$

In the mutual exclusion system in Fig. 4.3

$$\mathsf{AE} \rightarrow \mathsf{D} \text{ and } \mathsf{CE} \rightarrow \neg\mathsf{D}.$$

We occasionally use the notation

$$M \models \alpha \text{ or } N \models \alpha$$

to express that a property α holds in a marking M or in a system net N, respectively.

Theorem 5 (Validity Theorem for Propositional Properties)**.** *For a system net N and propositional properties α and β, the following hold:*

(a) *$N \models \alpha \wedge \beta$ iff $N \models \alpha$ and $N \models \beta$.*

(b) *If not $N \models \alpha$, then not necessarily $N \models \neg\, \alpha$.*

(c) *If $N \models \alpha$ or $N \models \beta$, then $N \models \alpha \vee \beta$.*

(d) *If $N \models \alpha$, then $N \models \beta \rightarrow \alpha$.*

(e) *If $N \models \neg\alpha$, then $N \models \alpha \rightarrow \beta$.*

Exercises

1. Which of the following equations and inequalities hold in the system net in Fig. 5.7?

   (a) 2A + B + C + D + E = 2

   (b) 2A + B + C + D + E + F + G = 3

   (c) F + G = 1

   (d) B + C = D + E

   (e) C + E + G $\leq$ 1

   (f) B + D + F $\leq$ 1

2. Construct a valid equation for each of the following system nets. The factor in front of each place must not equal 0.

   (a) Fig. 4.1

   (b) Fig. 4.20

   (c) Fig. 9.5

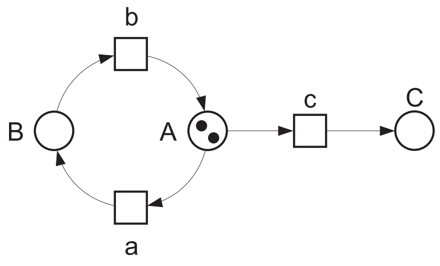


Figure 9.5: System net

3. Show that the following equations hold in the system net in Fig. 2.1.

   (a) $|\mathsf{H}| = cont(\mathsf{E}) + 2|\mathsf{B}|$

   (b) $|\mathsf{H}| + 2|\mathsf{G}| = cont(\mathsf{E}) + 2$

4. What property follows from the modulo-2 equation A + B + E + F = 0 for the number of tokens in the system net in Fig. 9.4?

5. Prove the five propositions of Theorem 5.

6. Show that the converse of each proposition (c)–(e) of Theorem 5 does not hold.

7. Which of the following statements hold in the system net N of the five philosophers in Fig. 9.1?

   (a) $N \models \mathsf{thinking.p}_i \to \neg\mathsf{dining.p}_i$

   (b) $N \models \mathsf{dining.p}_i \to \neg\mathsf{available.}l(\mathsf{p}_i) \;\wedge\; \neg\mathsf{available.}r(\mathsf{p}_i)$

   (c) $N \models \mathsf{thinking.p}_i \to \mathsf{available.}l(\mathsf{p}_i) \;\wedge\; \mathsf{available.}r(\mathsf{p}_i)$

   (d) $N \models \mathsf{available.g}_{i-1} \to \mathsf{thinking.p}_i$

   (e) $N \models \mathsf{available.g}_i \to \mathsf{thinking.p}_{i-1}$

   (f) $N \models \mathsf{available.g}_i \to \neg\mathsf{thinking.p}_{i-1}$

* 8. The following refers to this chapter's postscript "The Polite Philosophers" (next page):

   (a) Construct an "impolite" distributed run of the system net in Fig. 9.1.

   (b) Construct three more behavioral patterns of polite philosophers.

   (c) Generalize the case of $n = 5$ philosophers to an arbitrary number of philosophers. How many distributed runs exist? How many runs have distinct structures?

   (d) Expand the system net in Fig. 9.4 such that only the polite runs are generated.

Further Reading

Each modeling language formulates system properties in its own way: as a combination of and a compromise between what is needed in practice and what can be proven algorithmically. For Petri nets we use expressions with places as variables and, with them, formulate equations and inequalities. In the literature, such equations and inequalities are usually presented together with the corresponding techniques for proving their validity. Kurt Lautenbach [43] was the first to propose equations as a means to formulate properties of elementary system nets, and place invariants (Chapter 11) as a means to prove their correctness.

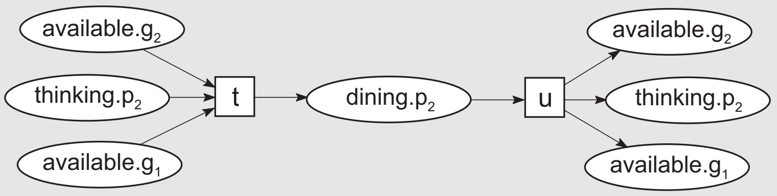
In addition to the linear and modulo equations presented in Sections 9.5 and 9.6, one could also discuss nonlinear equations. As an alternative to inequality (20), for instance, the mutual exclusion property could also be described via

$$\mathsf{C} \cdot \mathsf{G} = 0.$$

The Polite Philosophers

We return to the example of the five philosophers in Section 9.3, but only consider *polite* philosophers now: Two neighboring philosophers are *polite* to each other if they use their shared chopstick *alternatingly*. We then show that the system's distributed runs take clear and very regular structures if all neighbors are polite to each other.

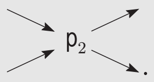
A partial distributed run in which the philosopher p_2 picks up his chopsticks, eats and then returns his chopsticks takes the form



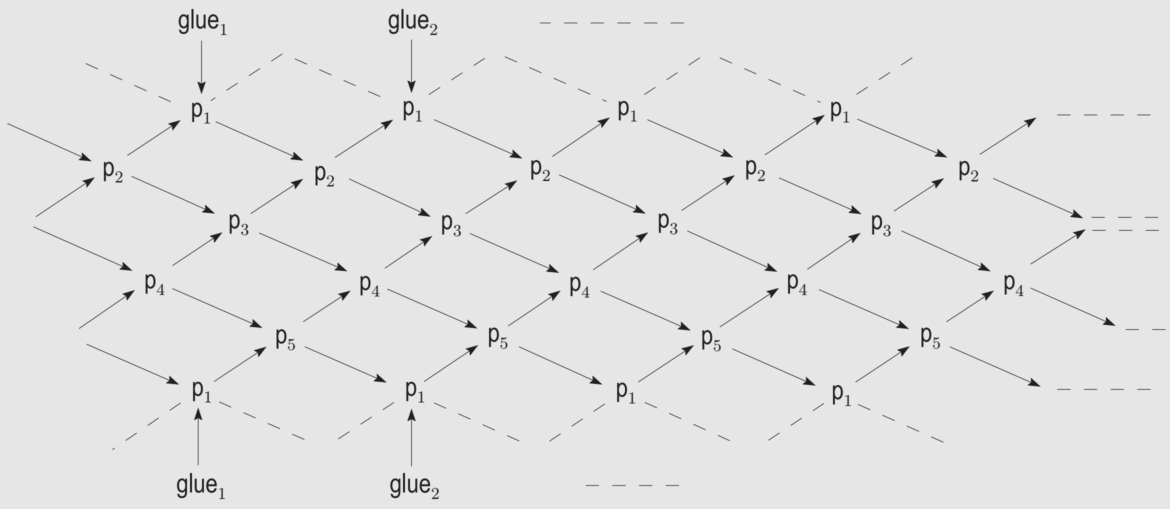
(26)

(the transitions t and u occur in the mode $x = 2$). Each "polite" distributed run of the system net in Fig. 9.1 is composed of such "meals" of the philosophers.

For convenience, partial runs of the form (26) can be simplified into



Then



(27)

shows a typical distributed run of the system of polite philosophers. Occurrences of p_1 lying directly on top of each other at the top or bottom edge are identical here.

Different initial markings eventually assume the behavioral pattern (27). It can be intuitively characterized by “While p_i is dining, p_{i+2} passes his right chopstick to p_{i+3}.” In a different pattern p_{i+3} passes his left chopstick to p_{i+2}. There exist two more patterns in which the polite philosophers dine “around” the table. Further details are described in [66].

Traps and Cotraps of Elementary System Nets

Chapter 10

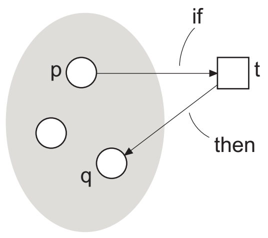
Traps form the basis of a particularly simple technique for proving inequalities. They take advantage of the structure of Petri nets, in particular, the simple rule for the occurrence of transitions and the alternating pattern of places and transitions. We will consider traps of elementary system nets first.

10.1 Traps of Elementary System Nets

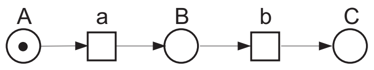
trap

Technically, a *trap* of an elementary system net N is a subset Q of the places of N such that for each transition t of N the following holds:

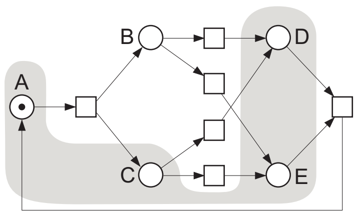
If there exists a place $p \in Q$ such that $p \in {}^\bullet t$,
then there also exists a place $q \in Q$ such that $q \in t^\bullet$. (1)



trap: $\{$p, q, $\dots\}$



marked trap: $\{$A,B,C$\}$
unmarked traps: $\{$B,C$\}$, $\{$C$\}$



trap: $\{$A,C,D,E$\}$

Intuitively speaking: “A transition that takes something out of Q also puts something back in.”

A trap Q of an elementary system net N is called *marked in a marking* M if there exists at least one place $q \in Q$ such that $M(q) \geq 1$. If $Q = \{q_1, \dots, q_r\}$ is a trap marked in M, then obviously

$$M(q_1) + \ldots + M(q_r) \geq 1. \tag{2}$$

For each step $M \xrightarrow{t} M'$ of N there exist only two possibilities now: either there exists a place $p \in Q$ such that $p \in {}^\bullet t$, in which case, according to (1), there also exists a place $q \in Q$ such that $q \in t^\bullet$. Then $M'(q) \geq 1$ and thus

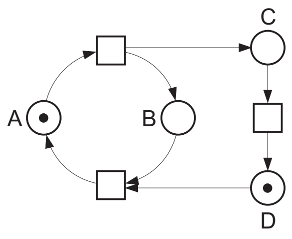
$$M'(q_1) + \ldots + M'(q_r) \geq 1. \tag{3}$$

initially marked trap

Or there exists no such place p. Then for each place $q \in Q$: $M'(q) \geq M(q)$, in which case (3) also holds. Hence, a marked trap will remain marked. Of particular interest are *initially marked* traps, that is, traps marked in the initial marking M_0:

Theorem 6 (Trap Theorem)**.** *Let N be an elementary system net with an initially marked trap $Q = \{q_1, \ldots, q_r\}$. Then the following inequality holds in N:*

$$q_1 + \ldots + q_r \geq 1. \tag{4}$$



$\mathsf{C} + \mathsf{D} \geq 1$, but $\{\mathsf{C},\mathsf{D}\}$ is not a trap

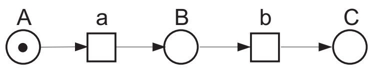
The converse is not generally true: a valid inequality of the form (1) does not imply that the set $\{q_1, \ldots, q_r\}$ is a trap.

10.2 Cotraps

cotrap

As a counterpart to a trap, a *cotrap* of an elementary system net N is a subset Q of the places of N such that for each transition t of N the following holds:

$$\begin{array}{l}\text{If there exists a place } p \in Q \text{ such that } p \in t^\bullet,\\ \text{then there also exists a place } q \in Q \text{ such that } q \in {}^\bullet t.\end{array} \tag{5}$$

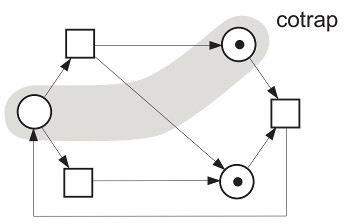


cotraps: {A,B,C}, {A,B}, {A}

Intuitively speaking: "A transition that puts something into Q also takes something out."

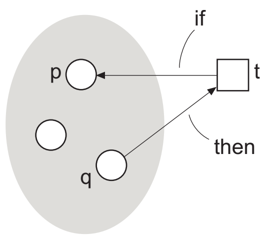
Now let $Q = \{q_1, \ldots, q_n\}$ be a cotrap that is unmarked in a marking M, that is

$$M(q_1) + \ldots + M(q_n) = 0. \tag{6}$$



Furthermore, let $M \xrightarrow{t} M'$ be a step. Then for each place $p \in {}^\bullet t : M(p) \geq 1$. This means, because of Eq. (6), that there cannot exist a place $q \in Q$ such that $q \in {}^\bullet t$. Hence, because of (5), there also cannot exist a place $p \in Q$ such that $p \in t^\bullet$. Then it follows from Eq. (6) that

$$M'(q_1) + \ldots + M'(q_n) = 0.$$



cotrap: {p, q, ...}

Hence, an unmarked cotrap will remain unmarked. If, during a run w, a cotrap Q loses its last token, then no transition in $Q^\bullet$ can occur for the remainder of w. In particular, a cotrap may already be unmarked in the initial marking:

Theorem 7 (Cotrap Theorem)**.** *Let N be an elementary system net with an initially unmarked cotrap $Q = \{q_1, \ldots, q_r\}$. Then the following equation holds in N:*

$$q_1 + \ldots + q_r = 0. \tag{7}$$

In particular, an unmarked cotrap occurs during a total system deadlock, that is, when a marking M does not enable a single transition. Of interest is the set R of the places that are unmarked in M: for each transition t there exists a place $p \in {}^\bullet t$ such that $p \in R$. Thus, R is an unmarked cotrap. The contraposition of this argument results in the following theorem.

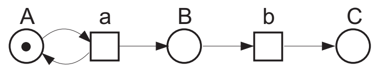
Theorem 8 (Theorem on Marked Cotraps)**.** *Let N be an elementary system net and let M be a marking of N that marks each cotrap of N. Then M enables at least one transition.*

10.3 The Trap/Cotrap Property

Sometimes the structure of a net guarantees that each reachable marking marks each cotrap, for instance if each cotrap contains an initially marked trap as a subset. We define a system net N to have the *trap/cotrap property* if:

trap/cotrap property

$$\text{Each cotrap } R \text{ of } N \text{ contains as a subset an initially marked trap.} \tag{8}$$



{A} is an initially marked trap and a subset of each cotrap

This definition leads us to:

Theorem 9 (Trap/Cotrap Theorem)**.** *Let N be an elementary system net that has the trap/cotrap property. Then each reachable marking M of N enables at least one transition.*

Figure 9.3 shows a technical example in which {A,B,C,D} is a cotrap that does not contain a trap. Thus, Theorem 9 does not apply. In fact, all tokens can accumulate in E, in which case no more steps are possible.

Exercises

1. Construct either a proof or a counterexample for each of the following propositions:

   (a) The union of two traps is a trap.

   (b) The intersection of two traps is a trap.

   (c) The union of two cotraps is a cotrap.

   (d) The intersection of two cotraps is a cotrap.

2. Show that in the system net in Fig. 10.1 each cotrap is also a trap. How does this change if the loop between a and C is deleted?

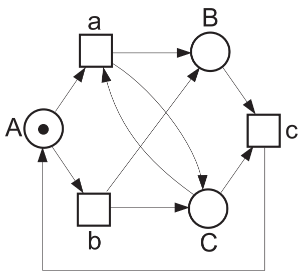


Figure 10.1: System net

3. (a) Show that the system net in Fig. 10.2 has the trap/cotrap property.

   (b) Which property follows from the observation in (a)?

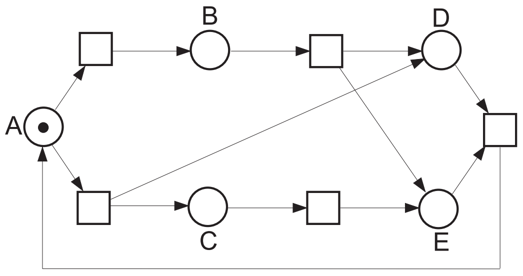


Figure 10.2: System net

* 4. Let N be an elementary system net and let $Q = \{q_1, ..., q_n\}$ be a set of the places of N such that

$$q_1 + ... + q_n = 1.$$

Prove or disprove the following propositions.

   (a) Q is a trap.

   (b) Q is a trap if each transition of N is enabled at least once.
   (c) Replace the above equation with the inequality

$$q_1 + ... + q_n \geq 1.$$

   Prove or disprove propositions (a) and (b) for this case.

5. Is the set P of all places of a system net a trap? Is P a cotrap?

Further Reading

Traps and cotraps have been known since the 1970s [33]. The name “trap” (German, “Falle”; French, “siphon”) directly describes the intuitive meaning of this construction. Cotraps originally came to be known as “deadlocks” or sometimes “siphons” in the literature – a confusing terminology. In modeling practice, traps occur much more often than cotraps.

In [44] Lautenbach describes linear-algebraic characterizations of and corresponding algorithms for traps and cotraps (called “deadlocks” there). Of particular interest for practical applications are minimal (smallest) traps. How to calculate them efficiently is covered in [80].

To increase efficiency, Wegrzyn and her coauthors [79] utilize satisfiability algorithms found in propositional logic. The paper also describes a range of other procedures for calculating traps and cotraps.

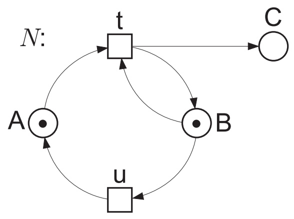
Place Invariants of Elementary System Nets

Chapter 11

Place invariants are the most important analysis technique for system nets. They take advantage of the *constant effect* of transitions: each time a transition t occurs in the mode β, the same multisets are moved. The effect of (t, β) is *linear*. For instance, if the places in ${}^\bullet t$ hold twice as many tokens, then t may occur twice as many times. The mathematics for linear behaviors is the well-known linear algebra with vectors, matrices and systems of equations. In fact, many aspects of Petri nets can be represented and calculated with these structures. We start with linear-algebraic notation for elementary system nets.

11.1 Vector Representation for Elementary System Nets

Chapter 3 defines a marking M of an elementary system net N as a mapping $M : P \longrightarrow \mathbb{N}$ that maps each place p of N to a natural number. It is very easy to impose an order on the places of a system net, for instance, by means of indices of the form $p_1, \ldots, p_k$ or by alphabetical order. Then M can be written as the column vector



$$\underline{M} = \begin{pmatrix} a_1 \\ \vdots \\ a_k \end{pmatrix}$$

$$\underline{M_0} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

where a_i is the number of tokens in the ith place.

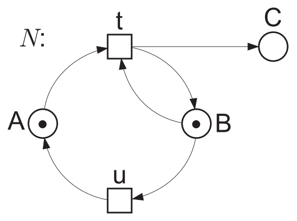
vector representation of markings

Likewise, each transition t can be written as a vector

$$\underline{t} = \begin{pmatrix} z_1 \\ \vdots \\ z_k \end{pmatrix}$$

$$\underline{\mathsf{t}} = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \quad \underline{\mathsf{u}} = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$$

vector representation of transitions



$\underline{M_1} = \underline{M_0} + \underline{t}$:

$$\begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$$

$\underline{M_2} = \underline{M_0} + \underline{u}$:

$$\begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$$

where for $i = 1, \ldots, k$

$$z_i =_{def} \begin{cases} -1, & \text{if } p_i \in {}^\bullet t \text{ and } p_i \notin t^\bullet, \\ +1, & \text{if } p_i \in t^\bullet \text{ and } p_i \notin {}^\bullet t, \\ 0, & \text{otherwise.} \end{cases}$$

It is generally not possible to reconstruct from the vector $\underline{t}$ the input and output places ${}^\bullet t$ and $t^\bullet$, respectively: $z_i = 0$ if no arc connects p_i and t, but also if there exists a loop between p_i and t.

With the above vectors and the classic definition of the sum of vectors, for each step $M \xrightarrow{t} M'$ of N:

$$\underline{M'} = \underline{M} + \underline{t}. \tag{1}$$

This is called the *vector representation* of steps.

11.2 The Matrix $\underline{N}$

matrix representation of a net

It is equally easy to impose an order of the form $t_1, \ldots, t_l$ on the transitions of N. Then the column vectors $\underline{t_i}$ together form the *matrix* $\underline{N}$ of N:

$$\underline{N} =_{def} (\underline{t_1}, \ldots, \underline{t_l}) = \begin{pmatrix} z_{1\,1} & \ldots & z_{l\,1} \\ \vdots & & \vdots \\ z_{1\,k} & \ldots & z_{l\,k} \end{pmatrix}$$

$$\underline{N} = (\underline{t}, \underline{u}) = \begin{pmatrix} -1 & 1 \\ 0 & -1 \\ 1 & 0 \end{pmatrix}$$

$$\begin{array}{c|ccc} \underline{N} & \ldots & j & \ldots \\ \hline \vdots & & & \\ i & & -1 & \\ \vdots & & & \\ i' & & 1 & \\ \vdots & & & \end{array}$$

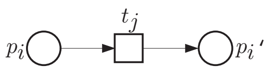


Figure 11.1 shows a technical example of an elementary system net N and its respective matrix $\underline{N}$.

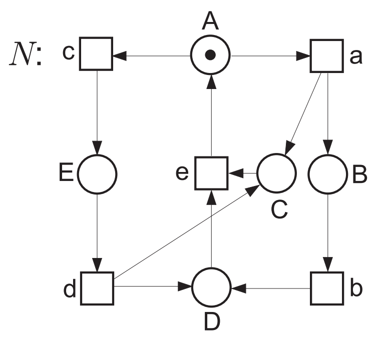
With this construction, for each step $M \xrightarrow{t_j} M'$ and each place p_i of N:

$$M'(p_i) = M(p_i) + \underline{N}(j, i).$$

11.3 Place Invariants

Section 9.5 has shown that a valid equation of an elementary system net N often takes the form

$$n_1 \cdot p_1 + \ldots + n_k \cdot p_k = n_0 \tag{2}$$



$\underline{N}$	a	b	c	d	e
A	-1		-1		1
B	1	-1			
C	1			1	-1
D		1		1	-1
E			1	-1	

Figure 11.1: An elementary system net N with its respective matrix $\underline{N}$ (entries of 0 are omitted)

with integers $n_0, \ldots, n_k$ and places $p_0, \ldots, p_k$. The number 0 is a possible factor, therefore we can always assume *all* places in their order $p_1, \ldots, p_k$ in (2).

Valid equations of the form (2) can be derived from solutions of the system of equations

$$\underline{x} \cdot \underline{N} = \vec{0}, \tag{3}$$

place invariant

where $\vec{0} = (0, \ldots, 0)$ is an l-dimensional row vector. A vector $\underline{n} = (n_1, \ldots, n_k)$ solves (3) if and only if $\underline{n} \cdot \underline{t} = 0$ for each transition t of N. A solution $\underline{n}$ of (3) is called a *place invariant of* N. With the initial marking M_0 of N, the number

equation of a place invariant

$$n_0 =_{\text{def}} \underline{n} \cdot \underline{M}_0$$

(i.e. $n_1 \cdot M_0(p_1) + \ldots + n_k \cdot M_0(p_k)$) is the *constant of* n. Equation (2) then is the *equation of* $\underline{n}$. The following theorem motivates this definition:

Theorem 10 (Elementary Place Invariants Theorem)**.** *Let N be an elementary system net with a place invariant $\underline{n}$. Then the equation of $\underline{n}$ holds in N.*

$\underline{N}$	t	u	M_0	$\underline{n}$
A	-1	1	1	1
B		-1	1	1
C	1			1

$\underline{n} \cdot M_0 = 2$

$\mathsf{A} + \mathsf{B} + \mathsf{C} = 2$

The equation derived from a place invariant is often called an invariant itself.

To prove this theorem, we have to show that $\underline{n} \cdot \underline{M} = n_0$ for each reachable marking M. The equation trivially holds for $M = M_0$. Furthermore, each reachable marking M can be reached from M_0 with a finite number of steps. Therefore, it suffices to show that $\underline{n} \cdot \underline{M} = \underline{n} \cdot \underline{M'}$ for each step $M \xrightarrow{t} M'$. This is now quite simple:

$$\begin{aligned} \underline{n} \cdot \underline{M'} &= \underline{n} \cdot (\underline{M} + \underline{t}) \text{ (according to (1))} \\ &= \underline{n} \cdot \underline{M} + \underline{n} \cdot \underline{t} \text{ (linear algebra)} \\ &= \underline{n} \cdot \underline{M} + 0 \text{ (because } \underline{n} \text{ is a place invariant)} \\ &= \underline{n} \cdot \underline{M} \text{ (linear algebra).} \end{aligned}$$

Figure 11.2 shows the initial marking M_0 of the system net N in Fig. 11.1 as a vector, as well as four place invariants of N together with their respective valid equations that have been derived from them.[1]

	$\underline{M_0}$	i_1	i_2	i_3	i_4
A	1	1	1	2	
B			1	1	1
C		1		1	-1
D			1	1	1
E		1	1	2	

from i_1: $\mathsf{A} + \mathsf{C} + \mathsf{E} = 1$
from i_2: $\mathsf{A} + \mathsf{B} + \mathsf{D} + \mathsf{E} = 1$
from i_3: $2\mathsf{A} + \mathsf{B} + \mathsf{C} + \mathsf{D} + 2\mathsf{E} = 2$
from i_4: $\mathsf{B} - \mathsf{C} + \mathsf{D} = 0$

Figure 11.2: Initial marking M_0, place invariants $i_1, \ldots, i_4$ and their respective equations for N in Fig. 11.1

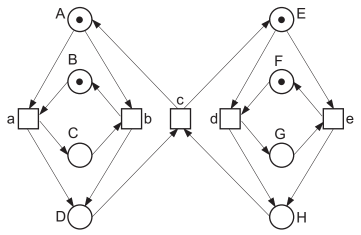
The converse of Theorem 10 holds under the mild assumption that each transition is enabled by at least one reachable marking:

Theorem 11 (Converted Place Invariants Theorem)**.** *Let N be an elementary system net such that for each transition t there exists a reachable marking M that enables t. Furthermore, let Eq. (2) hold in N. Then $\underline{n} = (n_1, \ldots, n_k)$ is a place invariant of N with the constant n_0.*

For a proof, consider a reachable step $M \xrightarrow{t} M'$. As in the proof of Theorem 10 above, it follows that $\underline{n} \cdot \underline{t} = 0$. Such a step exists for each transition t.

[1]To improve readability, the place invariants are written as column vectors although, in fact, they are row vectors.

The same argument can be used to prove the validity of modulo-2 equations. For the net in Fig. 9.4, for instance, $(1, 1, 0, 0, 1, 1, 0, 0)$ is a place invariant (with the places ordered alphabetically $\mathsf{A}, \dots, \mathsf{H}$). This invariant has the constant 4, which is congruent to 0 (modulo 2). The equation $\mathsf{A} + \mathsf{B} + \mathsf{E} + \mathsf{F} = 0$, already discussed in Section 9.6, can be derived from this.



Place invariants offer a powerful technique for proving properties of elementary system nets. For instance, all equations mentioned in Section 9.5 can be directly derived from place invariants.

11.4 Positive Place Invariants

The term "invariant" particularly applies to place invariants with nonnegative entries.

positive place invariant

A place invariant i of an elementary system net N is called *positive for a place p of N* if

$$i(p) > 0 \text{ and}$$
$$i(q) \geq 0 \text{ for each place } q \text{ of } N.$$

From this it follows that:

Theorem 12 (Positive Place Invariants Theorem)**.** *A place p with a positive place invariant is bounded.*

p thus has a number n such that $M(p) \leq n$ for all reachable markings M of N. The converse is not generally true.

Particularly simple and also quite common are positive place invariants with equations of the form

$$p_1 + \ldots + p_k = 1.$$

An example is the equation $\mathsf{A} + \mathsf{C} + \mathsf{E} = 1$ for Fig. 11.1. These places are connected via arcs and transitions (in this case a, c, d and e) and thus form "paths through the net that a token can traverse." Such a path contains

- each involved place with *all* its incoming and *all* its outgoing arcs,

- each involved transition with *exactly one* incoming and *exactly one* outgoing arc.

Figure 11.3 indicates such a path with boldfaced places, transitions and arcs.

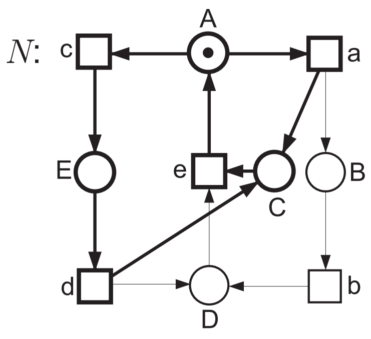


Figure 11.3: Paths of the place invariant of the equation A + C + E = 1

The *carrier* of a positive place invariant i is the set of all places p with $i(p) > 0$. Then:

Theorem 13 (Invariant Trap/Invariant Cotrap Theorem)**.** *The carrier of a positive place invariant of a system net N is also a trap and a cotrap of N.*

It is also possible in Eq. (2) to replace the integers $n_0, \ldots, n_k$ with rational numbers. Solving these equations would then be even simpler. However, rational numbers would inhibit an intuitive understanding here.

Exercises

1. Exercises relating to Fig. 11.1:

   (a) Construct two place invariants that are linearly independent. Derive their respective equations.

   (b) Construct a place invariant whose carrier contains all places, and derive its respective equation.

2. Which of the following equations and propositional formulas hold in the system net in Fig. 11.4? Either prove them with the help of place invariants or construct a counterexample.

   (a) $\mathsf{A} + 2\mathsf{B} + \mathsf{C} + \mathsf{D} + \mathsf{E} = 2$

   (b) $\mathsf{A} + \mathsf{B} + \mathsf{C} = 1$

   (c) $\mathsf{E} \rightarrow \mathsf{A} \vee \mathsf{C} \vee \mathsf{D}$

   (d) $\mathsf{A} \rightarrow \mathsf{E}$

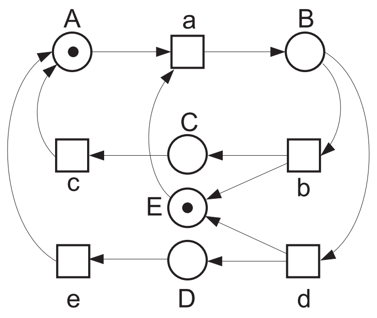


Figure 11.4: System net

3. Construct a counterexample for the converse of each of the two theorems in Section 11.4.

4. Derive all the equations and inequalities shown in Section 9.5 from place invariants and traps of their respective system nets.

5. Let N be an elementary system net in which each place has a positive place invariant. Show that there exists a single place invariant that is positive for all places of N.

Further Reading

Place invariants of elementary system nets as well as the matrix of a net and the vector representation of markings and steps were introduced by Lautenbach in [43].

With modulo-n invariants, Desel et al. [17] broke new ground. The matrix representation and the concept of place invariants seamlessly carry over to Valk's self-modifying Petri nets [75]: for a net N with an arc (t, p), whose weight is equal to the current number of tokens in the place q, the matrix entry $\underline{N}(t, p) =_{def} q \cdot p$ is generated. Equations for the calculation of place invariants are thus no longer linear.

The *calculation* of place invariants is complex, because only integer solutions to the system of Eqs. (3) are valid; [22] explains the details. In [80], Yamauchi and his coauthors exploit Theorem 13 to derive an efficient heuristic that discards candidates that are neither traps nor cotraps.

Combining Traps and Place Invariants of Elementary System Nets

Chapter 12

When two valid equations or inequalities are added or subtracted, the result is again a valid equation or inequality. We will show that such calculations substantially increase the expressive power of traps and place invariants. We will start with equations and inequalities of elementary system nets.

12.1 Calculating with Equations and Inequalities

According to Eq. (2) in Section 11.3, an equation of an elementary system net takes the form

$$G_1 : \quad n_1 \cdot p_1 + \ldots + n_k \cdot p_k = n_0.$$

The *addition* of an equation

$$G_2 : \quad m_1 \cdot p_1 + \ldots + m_k \cdot p_k = m_0$$

to G_1 yields the equation

$$G_1 + G_2 : \quad (n_1 + m_1) \cdot p_1 + \ldots + (n_k + m_k) \cdot p_k = n_0 + m_0.$$

sum of equations

The *scalar product* of G_1 and a factor z yields

$$z \cdot G_1 : \quad z \cdot n_1 \cdot p_1 + \ldots + z \cdot n_k \cdot p_k = z \cdot n_0,$$

scalar product of an equation and a number

where z may very well be a negative number. Likewise, calculations with inequalities are done according to the usual rules. These operations retain the validity of equations and inequalities, and we can exploit this fact for a very powerful proving technique:

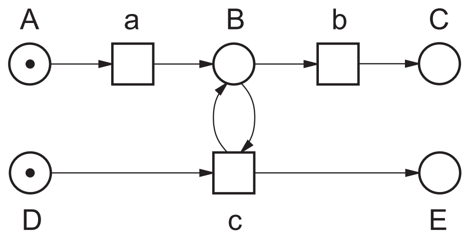


Figure 12.1: Technical example: $\mathsf{A} + \mathsf{E} \leq 1$

Theorem 14 (Addition Theorem of Valid Equations and Inequalities)**.** *Let N be an elementary system net. The sum of two equations or inequalities that hold in N, as well as the product of such an equation or inequality with a factor z, again hold in N.*

As a proof, consider a marking M that is reachable in N. Thus, G_1 and G_2 hold in M. Because $n_i \cdot M(p_i) + m_i \cdot M(p_i) = (n_i + m_i) \cdot M(p_i)$ for $(i = 1, \ldots, k)$, the sum $G_1 + G_2$ also holds in M. For z the argument is analogous.

These apparently obvious calculating steps massively increase the expressiveness of the information that can be derived from traps, place invariants and the canonical inequalities $p \geq [\,]$ (cf. Section 9.4).

Figure 12.1 shows a technical example system N with the valid inequality

$$\mathsf{A} + \mathsf{E} \leq 1. \tag{1}$$

To prove (1) we need the following two equations derived from place invariants:

$$\mathsf{A} + \mathsf{B} + \mathsf{C} = 1, \tag{2}$$

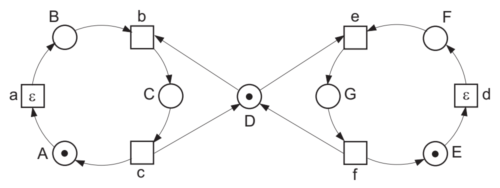
$$\mathsf{D} + \mathsf{E} = 1 \tag{3}$$

as well as the inequality

$$\mathsf{B} + \mathsf{C} + \mathsf{D} \geq 1 \tag{4}$$

derived from the initially marked trap $\{\mathsf{B,C,D}\}$. With this, (1) is the result of (2) + (3) − (4).

12.2 State Properties of the Mutual Exclusion System



The central property of the mutual exclusion system in Fig. 3.2 is

$$\mathsf{C} + \mathsf{G} \leq 1. \qquad (5)$$

This follows from the place invariant

$$\mathsf{C} + \mathsf{D} + \mathsf{G} = 1$$

by subtracting the canonical inequality

$$\mathsf{D} \geq 0$$

of D (i.e., the addition of $-\mathsf{D} \leq 0$).

12.3 State Properties of the Crosstalk Algorithm

Figure 12.2 expands the crosstalk algorithm in Fig.5.2 by the six places Q, R, S and U, V, W, where Q, R and S “count” the cycles of the left agent l, ordered by three properties:

Q-cycle:	l sends a message and receives a confirmation.
R-cycle:	l sends a message and receives a message (crosstalk).
S-cycle:	l receives a message and sends a confirmation.

Likewise, the places U, V and W “count” the cycles of the right agent r, ordered by the respective properties of r.

Now we want to show that the cycles “match each other”: if both agents are in their idle states, each two corresponding places hold equally many tokens. Technically, this means for each reachable marking M with $M(\mathsf{A}) = M(\mathsf{D}) = 1$:

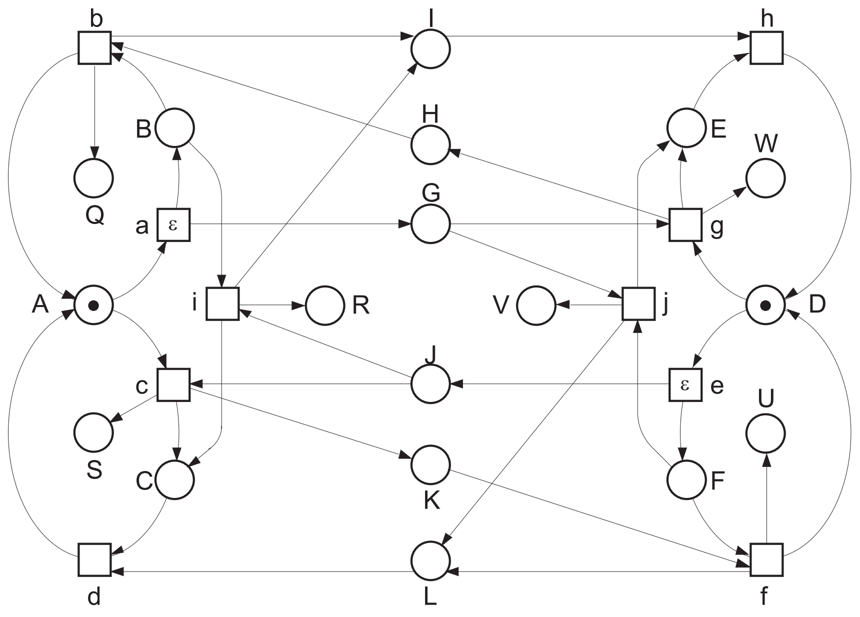


Figure 12.2: The crosstalk algorithm, expanded by six counters Q, R, S, U, V, W

$$M(\mathsf{Q}) = M(\mathsf{W}) \tag{6}$$
$$M(\mathsf{R}) = M(\mathsf{V}) \tag{7}$$
$$M(\mathsf{S}) = M(\mathsf{U}) \tag{8}$$

To prove (6) we first derive a series of valid inequalities:

(i)	$\mathsf{Q}+\mathsf{H}-\mathsf{W}=0$	place invariant
(ii)	$\mathsf{H}\geq 0$	canonical inequality
(iii)	$\mathsf{Q}-\mathsf{W}\leq 0$	(i) − (ii)
(iv)	$\mathsf{Q}\leq\mathsf{W}$	(iii), transposition
(v)	$\mathsf{W}-\mathsf{H}-\mathsf{Q}=0$	place invariant
(vi)	$\mathsf{D}+\mathsf{H}+\mathsf{I}+\mathsf{J}+\mathsf{K}=1$	place invariant
(vii)	$\mathsf{I}\geq 0$	
(viii)	$\mathsf{J}\geq 0$	
(ix)	$\mathsf{K}\geq 0$	
(x)	$\mathsf{W}-\mathsf{Q}+\mathsf{D}\leq 1$	(v)+(vi)-(vii)-(viii)-(ix)

For each reachable marking M with $M(\mathsf{D})=1$:

(xi) $M(\mathsf{Q}) \leq M(\mathsf{W})$ according to (iv)
(xii) $M(\mathsf{W}) \leq M(\mathsf{Q})$ according to (x)

(6) now follows from (xi) and (xii). Propositions (7) and (8) are derived analogously.

12.4 Unstable Properties

A majority of the important state properties of an elementary system net N can usually be formulated as valid equations or inequalities (cf. Section 9.2). In Sections 9.4, 10.1 and 11.3 we introduced three techniques for deriving valid equations and inequalities:

- For each place p of N, the canonical inequality

  $$p \geq 0$$

  of p holds, based on the rules for the occurrence of transitions.

- Each initially marked trap generates a valid inequality of the form

  $$p_1 + \ldots + p_k \geq 1,$$

  where $p_1, \ldots, p_k$ are the places of N.

- Each place invariant generates a valid equation of the form

  $$n_1 \cdot p_1 + \ldots + n_k \cdot p_k = n_0,$$

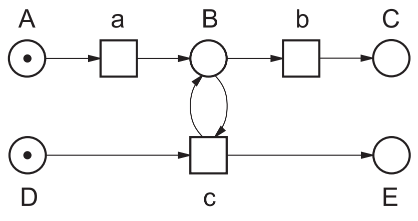
  where $n_0, \ldots, n_k$ are integers, and $p_1, \ldots, p_k$ are the places of N.

An equation or inequality α is *stable* in an elementary system net if for each step $M \xrightarrow{t} M'$ of N the following holds:

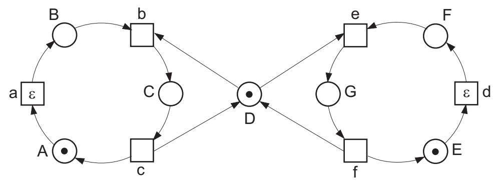
stable state property

If α holds in M, then α also holds in M'.

It is irrelevant here whether or not M is reachable in N. It is easily asserted that each canonical inequality as well as each



unstable property: A+E≤ 1



equation or inequality derived from a trap or place invariant is stable.

However, important valid equations or inequalities are often unstable. A typical example is the inequality (1) for the system net in Fig. 12.1. This inequality holds in the (unreachable) marking ABD. The step ABD $\xrightarrow{\mathsf{c}}$ ABE is a step of N, where the reached marking ABE violates the inequality (1).

Similarly, the inequality (5) for the mutual exclusion system holds in the (unreachable) – marking BDG. After the step BDG $\xrightarrow{\mathsf{b}}$ CG the reached marking CG violates (5).

Nevertheless, it was possible to prove the inequalities (1) and (5) for the system nets in Figs.12.1 and 12.2: the addition of valid, stable equations and inequalities yields valid equations and inequalities that may be unstable.

In practice, most important state properties can be proven by adding canonical inequalities as well as equations and inequalities derived from traps and place invariants. Technical examples like modulo equations (cf. Section 9.6) or properties like “The number of tokens in $p_1, \ldots, p_n$ is a prime number.” rarely occur in practical applications.

Exercises

1. Find a proof for each of the inequalities given in (a), (b) and (c), Figs. 12.3–12.5. Do this by adding equations and inequalities derived from place invariants, initially marked traps and canonical inequalities of the respective system nets.

(a)

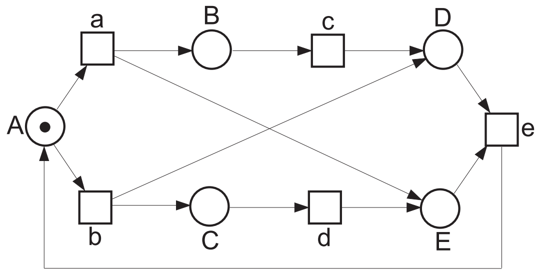


Figure 12.3: $\mathsf{B}+\mathsf{C} \leq 1$ is valid

(b)

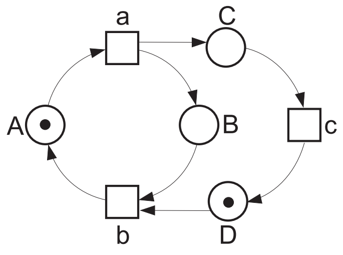


Figure 12.4: $\mathsf{C}+\mathsf{D} \geq 1$ is valid

(c)

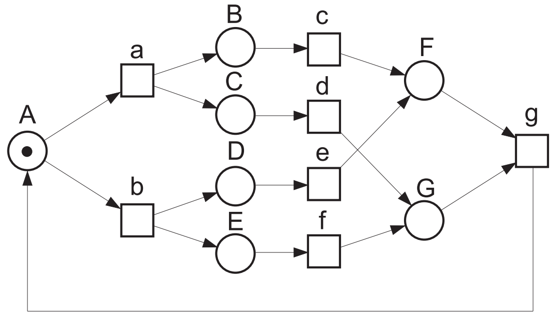


Figure 12.5: $\mathsf{C}+\mathsf{D} \leq 1$ is valid

2. In the light/fan system in Fig. 7.2, the cold transitions switch light on and switch light off are the user's *actions*. The transitions fan starts and fan stops are the system's *reactions*. Show that there cannot occur more reactions than actions.
Hint: Expand Fig. 7.2 by a place E, as shown in Fig. 12.6. Evidently, E is marked if and only if there have occurred more actions than reactions. Now show that whenever a reaction is possible in the original system in Fig. 7.2, the place E in the expanded system is marked. (Thus, the corresponding reaction is also possible in the expanded system.)

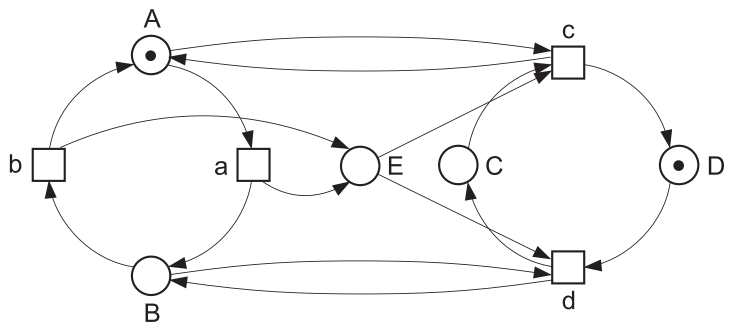


Figure 12.6: Light/fan system from Fig. 7.2 with changed denotations and expanded by place E

3. Show that the propositions (7) and (8) hold in the system net in Fig. 12.2.

* 4. Prove the following: each equation derived from a place invariant, each inequality derived from an initially marked trap and each canonical inequality is stable.

Traps and Place Invariants of Generic System Nets

Chapter 13

Analogously to Chapters 10 and 11, traps and place invariants can also be formulated for generic system nets. Traps are less important here, and we will show only one example. The place-invariant calculus uses expressions as they occur in arc labelings.

13.1 Traps of a System Net

Traps of elementary and generic system nets are structurally identical. Condition (1) in Chapter 10 suffices. Of interest is the following inequality derived from an initially marked trap $Q = \{q_1, \ldots, q_r\}$:

If there exists a place $p \in Q$ such that $p \in {}^\bullet t$, then there also exists a place $q \in Q$ such that $q \in t^\bullet$.

$$|q_1| + \ldots + |q_r| \geq 1. \tag{1}$$

inequality of a trap

Its validity is obvious. One can often be even more precise, as Fig. 13.1 shows. If f(f(u$_i$)) = u$_i$ for $i = 1, \ldots, n$, then the inequality $\mathsf{A}+\mathsf{C}+\mathsf{f}(\mathsf{D}) \geq [\mathsf{u}_1, \ldots, \mathsf{u}_n]$ holds for the trap $\{\mathsf{A,C,D}\}$.

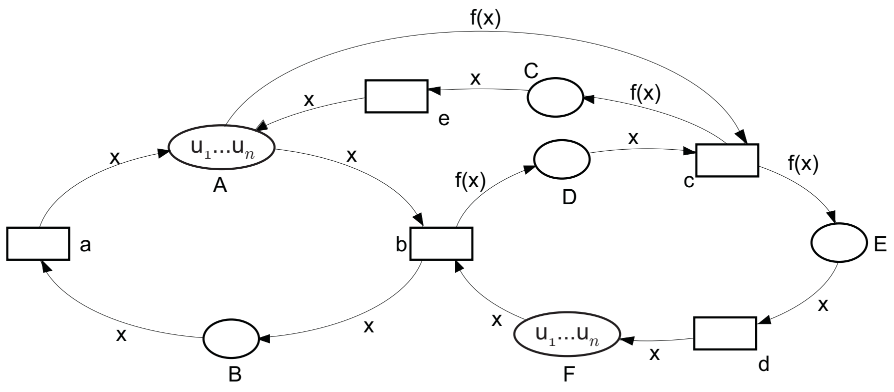


Figure 13.1: Trap $\{\mathsf{A, C, D}\}$ with the inequality $\mathsf{A} + \mathsf{C} + \mathsf{f}(\mathsf{D}) \geq [\mathsf{u}_1, \ldots, \mathsf{u}_n]$, holding if f(f(u$_i$))=u$_i$

13.2 Sum Expressions

As described in Chapter 2, the arcs of a system net N are labeled with expressions. Typical expressions of this type are constants like 1€, ◻, ⬭, ●, variables like x, y and expressions like x-1 or f(x). However, they can also be more complex, as for instance, (x+1)mod 60 in Fig. 4.9. In general, an arc can be labeled with several expressions, for instance, with l(x) and r(x) as in Fig. 9.1. Section 2.6 explains that such a set of expressions (with values assigned to the variables) describes a multiset. We now use expressions to formulate the sum of such multisets: the *sum* $b_1 + b_2$ of two expressions b_1 and b_2 as well as the *inverse* $-b$ of an expression b are again expressions. As with numbers, we write $b_1 - b_2$ for $b_1 + (-b_2)$ and assume a special expression “0” for the empty multiset.

sum expression

By applying the following arithmetic rules, we can *calculate* with such expressions b_1, b_2, b_3 just as with numbers:

$$\begin{aligned}
&b_1 + b_2 = b_2 + b_1,\\
&(b_1 + b_2) + b_3 = b_1 + (b_2 + b_3),\\
&b_1 + 0 = b_1,\\
&b_1 - b_1 = 0.
\end{aligned}$$

Such expressions together with their arithmetic rules are called *sum expressions*. With these arithmetic rules, a sum expression describes a multiset:

$$b_1 + \ldots + b_n \text{ describes } [b_1, \ldots, b_n] \text{ and } 0 \text{ describes } [\,]. \qquad (2)$$

13.3 Multiplying Sum Expressions

product of sum expressions

For technical reasons, we need to be able to *multiply* sum expressions. Let b_1 and b_2 be sum expressions, where b_1 contains at most one variable. However, this one variable may occur several times in b_1, as for instance, the variable z in $g(z, h(z))$. We define the *product* $b_1 \cdot b_2$ to be again a sum expression. It is generated by replacing each occurrence of the variable in b_1 with b_2. As an example, consider the product $b_1 \cdot b_2$ of $b_1 = g(z, h(z))$ and $b_2 = f(x, y)$:

$$g(z, h(z)) \cdot f(x, y) = g(f(x, y), h(f(x, y))).$$

Likewise, the product $b_1 \cdot b_2$ of $b_1 = f(x)$ and $b_2 = (y + z)$ is generated as follows:

$$f(x) \cdot (y + z) = f(y + z).$$

In particular, $b_1 \cdot b_2 = b_1$ if b_1 does not contain any variables at all; and $b_1 \cdot b_2 = b_2$ if $b_1 = x$.

The product $b_1 \cdot b_2$ evidently contains only the variables of b_2. In particular, "real" sum expressions may also occur in such products as the arguments for functions, as in $f(x + y)$. With additional arithmetic rules, such expressions can always be transposed into "normal" sum expressions. For each unary function f:

calculation rules

$$f(b_1 + b_2) = f(b_1) + f(b_2),$$
$$f(-b) = -f(b),$$
$$f(0) = 0.$$

13.4 Applying a Sum Expression to a Multiset

As described in Section 2.6, each expression b with at most *one* variable x for an element u of a universe U can be evaluated by replacing each occurrence of x with u. The result is again an element of U, written as $b(u)$. Generalized to include multisets, let

$$b([u_1, \ldots, u_n]) = [b(u_1), \ldots b(u_n)]. \qquad (3)$$

If $u_1, \ldots, u_n$ are considered expressions (without variables), then (3) together with (2) corresponds to the equation

$$b \cdot (u_1 + \ldots + u_n) = b \cdot u_1 + \ldots + b \cdot u_n.$$

In (3) the sum expression b represents a function that is *applied to* a multiset $[u_1, \ldots, u_n]$ and again yields a multiset.

13.5 The Matrix $\underline{N}$ of a System Net N

matrix of a system net

In the following sections, let N be a system net with the places $p_1, \dots, p_k$ and the transitions $t_1, \dots, t_l$. The labeling $\overline{p_i t_j}$ of the arc from p_i to t_j generally consists of *several* expressions $a_1, \dots, a_n$. They are henceforth written as a sum expression $a_1 + \dots + a_n$. Likewise, the labeling $\overline{t_j p_i}$ of the arc from t_j to p_i forms a sum expression. With this, we construct the entry $\underline{N}(i, j)$ of the *matrix* $\underline{N}$ of N as the sum expression

$$N(i,j) =_{\text{def}} \overline{t_j p_i} - \overline{p_i t_j}.$$

Figure 13.2 shows six technical examples of generic system nets N_i and their respective matrices $\underline{N}_i$. To improve readability, occurrences of the term “0” are omitted.

13.6 The Place Invariants of a System Net

Now let $b = (b_1, \dots, b_k)$ be a vector of sum expressions b_i $(i = 1, \dots, k)$, each containing at most one variable. b *solves* the system of equations

$$x \cdot \underline{N} = 0 \tag{4}$$

if and only if for each j:

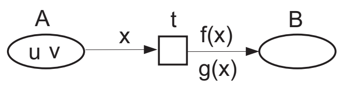
$$b_1 \cdot \underline{N}(1,j) + \dots + b_k \cdot \underline{N}(k,j) = 0.$$

place invariant of a system net

Each solution to (4) is a *place invariant* of N. Figure 13.2 shows place invariants of the system nets $N_1, \dots, N_6$. The invariant of N_5 presupposes the inverse functions f^{-1} and g^{-1}. The invariant of N_6 only holds if $\mathsf{f(f(x)) = x}$.

The variable of each sum expression b_i of a place invariant $(b_1, \dots, b_n)$ of a system net N is arbitrary. However, it is often convenient to use the ith place of N as the variable for b_i. In Fig. 13.2 we follow this convention.

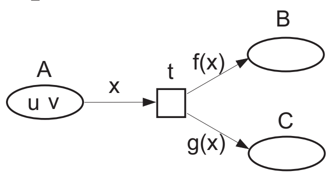
N_1



N_1	t	i	M_0
A	-x	$f(A) + g(A)$	$u + v$
B	$f(x) + g(x)$	B	

equation of i:
$f(A) + g(A) + B =$
$[f(u), g(u), f(v), g(v)]$

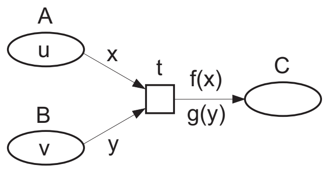
N_2



N_2	t	i	M_0
A	-x	$f(A) + g(A)$	$u + v$
B	f(x)	B	
C	g(x)	C	

equation of i:
$f(A) + g(A) + B + C =$
$[f(u), g(u), f(v), g(v)]$

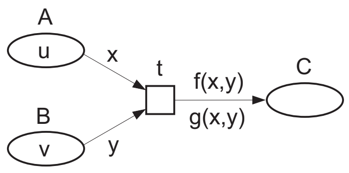
N_3



N_3	t	i	M_0
A	-x	f(A)	u
B	-y	g(B)	v
C	$f(x) + g(y)$	C	

equation of i:
$f(A) + g(B) + C =$
$[f(u), g(u)]$

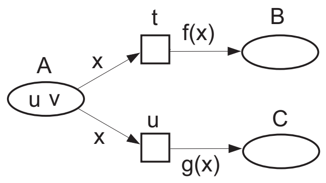
N_4



N_4	t	i	M_0
A	-x	$\|f(A)\|$	u
B	-y	$\|g(B)\|$	v
C	$f(x,y) + g(x,y)$	$\|C\|$	

equation of i:
$|A| + |B| + |C| = 2$

N_5

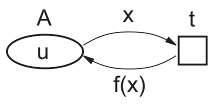


with $f^{-1}(f(x)) = x$
and $g^{-1}(g(x)) = x$

N_5	t	u	i	M_0
A	-x	-x	A	$u + v$
B	f(x)		$f^{-1}(B)$	
C		g(x)	$g^{-1}(C)$	

equation of i:
$A + f^{-1}(B) + g^{-1}(C) =$
$[u, v]$

N_6



with $f(f(x)) = x$

N_6	t	i	M_0
A	f(x)-x	$A + f(A)$	u

equation of i:
$A + f(A) = [u, f(u)]$

Figure 13.2: System nets with matrices, invariants and equations derived from them

13.7 The Constant of a Place Invariant

The initial marking M_0 of N determines the value of the constant of a place invariant $b = (b_1, \ldots, b_k)$. Thereby, the expressions b_i are applied to the initial marking of the places p_i (cf. Section 13.4). The resulting multisets are then added. The *constant of* b is thus the multiset

constant of a place invariant

$$b_1(M_0(p_1)) + \ldots + b_k(M_0(p_k)).$$

Example: The constant of the place invariant i of N_1 in Fig. 13.2 is the multiset $[\mathsf{f}(\mathsf{u}),\mathsf{g}(\mathsf{u}),\mathsf{f}(\mathsf{v}),\mathsf{g}(\mathsf{v})]$.

13.8 The Equation of a Place Invariant

We have now gathered everything we need to formulate valid equations: for a place invariant $b = (b_1, \ldots, b_k)$ of N with the constant b_0,

equation of a place invariant

$$b_1(p_1) + \ldots + b_k(p_k) = b_0$$

is the *equation of* b. Figure 13.2 shows the equations of the place invariants of the system nets $N_1, \ldots, N_6$. Section 9.2 describes the validity of such equations in N. With this, we can formulate the central theorem for place invariants of generic system nets:

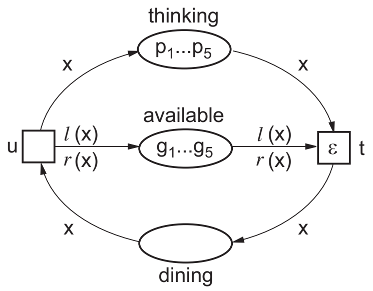
Theorem 15 (Generic Place Invariant Theorem)**.** *Let N be a system net and let b be a place invariant of N. Then the equation of b holds in N.*

The (quite extensive) proof is left to the reader. With this theorem, the derived equations shown in Fig. 13.2 thus hold in the system nets $N_1, \ldots, N_6$.

The question remains of how to calculate place invariants. This is already quite challenging for elementary system nets, because integer solutions are required there. Otherwise, the invariants would not have any intuitive meaning. At least it is possible to divide and apply the usual Gauss elimination there. However, generic system nets do not know division. In practice, system properties and place invariants are not aimlessly calculated, but are specifically required or assumed. Such an assumption is easily tested by inserting the respective values. Occasionally, an assumed place invariant is "almost" correct and can be adjusted by systematic trial and error.

13.9 Properties of the Philosophers System

Figure 13.3 shows the matrix and three place invariants of the philosophers system in Fig. 9.1. The equations of these invariants are



$$\text{thinking} + \text{dining} = [\text{p}_1, \dots, \text{p}_5], \tag{5}$$

$$\text{available} + l(\text{dining}) + r(\text{dining}) = [\text{g}_1, \dots, \text{g}_5], \tag{6}$$

$$l(\text{thinking}) + r(\text{thinking}) - \text{available} = [\text{g}_1, \dots, \text{g}_5]. \tag{7}$$

According to Theorem 15, they hold in the philosophers system. Intuitively, (5) means for each reachable state of a philosopher p_i: either p_i is thinking or p_i is dining. Equation (6) asserts for each chopstick g_i: it is either available or is the left chopstick of a dining philosopher or the right chopstick of a dining philosopher. The intuitive meaning of (7) is better understood when represented as

$$\text{available} + [\text{g}_1, \dots, \text{g}_5] = l(\text{thinking}) + r(\text{thinking}), \tag{8}$$

$\underline{N}$	t	u	M_0
thinking	$-x$	x	$[p_1, \dots, p_5]$
available	$-(l(x) + r(x))$	$l(x) + r(x)$	$[g_1, \dots, g_5]$
dining	x	$-x$	

	i_1	i_2	i_3
thinking	t		$l(\text{t}) + r(\text{t})$
available		a	-a
dining	d	$l(\text{d}) + r(\text{d})$	

Figure 13.3: Matrix $\underline{N}$, initial marking M_0 and three place invariants of the philosophers system

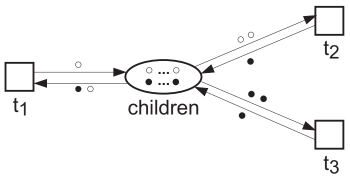
where for each chopstick g_i we distinguish two cases:

1. g_i is not available. Then g_i occurs exactly once on the left side of (8). This means that, according to the right side of (8), there exists exactly one thinking philosopher, whose left or right chopstick is g_i. Thus, the other user of the chopstick is dining.

2. g_i is available. Then g_i occurs twice on the left side of (8). This means that there exist two thinking philosophers p and q such that g_i is the left chopstick of p and the right chopstick of q.

13.10 Properties of the Kindergarten Game

To understand the model of the kindergarten game in Fig. 4.13, we derive a valid equation from one of its invariants.



	t_1	t_2	t_3	b	M_0
children	$-[\bullet]$	$[\bullet]-[\circ,\circ]$	$-[\bullet]$	$f(x)$	$B\cdot[\bullet]+W\cdot[\circ]$

with $f(\bullet)=0, f(\circ)=1$ and $1+1=0$

Figure 13.4: Matrix, place invariant and initial marking of the kindergarten game

Figure 13.4 shows its matrix, an invariant b and its initial marking M_0. Together with the modulo-2 equation $1+1=0$ in Fig. 13.4, the function f in b maps each multiset of children to either 0 or 1 (cf. Sect. 9.6). First we show that b is indeed a modulo invariant (cf. Sect. 11.3). For the product of $b=(b_1)$ and the second column $(\underline{N}(1,2))$, the following holds:

$$\begin{aligned} b_1\cdot\underline{N}(1,2) &= f(x)\cdot([\bullet]-[\circ,\circ]) \\ &= f(x)\cdot[\bullet]-f(x)\cdot[\circ,\circ] \\ &= f([\bullet])-f([\circ,\circ]) \\ &= f([\bullet])-(f([\circ]+f([\circ])) \\ &= 0-(1+1)=0-0=0 \end{aligned}$$

For the first and third column, the following holds:

$$f(x) \cdot (-[\bullet]) = f(-[\bullet]) = -f([\bullet]) = -0 = 0.$$

The initial marking M_0 consists of two numbers B and W of children dressed in black and white, respectively: $M_0(\mathsf{children}) = B \cdot [\bullet] + W \cdot [\circ]$. With this marking, the constant of the invariant b has the value

$$\begin{aligned} f(M_0(\mathsf{children})) &= f(B \cdot [\bullet] + W \cdot [\circ]) \\ &= f(B \cdot [\bullet]) + f(W \cdot [\circ]) \\ &= B \cdot f([\bullet]) + W \cdot f([\circ]) \\ &= B \cdot 0 + W \cdot 1 \\ &= 0 + W = W \\ &= \begin{cases} 0, & \text{if } W \text{ is even} \\ 1, & \text{if } W \text{ is odd.} \end{cases} \end{aligned}$$

Thus, the equation of b is

$$f(\mathsf{children}) = \begin{cases} 0, & \text{if } W \text{ is even} \\ 1, & \text{if } W \text{ is odd.} \end{cases}$$

Because the equation holds in N, in particular, it holds in the final marking M_{final} with only one token m in $\mathsf{children}$: $M_{final}\,(\mathsf{children}) = [m]$. Thus:

$$\begin{aligned} m = \bullet &\Leftrightarrow f(m) = 0 \\ &\Leftrightarrow W \text{ is even,} \end{aligned}$$

$$\begin{aligned} m = \circ &\Leftrightarrow f(m) = 1 \\ &\Leftrightarrow W \text{ is odd.} \end{aligned}$$

It follows directly: the dress color of the last remaining child depends neither on the order in which the children form pairs and leave the play area, nor on the number of children initially dressed in black. It depends solely on whether the number of children initially dressed in white is odd or even.

Exercises

1. Construct the matrix of the system net in Fig. 13.1. Find two place invariants and derive their respective equations.

2. Construct the matrix of the system net in Fig. 13.5. Find two place invariants and derive their respective equations. Let $f(f(x)) = x$ in this system net.

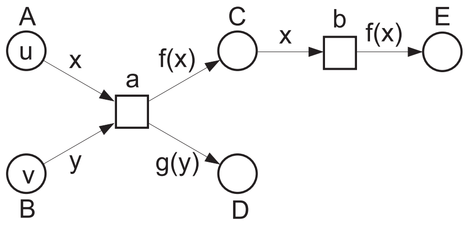


Figure 13.5: System net with $f(f(x)) = x$

* 3. Prove Theorem 15.

Further Reading

A calculus for place invariants of generic system nets was proposed in the first publication on nets with individual tokens [28], but with a commutative product. Jensen [37] has developed a semantic calculus with multisets and functions on multisets. Numerous works on the clarification of the respective interrelations followed.

The representations and calculations of place invariants of generic system nets are as diverse as the representations of the respective system nets. For instance, a colored net is, according to [37] and [38], a "folded" elementary system net. Likewise, matrices and place invariants are "folded" versions of elementary system nets. Girault and Valk [29] emphasize the functions on arcs, while [64] emphasizes the role of symbols.

Schmidt [72] shows how to calculate traps and cotraps of system nets. There exist comparatively few publications on this topic.

Typed and Untyped Formalisms

Most programming languages use explicit data types, typically *real*, *int*, *char*, *bool*, as well as types like *records* or *arrays* composed from them. Each variable of a program is assigned such a type. An attempt to assign to a variable a value that falls outside the variable's type results in an error message. This *typing* allows a compiler to generate efficient code and to recognize some inconsistencies in the program during compile time.

Similarly to the variables of a program, the places of a system net can be typed: a place p can only hold tokens of specific type (as, for instance, the place available in Fig. 9.1, which can only hold chopsticks) or p may have an upper bound for the number of tokens it can hold (cf. Chapter 6). Many variants of Petri nets type their places by extending the rule for the occurrence of transitions: the generated tokens have to fulfill the respective conditions. Jensen's colored nets [38] are a prominent example.

We use an untyped version in this book. It is technically simpler, but is as expressive as a typed calculus. However, a system net is intuitively easier to understand if each place has a specified token type. We suggest to not *demand* and construct such properties, but to *verify* them instead. Place invariants are especially suited for this. This suggestion conforms to the approach of numerous other modeling techniques in which properties (e.g., typing) are not demanded, but are verified. Lamport, for instance, uses this approach to motivate the absence of types in TLA [42].

Dijkstra’s Pebble Game

The *Pebble Game* originated in the circle around Edsger W. Dijkstra [20] (David Gries names K. Scholten as the author). It is a variant of the kindergarten game described in Section 4.6 that uses black and white pebbles in a pot as well as a game master instead of autonomous children in a play area. The system is modeled nondeterministically and is an example of a program whose complex termination behavior (the last pebble is black if and only if the initial number of white pebbles is even) is easily understood by means of program invariants. The program is very popular in introductory texts (for instance, in [31]).

This example clearly shows the general connection between programs and Petri nets: a program is executed by an operating system (here: the game master). The operating system initiates the execution of an instruction. In the case of nondeterminism, it chooses one of several alternatives. Petri nets do not assume such an executing entity. An enabled transition occurs without any external influence. Therefore, the kindergarten game in Section 4.6 does not have a game master. If the play area is “very large” or if “very many” children are playing, nobody can keep track of, much less control, the game. The concept of the distributed run is therefore very appropriate here. Further aspects of this example are discussed in [67].

Marking and Covering Graphs — Chapter 14

The marking graph G of a system net N has already been introduced in Section 2.8. Its nodes are the reachable states, its edges the reachable steps of N. We will specify a series of properties of N that are mirrored in G.

However, G is generally infinitely large. We therefore construct the finite *covering graph* H, which approximates G. A series of necessary or sufficient conditions for some properties of N can be specified by means of H.

14.1 Deriving Properties from the Marking Graph

Finite marking graphs may very well exceed exponential growth with respect to the size of N. Therefore, they are generally unsuited for manual analysis of properties of N. Nevertheless, a few properties of a system net can be identified by means of its marking graph.

We define six such properties of system nets N and immediately characterize them as properties of the marking graph G of N:

- N *terminates*, i.e., each run of N is finite: G is finite and does not contain any loops.
- N is *deadlock-free*, i.e., each reachable marking enables at least one transition: each node of G is the start of at least one edge.
- N is *live*, i.e., for each reachable marking M and each transition t there exists a marking M' that is reachable from M and enables t: for each transition t there starts a path at each node of G such that t occurs in its edge labeling.

- N is *weakly live*, i.e., for each transition t there exists a reachable marking that enables t: at least one edge of G is labeled with t.

- N is *bounded*, i.e., there exists a number b such that each place contains at most b tokens in any reachable marking: G is finite.

- N is *reversible*, i.e., from each reachable marking, the initial marking can be reached: G is strongly connected.

With this, a series of properties of system nets are reduced to graph theoretical problems. As an example, Fig. 14.1 shows an elementary system net with two unbounded places B and C. Figure 14.2 shows an initial segment of its marking graph. With the above characterizations of properties, N is deadlock-free, not live, weakly live, unbounded and not reversible.

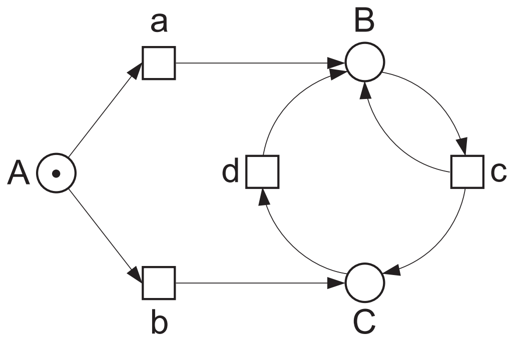


Figure 14.1: System net with unbounded places

14.2 The Idea of the Covering Graph

An elementary system net N with at least one unbounded place has an infinite marking graph. We therefore construct a finite *covering graph* H that approximates the marking graph: regular, infinite substructures of the marking graph are condensed into finite subgraphs by means of ω-*markings*. The procedure is ambiguous: N may have several distinct covering graphs. However, they all fulfill the required purpose.

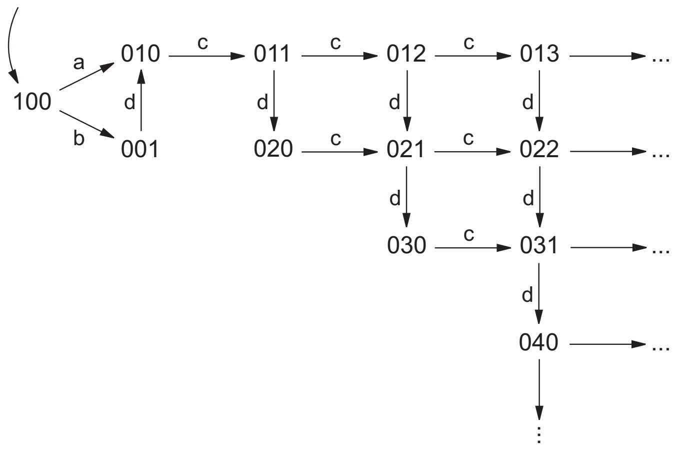


Figure 14.2: Initial segment of the marking graph of Fig. 14.1 (each marking M is written as M(A) M(B) M(C))

Technically, H uses markings M with entries of the form $M(p) = \omega$. Intuitively, such an ω-marking indicates that the place p is unbounded: for each bound n there exists a reachable marking M such that $M(p) > n$.

Four questions can be answered with the help of the covering graph:

- Is the number of reachable markings finite or infinite?
- Which places may accumulate unboundedly many tokens?
- Which sets of places may simultaneously accumulate unboundedly many tokens?
- For a given transition t, does a reachable marking M exist that enables t?

14.3 ω-Markings

ω-markings M generalize the usual markings by indicating the unboundedness of a place p with $M(p) = \omega$. An *ω-marking* M of an elementary system net N with a place set P is thus a mapping

$$M : P \to \mathbb{N} \cup \{\omega\}.$$

To handle ω-markings, we supplement the usual arithmetic rules with

$$\omega + 1 = \omega - 1 = \omega.$$

With the definitions in Section 3.1, steps $M \xrightarrow{t} M'$ in particular are well-defined for ω-markings M and M'.

We extend the order $<$ of natural numbers n by

$$n < \omega$$

and call an ω-marking M less than or equal to M', written as

$$M \leq M'$$

if $M(p) \leq M'(p)$ for each place p of N.

With the marking notation $M(\mathsf{A})M(\mathsf{B})M(\mathsf{C})$ shown in Fig. 14.2,

$$0\omega1 \xrightarrow{\mathsf{d}} 0\omega0 \xrightarrow{\mathsf{c}} 0\omega1$$

describes two (unreachable) steps of the system net in Fig. 14.1.

14.4 The Construction of the Covering Graph

covering graph

A *covering graph* H of a system net N is constructed step by step. Each step generates an edge to an existing or to a new node. Initially, H does not have any edges and only one node, the initial marking M_0 of N.

Now let the graph be partially constructed:

(a) initial segment of H for Fig. 14.1:

$$\begin{tikzcd} {} \arrow[r, bend right] & 100 \arrow[r, "a"] & 010 \arrow[r, "c"] & 01\omega \end{tikzcd}$$

(b) step $01\omega \xrightarrow{c} 01\omega$

(c)

$$\begin{tikzcd} {} \arrow[r, bend right] & 100 \arrow[r, "a"] & 010 \arrow[r, "c"] & 01\omega \arrow[loop above, "c"] \end{tikzcd}$$

a) Let the ω-marking M already be a node of H.

b) Let N have a step of the form $M \xrightarrow{t} M'$.

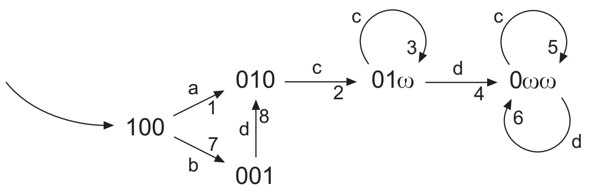
c) Let there be no t-labeled edge in H that starts in M.

Then the ω-marking M'' is defined for each place p of N as:

$$M''(p) =_{\text{def}} \begin{cases} \omega, & \text{if there exists a path from } M_0 \text{ to } M \\ & \text{that contains a node } L \text{ such that} \\ & L \leq M \text{ and } L(p) < M(p) \\ M'(p), & \text{otherwise} \end{cases}$$

The edge $M \xrightarrow{t} M''$ is now added to the graph. If M'' is not already a node, it is added as a new one. The algorithm terminates if for each node M and each transition t that is enabled in M, there exists a t-labeled edge that starts in M.

During the construction of H there generally exist several nodes with several enabled transitions that have not been processed yet. Depending on the order in which these transitions are processed, different covering graphs may be generated. Figure 14.3 shows two distinct covering graphs of the system net shown in Fig. 14.1.



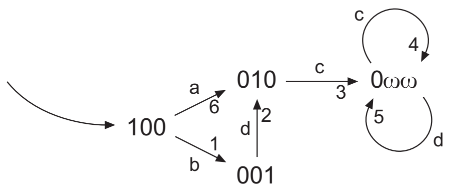


Figure 14.3: Two covering graphs of the elementary system net in Fig. 14.1. As in Fig. 14.2, markings M are written as $M(\mathsf{A})M(\mathsf{B})M(\mathsf{C})$. The indices on the edges indicate the order of their construction

If a system net is bounded and has a finite number of reachable markings, its covering graph does not contain any “real” ω-markings. The result of the construction is its marking graph as introduced in Section 2.8.

14.5 The Finiteness of the Covering Graph

A covering graph is only useful if its construction terminates. This is always the case:

Theorem 16 (Theorem on the Finiteness of Covering Graphs)**.** *A covering graph of an elementary system net N is always finite.*

The theorem's proof is essentially based on the following property: to each sequence $M_0M_1\ldots$ of mutually distinct markings M_i of N there exists an increasing sequence $n_0 < n_1 < \ldots$ of indices such that

$$M_{n_0} < M_{n_1} < \ldots, \tag{1}$$

where $M < M'$ for two markings M and M' if and only if $M(p) \leq M'(p)$ for each place p of N and $M(q) < M'(q)$ for at least one place q of N. The proof is left as an exercise. Intuitively, (1) is based on the fact that there do not exist any infinitely decreasing sequences $k_1 > k_2 > \ldots$ of natural numbers.

For an indirect proof of the theorem, we now assume that an infinite covering graph H of N would indeed exist. According to the construction of covering graphs in the previous section, each node is reachable from M_0 and only has a finite number of successors. Therefore, because H is infinite, during its construction, an infinite path

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots$$

of mutually distinct nodes (according to "König's Lemma") is generated. According to (1), the (infinite) sequence $M_0M_1\ldots$ contains an infinitely increasing subsequence $M_{n0} < M_{n1} < \ldots$. According to the construction of covering graphs, the number of places p with $M_{ni}(p) = \omega$ increases with each n_i. This would require infinitely many places. However, N only has a finite number of places. The proof follows by contradiction.

14.6 The Covering of Sequential Runs

Let N be an elementary system net. An ω-marking $\overline{M}$ of N *covers* a marking M if for each place p of N:

$$\overline{M}(p) = M(p) \text{ or } \overline{M}(p) = \omega.$$

This leads to:

Theorem 17 (Covering Theorem)**.** *Let H be a covering graph of an elementary system net N. For each sequential run $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots$ of N there exists in H a path $M_0 \xrightarrow{t_1} \overline{M}_1 \xrightarrow{t_2} \overline{M}_2 \cdots$ in which each $\overline{M}_i$ covers the marking M_i ($i = 1, 2, \ldots$).*

counterexample:

$100 \xrightarrow{a} 010 \xrightarrow{c} 01\omega \xrightarrow{d} 0\omega\omega \xrightarrow{d} 0\omega\omega \cdots$

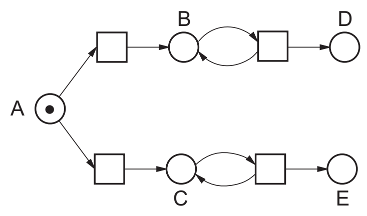
is a path in both covering graphs of Fig. 14.3, but not a step sequence in Fig. 14.1

The proof follows from the construction of H via induction over i. The converse is generally not true.

In general, different system nets may have the same covering graph. The construction of such an example is left as an exercise.

14.7 Simultaneously Unbounded Places

It is easy to construct trivial covering graphs covering each and every sequential run of an elementary system net: each place contains an ω-entry. A sensible covering graph only contains ω-entries for the unbounded places. Furthermore, several ω-entries in a single node indicate simultaneously unbounded places: a set Q of places of N is *simultaneously unbounded* if for each number $i \in \mathbb{N}$ there exists a reachable marking M^i of N such that for each $q \in Q$:



D and E are unbounded, but not simultaneously

$$M^i(q) \geq i.$$

In fact, the construction procedure in Section 14.4 only generates "sensible" covering graphs. For a node M of a covering graph and a place p of N, let

$$p \in \omega_M \text{ iff } M(p) = \omega.$$

Then:

Theorem 18 (Simultaneous Unboundedness Theorem)**.** *Let H be a covering graph of an elementary system net. Then, for each node M of H, the set ω_M is simultaneously unbounded.*

For a proof, let M be a node of H and let $i \in \mathbb{N}$. We need to show: there exists a reachable marking M^i of N such that $M^i(q) \geq i$ for each $q \in \omega_M$.

The proof is accomplished by induction over the cardinality of ω_M: if ω_M is empty, the proposition is trivial. Otherwise, the following obviously holds for each edge $M_1 \xrightarrow{t} M_2$ of H: $\omega_{M_1} \subseteq \omega_{M_2}$. Furthermore, because the initial marking of N does not mark any place with ω, there exists a path in H of the form $M' \xrightarrow{t_1} \ldots \xrightarrow{t_n} M$ with $\omega_{M'} = \omega_M$ and $\omega_{M''} \subsetneqq \omega_{M'}$ for each step $M'' \xrightarrow{t} M'$.

Then there exists a path in H of the form $L \xrightarrow{u_1} \ldots \xrightarrow{u_m} M'$ with $L \leq M'$ and $\omega_L \subsetneqq \omega_{M'}$. For the construction of M^i, we assume according to the induction hypothesis a reachable marking L^i with $L^i(p) = i + (i+n) \cdot m + n$ if $p \in \omega_L$, or $L^i(p) = L(p)$ otherwise. Now, starting from L^i let the transition sequence $u_1, \ldots, u_m$ occur $i + n$ times in total, followed by $t_i, \ldots t_n$. Each step u_i reduces the number of tokens of each place in ω_L by at most one. Each sequence $u_1, \ldots, u_m$ increases the number of tokens of each place in ω_{M^i} and ω_L by at least one. Each step t_i reduces the number of tokens of each place in $\omega_{M'}$ by at most one. Therefore, the resulting marking has the property required of M'.

14.8 Dead Transitions

A transition t is *dead* if no reachable marking enables t. The following theorem is easily proven:

Theorem 19 (Theorem on Dead Transitions)**.** *Let H be a covering graph of an elementary system net N. A transition t is dead in N if and only if H does not have a t-labeled edge.*

14.9 Covering Graphs of Generic System Nets

The procedure in Section 14.4 can also be applied to construct the covering graph of a generic system net. The order on multisets has already been introduced in Section 2.3. Everything else is analogous. In particular, the properties described in Sections 14.5 through 14.8 also apply to generic system nets.

Exercises

1. Construct two different elementary system nets that have the same covering graph.
2. In the proof of Theorem 16, the following subtask remains:
   Let P be a finite set and let $M_0 M_1 \ldots$ be a sequence of mappings $M_i : P \to \mathbb{N} (i = 0, 1, \ldots)$. Show that there exists a strictly increasing sequence $n_0 < n_1 < \ldots$ of indices such that $M_{n_0} \leq M_{n_1} \leq \ldots$.
   Hint: We recommend induction over the cardinality of P. For $|P| = 1$ and for the induction step, we recommend the inductive construction of the indices $n_0, n_1, \ldots$.
3. Construct elementary system nets N with the following properties:
   (a) N is deadlock-free and not live.
   (b) N is unbounded and reversible.
   * (c) N is live, bounded and not reversible.
   * (d) N is live (i.e., in particular deadlock-free) and reversible and for the initial marking M_0, the following holds:
   If in M_0 an additional token is placed on any of the places of N, the net with this new initial marking is no longer deadlock-free.
4. Show for any given elementary system net N:
   (a) If N is deadlock-free, it does not terminate.
   (b) If N is live, it is deadlock-free.
   (c) N is live if and only if for each reachable marking of N, when treated as the initial marking, there do not exist any dead transitions.
   (d) N is bounded if and only if the number of reachable markings of N is finite.
   (e) If N is bounded, the construction of the covering graph yields the marking graph of N.

Further Reading

As early as 1969, Karp and Miller [39] proposed the idea of the covering graph. Since then, various versions have been circulating. They differ in understandability, size, speed of termination and practicality for particular classes of nets. Finkel [25] shows how to construct minimal covering graphs.

On the basis of classic model-checking procedures, Schmidt [73] extracts many of the temporal logic formulas valid in a simple system net from its respective covering graph.

Reachability in Elementary System Nets

Chapter 15

Determining the reachability of an arbitrary marking is one of the interesting, but also one of the most difficult problems of elementary system nets. How to decide whether a marking M of an elementary system net N is reachable (from the initial marking M_0)? If only a finite number of markings is reachable, it is possible to construct each of them and test whether M is among them. If M is reachable, it is also possible to incrementally construct the (possibly infinite) marking graph until M is eventually encountered.

However, if an infinite number of markings is reachable in N, and M is not one of them, then this procedure fails. Nevertheless, the problem can be solved: it is possible to construct a finite set K of reachable markings of N such that M is reachable if and only if M is an element of K. However, this set is incredibly large and it was a long time before the reachability problem was solved.

In this chapter, we will discuss a necessary condition for the reachability and thus a sufficient condition for the unreachability of a marking. At the same time, we will formulate criteria for deciding whether a finite or an infinite number of markings are reachable.

To do this, we will add the *marking equation* and *transition invariants* to our set of linear-algebraic tools. The covering graph, too, yields criteria for the reachability of markings and for the finiteness of the set of reachable markings.

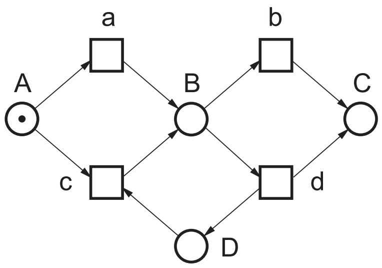
15.1 Corollaries of Place Invariants

Some information on the reachable and unreachable markings of an elementary system net N can be derived from its place invariants. A direct corollary of the Elementary Place Invariants Theorem in Section 11.3 for each marking M of N is: if N has a place invariant i such that

$$i \cdot \underline{M} \neq i \cdot \underline{M_0},$$

then M is unreachable. The converse is not generally true: $i \cdot \underline{M} = i \cdot \underline{M_0}$ does not guarantee that M is reachable.

A direct corollary of the *Positive* Place Invariants Theorem in Section 11.4:



D is not part of any place invariant. Nevertheless, only a finite number of markings is reachable.

Theorem 20 (Finiteness Theorem of Positive Place Invariants)**.** *If each place of an elementary system net N has a positive place invariant, then only a finite number of markings is reachable in N.*

The converse of this theorem is not generally true.

15.2 The Marking Equation

Using the vector representation from Section 11.1, for a sequence $M_0 \xrightarrow{t} M_1 \xrightarrow{t'} M_2$ of two steps, the following holds: $\underline{M_2} = \underline{M_0} + \underline{t} + \underline{t'}$. Consequently, for the sequence

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_1} M_3$$

the following holds:

$$\underline{M_3} = \underline{M_0} + 2\underline{t_1} + \underline{t_2}. \qquad (1)$$

This exploits the fact that M_3 does not depend on the order in which the transitions occur, only on their frequency.

For an elementary system net N with l transitions, let $v = (0,\dots,0,1,0,\dots,0)$ be an l-dimensional vector with "1" at the ith position. Then the product $\underline{N} \cdot v$ filters the ith column out of $\underline{N}$. Thus, for the step $M \xrightarrow{t_i} M'$ the following holds:

$$\underline{M'} = \underline{M} + \underline{N} \cdot v.$$

Thus, (1) can be written as

$$\underline{M_3} = \underline{M_0} + \underline{N} \cdot (2,1,0,\dots,0).$$

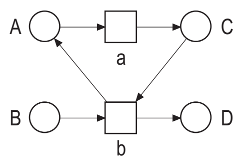
In general, for each step sequence σ from a marking M to a marking M', the following holds: if t_i occurs exactly a_i times in σ ($i = 1, \dots, l$), then the vector $\underline{a} = (a_1, \dots, a_l)$ solves the system of equations

$$\underline{M'} = \underline{M} + \underline{N} \cdot \underline{x}. \tag{2}$$

The vector $\underline{a}$ is called the *counting vector* of σ. Reasonably, we only consider vectors $\underline{a} = (a_1, \dots, a_l)$ with natural numbers a_i *solutions* of (2). The system of equations (2) is usually written as

$$\underline{N} \cdot \underline{x} = \underline{M'} - \underline{M} \tag{3}$$

and is called the *marking equation for M and M'*. A direct consequence of its construction is:



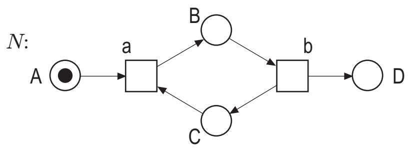
σ: AB $\xrightarrow{a}$ BC $\xrightarrow{b}$ AD $\xrightarrow{a}$ CD

counting vector: $(2,1)$

marking equation

Theorem 21 (Theorem on the Marking Equation)**.** *Let N be an elementary system net with a step sequence σ from a marking M to a marking M'. The counting vector of σ solves the marking equation for M and M'.*

The converse is not generally true: not each solution $\underline{a}$ to the marking equation (3) is a counting vector of a step sequence from M to M'. However, if it is possible to – figuratively speaking – “borrow” tokens, then each solution to the marking equation is a counting vector of a step sequence:



Let $M =_{def}$ A and $M' =_{def}$ D.
$\underline{a} = (1,1)$ solves $\underline{N} \cdot x = M' - M$.
No step sequence $M \to \dots \to M'$ has $\underline{a}$ as its counting vector.
With $\underline{L} =$ C, the following holds:
$\underline{M} + \underline{L} \xrightarrow{a}$ B $\xrightarrow{b} \underline{M'} + \underline{L}$.

Theorem 22 (Viability Theorem)**.** *Let N be an elementary system net with markings M and M' and a solution $\underline{a}$ to the marking equation (3). Then there exists a marking L of N and a step sequence σ from $\underline{M} + \underline{L}$ to $\underline{M'} + \underline{L}$ such that $\underline{a}$ is the counting vector of σ.*

For a proof, one can choose a sufficiently large L such that all transitions in σ can occur independently from one another.

The “borrowing” of tokens is only necessary if the net N contains a cycle, that is, if a chain of arcs closes a circle. Otherwise, N is *acyclic*. Each solution to (3) is then the counting vector of a step sequence from M to M':

Theorem 23 (Acyclic Viability Theorem)**.** *Let N be an acyclic elementary system net with markings M and M', and let $\underline{a}$ be a solution to the marking equation for M and M'. Then $\underline{a}$ is the counting vector of a step sequence from M to M'.*

The theorem can be proved by induction over the sum of the components of $\underline{a}$. The transitions "leading" in N are marked in M.

From the Theorem on the Marking Equation, there immediately follows: if there does not exist a solution to (3), then M' is unreachable from M.

15.3 Transition Invariants

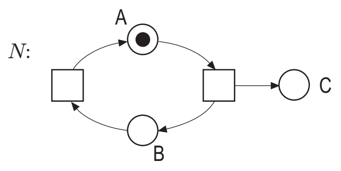
Of interest is the special case $M = M'$ in the marking equation (3). Then the equation is reduced to

$$\underline{N} \cdot \underline{x} = \overrightarrow{0}, \tag{4}$$

where the length of the vector $\overrightarrow{0} = (0, \ldots, 0)$ is equal to the number of places of N. Analogously to (3) in Section 11.3, a solution $m = (m_1, \ldots, m_k)$ to (4) with natural numbers $m_1, \ldots, m_k$ is a *transition invariant* of N. A direct corollary of (3) in Section 15.2 is:

transition invariant

Theorem 24 (Transition Invariants Theorem)**.** *Let $\underline{a}$ be a transition invariant of an elementary system net N, and let σ be a step sequence from a marking M to a marking M' such that $\underline{a}$ is the counting vector of σ. Then M and M' are identical.*



N has no transition invariant

The converse also holds and yields:

Theorem 25 (Reproducibility Theorem)**.** *If an elementary system net N does not have any transition invariants, then no marking is again reachable from itself.*

Transition invariants are of interest in various contexts. If N is bounded (if covered by a positive place invariant, for instance) and a transition t does not occur in any transition invariant with an entry > 0, then t occurs only a finite number of times.

The frequencies of the transitions of a scenario (cf. Sect. 5.1) form a transition invariant. Likewise, transition invariants with small entries are candidates for the construction of scenarios.

Exercises

1. Disprove the converse of Theorem 21: Construct an elementary system net N with an unreachable marking M', whose marking equation $\underline{N} \cdot \underline{x} = \underline{M'} - \underline{M}_0$ has a solution with natural-number entries.

2. Calculate three viable transition invariants of the crosstalk algorithm in Fig. 3.9.
Hint: Consider the connection between transition invariants and scenarios.

3. Prove the Acyclic Viability Theorem (Theorem 23).

Further Reading

The question regarding the reachability of a marking M from a marking M_0 has been around since the 1960s. Karp and Miller [39] posed it as a purely mathematical problem concerning the existence of a sequence of arbitrarily many instances from a given finite set of integral vectors, whose partial sums do not have any negative components. This problem is much more difficult than it may seem, and it was not until 1980 that Mayr solved it [47], [48]. Since then, numerous versions of its proof have been presented.

Reutenauer [71] puts the reachability problem in the context of other mathematical questions. The proof of Priese and Wimmel [61] is comparatively easy to read.

The memory requirements of each algorithm for the problem grow exponentially with the size of the net (for the specialist: it is EXPTIME hard). This makes it clear that there exists no general, practicable procedure for deciding the reachability. Therefore, other techniques for deriving necessary and sufficient criteria for the reachability are employed, based in particular on traps, cotraps, place invariants, covering graphs, marking equations and transition invariants.

Transition invariants were introduced by Lautenbach [43].

Run Properties — Chapter 16

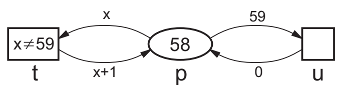
The question raised in the previous chapter, whether or not it is *possible* to reach a marking M of a system net, is now strengthened to considering whether or not M is *definitely* reached.

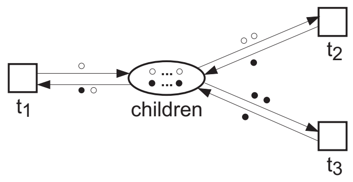
16.1 Intuitive Question

This is an entirely new problem, because each question covered thus far (in Chaps. 9–15) only relates to properties of individual, reachable markings or (like reversibility and liveness) to individual paths in the marking or covering graph. However, the question of whether a marking is definitely reached relates to each and every run of the corresponding system net.

The crosstalk algorithm N in Fig. 3.7 contains a typical example: the return of a waiting process to its idle state. In each sequential run $w = M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots$ of N, each marking M_i with $M_i(\mathsf{waiting}_l) = 1$ is followed by a marking M_{i+k}, $k \geq 0$, with $M_{i+k}(\mathsf{idle}_l) = 1$. This is not directly obvious and the question arises how to prove it.

In general, questions relating to properties with similar structures arise in many systems: the bell of the bell clock (Fig. 4.9) tolls every hour on the hour. In the kindergarten game (Fig. 4.13), only one child will eventually remain in the play area. In the system of the dining philosophers (Fig. 9.1), each chopstick will always become available again.





Some properties are composed of elementary properties. For instance, for each inserted coin, the cookie vending machine either gives out a cookie packet or returns the coin. When the light of the light/fan system (Fig. 7.2) is switched on, either the fan starts at some point or the light is switched off again.

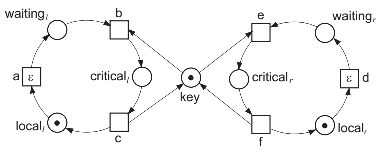
Occasionally, it is of interest that a certain property is *not* valid: the mutual exclusion system (Fig. 3.2) does not guarantee that each waiting process will eventually become critical. The crosstalk algorithm (Fig. 3.7) does not require a process to leave its idle state. All these properties are *run properties* of a system net.

16.2 Defining Run Properties

In order to comprehend run properties, we fall back on state properties (Chap. 9) of the form $p \geq 1$ for places p. As explained in Sect. 9.7, we write for a marking M

$$M \models p, \tag{1}$$

("p is valid in M") if $M(p) \geq 1$. For instance, local_l is valid in the initial marking M_0 of the mutual exclusion system in Fig. 3.2. The logical combination



$$\mathsf{local}_r \wedge \mathsf{key}.$$

is also valid in M_0.

run property

Technically, a *run property* is constructed from two state properties e and f of N and written as

$$e \mapsto f \tag{2}$$

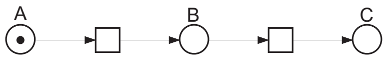
("e leads to f"). The validity of $e \mapsto f$ does not relate to an individual marking, but to a sequential run $\sigma = M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots$ of N. Property (2) *is valid in* σ if for each index i the following holds:

$$\text{if } M_i \models e, \text{ then } M_j \models f \tag{3}$$

run property valid in N

for a $j \geq i$. Property (3) is valid in a system net N if and only if (3) is valid in *each* complete sequential run of N. We write:

$$N \models e \mapsto f. \tag{4}$$



$\mathsf{A} \mapsto \mathsf{B}$ and $\mathsf{A} \mapsto \mathsf{C}$

As a technical example, $\mathsf{A} \mapsto \mathsf{E}$ is valid in the elementary system net in Fig. 16.1. This property is valid, because each reachable marking M with $M \models \mathsf{A}$ also marks C or D.

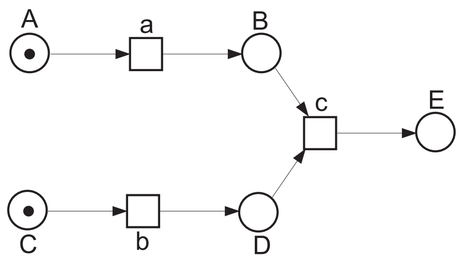
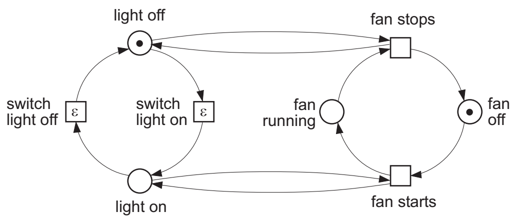


Figure 16.1: Elementary system net N with $N \models \mathsf{A} \mapsto \mathsf{E}$

A typical example is the run property

$$\mathsf{waiting}_l \mapsto \mathsf{idle}_l$$

for the crosstalk algorithm. It describes the already-mentioned return of the left process to its idle state (cf. Sect. 16.1). The run property of the light/fan system, also mentioned in Sect. 16.1, thus becomes



$$\mathsf{light\ on} \mapsto (\mathsf{fan\ running} \vee \mathsf{light\ off}).$$

One often wants to express that from *each* marking M a marking M' with a property f is reached. For instance, from each reachable marking of the crosstalk algorithm, a marking that marks idle_l is reached. To express this as a property of the form (2), a state property e is needed that is valid in each reachable marking. In the world of logic we obviously choose $e = true$. Thus, for the crosstalk algorithm, the following is valid:

$$true \mapsto \mathsf{idle}_l.$$

Thus far, we have covered run properties of elementary system nets. Run properties of generic system nets are formulated entirely analogously. The only difference is that individual tokens u in a place p are described as propositional atoms, analogously to (1). As explained in Section 9.7, we write $p.u$. With this notation, for instance, the return of the left and right chopstick of each dining philosopher in the system net in Fig. 9.1 is described as

$$\mathsf{dining}.a \mapsto (\mathsf{available}.l(a) \wedge \mathsf{available}.r(a)).$$

For each inserted coin, the cookie vending machine in Fig. 1.10 either gives out a cookie packet or returns the coin. To formulate this, we have to add a return slot to the model, as shown in Fig. 16.2. Then (using the notations from Section 9.7) the following holds:

$$\mathsf{coin\ slot}.\text{1€} \mapsto \mathsf{compartment}.▯ \vee \mathsf{return\ slot}.\text{1€}$$

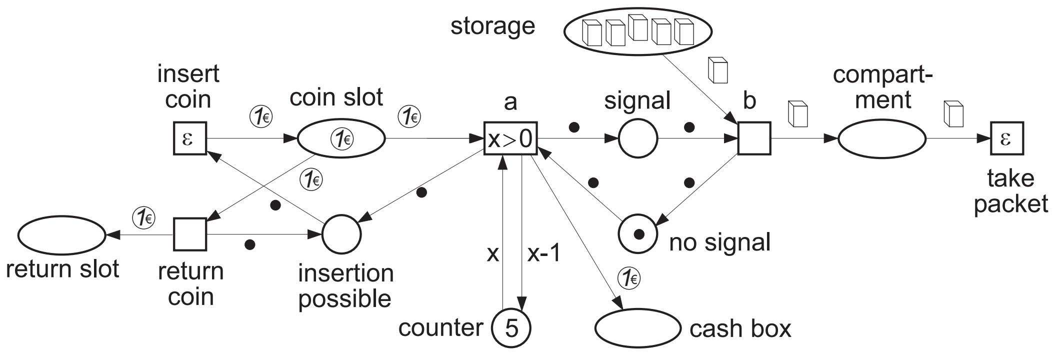


Figure 16.2: Addition of a return slot

For the bell clock in Fig. 4.10 and for the kindergarten game in Fig. 4.13, the respective properties $true \mapsto \mathsf{on}$ and $true \mapsto |\mathsf{children}| = 1$ are obviously valid.

16.3 The Deduction Rule

Section 16.2 defined the validity of a run property in a system net N over the set of all complete sequential runs of N. The net N may very well have infinitely many such runs and each may be infinitely long. To show that a run property $e \mapsto f$ is valid in a system net, it is obviously impossible to assert for each individual run that $e \mapsto f$ is valid in it. However, it is possible to deduce simple, valid run properties directly from the static structure of a system net and then use them to derive more complex properties.

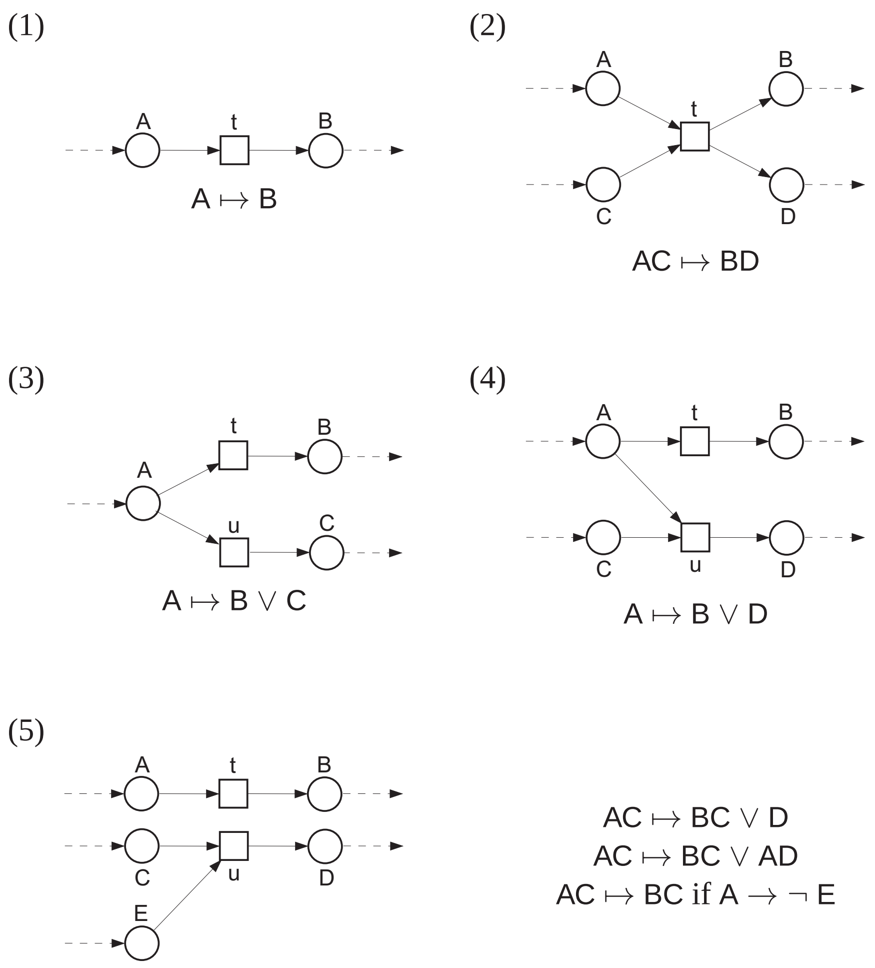


Figure 16.3: Substructures of a net N and run properties valid in N

Figure 16.3 shows five examples of small subnets. They are to be seen as being embedded into a larger net N, indicated by the dashed arcs. Each subnet's corresponding property is then valid in N (for the notation, cf. Eq. (25) in Sect. 9.7).

Consider, for instance, the case of the subnet (1) occurring in N: let w be a complete run of N, and let M be a marking of w with a token in A. Since M enables the transition t and there are no other transitions in $\mathsf{A}^\bullet$, the transition t will eventually occur in w. In doing so, a marking M' with a token in B is reached. Therefore, $\mathsf{A} \mapsto \mathsf{B}$ is valid in N.

The case of the subnet (4) occurring in N is completely analogous: let M be a marking of w with a token in A. The marking M thus enables the transition t. Now, during the course of w, the place C may also receive a token. In this case, u may occur instead of t. Therefore, we may only conclude that either B or D receives a token.

The example list of valid run properties in Fig. 16.3 is by no means complete. The general case of elementary system nets is described by a *deduction rule* for run properties. For an elementary system net N with the set P of places, this rule generates valid properties of the form

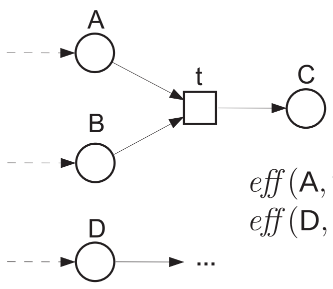
$$Q \mapsto (f_1 \vee \ldots \vee f_n),$$

where $Q, f_1 \ldots, f_n \subseteq P$. With the shorthand of (25) from Section 9.7, a set $\{p_1, \ldots, p_k\}$ of places describes the logical expression $p_1 \wedge \ldots \wedge p_k$. To formulate the deduction rule, we define the *effect* that a transition t of N has on a set $Q \subseteq P$ as the set of tokens of Q that remain after the occurrence of t or are newly generated:

$\mathit{eff}(Q,t)$

$$\mathit{eff}(Q,t) =_{\text{def}} (Q \setminus {}^\bullet t) \cup t^\bullet .$$

Figure 16.4 shows examples of the effect of a transition on different sets of places. If $Q \subseteq {}^\bullet t$ and $R \cap {}^\bullet t = \emptyset$, then obviously $\mathit{eff}(Q \cup R, t) = R \cup t^\bullet$.



$$\mathit{eff}(\mathsf{A},\mathsf{t}) = \mathit{eff}(\mathsf{AB},\mathsf{t}) = \mathsf{C}$$
$$\mathit{eff}(\mathsf{D},\mathsf{t}) = \mathit{eff}(\mathsf{AD},\mathsf{t}) = \mathit{eff}(\mathsf{ABD},\mathsf{t}) = \mathsf{CD}$$

Figure 16.4: The effect of t on different sets of places

All this leads to the following *deduction rule* for run properties of a system net N with a given set Q of places of N:

1. Test applicability: does N have a hot transition t such that ${}^\bullet t \subseteq Q$? If not, the rule is not applicable.

2. Exclude transitions: choose $T = \{t_1, \ldots, t_n\} \subseteq Q^\bullet$ such that for each $t \in Q^\bullet$ the following holds: either $t \in T$ or $N \models Q \to \neg^\bullet t$.

3. Derive property: the property $Q \mapsto \mathit{eff}(Q, t_1) \vee \ldots \vee \mathit{eff}(Q, t_n)$ is valid in N.

As an example, we prove $\mathsf{AC} \mapsto \mathsf{BC}$ for (5) in Fig. 16.3 under the assumption that $\mathsf{A} \to \neg\, \mathsf{E}$:

1. t is a hot transition with $^\bullet\mathsf{t} = \mathsf{A}$. Thus, the rule is applicable.

2. $(\mathsf{AC})^\bullet = \{\mathsf{t}, \mathsf{u}\}$. From the assumption that $\mathsf{A} \to \neg\, \mathsf{E}$, it follows that $\mathsf{A} \to \neg^\bullet\mathsf{u}$. Thus, we can exclude u and choose $T = \{\, \mathsf{t} \,\}$.

3. The proof follows, since $\mathit{eff}(\ \mathsf{AC,t}\ ) = \mathsf{BC}$.

The following theorem asserts that each property derived via this rule is actually valid in N:

Theorem 26 (Theorem on Deduced Run Properties)**.** *Let N be an elementary system net, and let Q be a subset of its places such that Q enables at least one hot transition of N. Let $T = \{t_1, \ldots, t_n\} \subseteq Q^\bullet$ such that $N \models Q \to \neg^\bullet t$ for each $t \in Q^\bullet \backslash T$. Then*

$$N \models Q \mapsto \mathit{eff}(Q, t_1) \vee \ldots \vee \mathit{eff}(Q, t_n).$$

With this theorem, the properties shown in Figs. 16.3 and 16.4 follow immediately. When testing the rule's applicability, it is important that t is hot. If, in the examples in Fig. 16.3, the transition t was cold and u was hot, only (3) would be valid. A property of the form $\mathsf{C} \mapsto q$ cannot be derived for any of the examples in Fig. 16.3, because C alone does not enable any transition.

16.4 Proof Graphs

Such simple run properties, deduced directly from the net structure, can be used in *proof graphs* to assert more complex properties. This technique exploits a series of characteristics of run properties. The most important is transitivity:

$$\begin{aligned}&\text{If } N \models e \mapsto f \text{ and } N \models f \mapsto g,\\ &\text{then also } N \models e \mapsto g.\end{aligned} \tag{5}$$

With this, it is possible to form chains

$$e_0 \mapsto e_1 \mapsto \ldots e_n \tag{6}$$

to prove that $e_0 \mapsto e_n$.

Another important characteristic is the rather simple observation that "$\mapsto$" is weaker than the logical implication:

$$\text{If } N \models e \rightarrow f, \text{ then } N \models e \mapsto f. \tag{7}$$

As to the proof: (3) requires $j \geq i$; here $j = i$ suffices.

Finally, the logical disjunction and "$\mapsto$" can be combined:

$$\begin{aligned}&\text{If } N \models e \mapsto (f_1 \vee f_2),\, N \models f_1 \mapsto g \text{ and } N \models f_2 \mapsto g,\\ &\text{then also } N \models e \mapsto g.\end{aligned} \tag{8}$$

Obviously, the propositional conjunction

$$N \models e \mapsto (f_1 \wedge f_2) \text{ implies that } N \models e \mapsto f_1 \text{ and } N \models e \mapsto f_2,$$

and for the propositional disjunction,

$$N \models e \mapsto f_1 \text{ or } N \models e \mapsto f_2 \text{ implies that } N \models e \mapsto (f_1 \vee f_2).$$

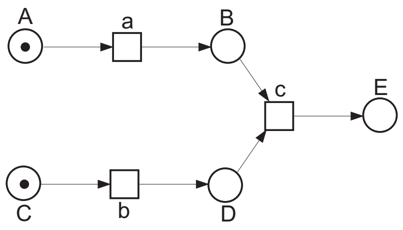
The respective converse does not hold.

We now define a *proof graph* as a finite, acyclic graph with state properties as nodes. A proof graph graph G for $N \models p \mapsto q$ has p as its initial and q as its final node. For each node r with

$$\begin{tikzcd} & \mathsf{r}_1 \\ \mathsf{r} \arrow[ur] \arrow[dr] & \vdots \\ & \mathsf{r}_n \end{tikzcd}$$

let $N \models r \mapsto (r_1 \vee \ldots \vee r_n)$.

Figure 16.5 shows a proof graph for $N \models \mathsf{A} \mapsto \mathsf{E}$ with N from Fig. 16.1. To aid an intuitive understanding, we write "$\mapsto$"-edges as "$\rightarrow$" if "$\mapsto$" represents the propositional implication according to (7). Edge labelings indicate the transitions that occur.



$$\begin{tikzcd}
\mathsf{A} \arrow[r, mapsto, "\mathsf{a}"] & \mathsf{B} \arrow[r] \arrow[rr, bend left] & \mathsf{BC} \arrow[r, mapsto, "\mathsf{b}"] & \mathsf{BD} \arrow[r, mapsto, "\mathsf{c}"] & \mathsf{E}
\end{tikzcd}$$

Figure 16.5: Proof graph for $N \models A \mapsto E$

The examples in Fig. 16.3 suffice to show the correctness of the steps of the proof graph in Fig. 16.5. The first step $\mathsf{A} \overset{\mathsf{a}}{\mapsto} \mathsf{B}$ follows from (1) in Fig. 16.3. Next is the logical implication $\mathsf{B} \rightarrow (\mathsf{BC} \vee \mathsf{BD})$, which is valid in N due to the place invariant $\mathsf{A} + \mathsf{B} - \mathsf{C} - \mathsf{D} = 0$. The step $\mathsf{BC} \mapsto \mathsf{BD}$ follows with $\mathsf{C} \rightarrow \neg\mathsf{D}$ from the place invariant $\mathsf{C} + \mathsf{D} + \mathsf{E} = 1$. Finally, $\mathsf{BD} \mapsto \mathsf{E}$ is obvious.

Exercises

1. Construct elementary system nets that disprove each of the following propositions:

   (a) If $N \models p \mapsto q$ and $N \models r \mapsto s$, then $N \models p \wedge r \mapsto q \wedge s$.

   (b) If $N \models q \wedge r \mapsto s$, then $N \models q \mapsto s$ or $N \models r \mapsto s$.

2. Prove the following for the crosstalk algorithm in Fig. 3.9: if the left process leaves its initial state, A, it returns there.

3. (a) The run property $N \models \mathsf{AB} \mapsto \mathsf{C}$ holds in the system net in Fig. 16.6. Show that this cannot be proved by means of a proof graph.

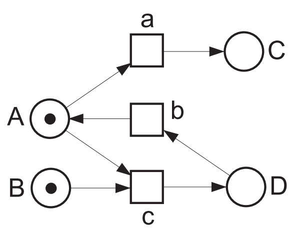


Figure 16.6: System net N with $N \models \mathsf{AB} \mapsto \mathsf{C}$

   (b) Add a place E to N in Fig. 16.6 that represents $\neg$B, that is, the complement of B. Now prove for this new system net N' that $N' \models \mathsf{AB} \mapsto \mathsf{C}$.

4. Prove or disprove:

   (a) If $N \models e_1 \mapsto f$ and $N \models e_2 \mapsto f$, then $N \models (e_1 \vee e_2) \mapsto f$.

   (b) If $N \models (e_1 \vee e_2) \mapsto f$, then $N \models e_1 \mapsto f$ and $N \models e_2 \mapsto f$.

* 5. Prove the reading rule for run properties (Theorem 26).

Further Reading

The term *run property*, as defined in Sect. 16.2, is central to specification techniques that are based on temporal logic [55], [50] and is usually termed “leads to”. The reading rule and proof graph are taken from [64]. A variant of run properties for *distributed* runs is also proposed there. Such properties are more liberal and the deduction rule is simpler than in Sect. 16.3.

Temporal Logic and *leads to*

Since the end of the 1970s, temporal logic has been used successfully in computer science to describe and prove important system properties [46]. In addition to the logical operations, temporal logic uses the *modal* operator “$\Box$” and interprets it in sequential runs σ as follows: if a property φ is valid in σ, then $\Box\varphi$ is even valid in each suffix, that is, in each residual segment of σ. As an abbreviation, $\Diamond\varphi$ stands for $\neg\Box\neg\varphi$.

With the “linear time” interpretation of temporal logic, a formula φ is valid in a system N (written: $N \models \varphi$) if and only if φ is valid in each run of N. With this, we can use temporal logic to express run properties (cf. (4) in Sect. 16.2):

$$N \models e \mapsto f \text{ iff } N \models \Box(e \mapsto \Diamond f).$$

The validity of an equation or inequality G in N (cf. Chap. 9) is expressed in the usual notation of temporal logic as $N \models \Box G$.

As early as 1982, Owicki and Lamport [55] emphasized the importance of the “$\mapsto$”-operator (“leads to”). Not unlike us, Misra and Chandy use it as the sole operator for run properties in their UNITY formalism [50].

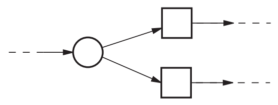
Occasionally, temporal logic is also used in connection with *distributed* runs. This leads to the formulation of properties that cannot be expressed within the ambit of sequential runs [64]. However, distributed runs mainly serve as the basis for *partial order model checking*. In this process, the validity of a property q in *a single*, cleverly chosen sequential run or in a small set of initial segments of distributed runs implies the validity of q in “large” sets of runs [49], [23].

Free-Choice Nets — Chapter 17

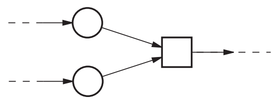
Traps and place invariants exploit the general structure of Petri nets. For nets with particular structural properties, like the *free-choice* property, there are further analysis techniques.

17.1 Defining Free-Choice Nets

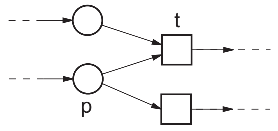
Essentially, two structural properties describe the dynamics of a system net:



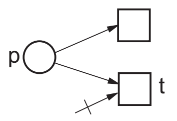
- Places with multiple succeeding transitions: such a place can describe different alternatives and continue a certain behavior.



- Transitions with multiple preceding places: such a transition can synchronize multiple behavioral strands.



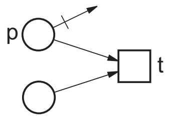
These two structural properties may be closely intertwined in a system net, for instance, if a place has multiple succeeding transitions and (at least) one of them has another preceding place. In this case, the local behavior may depend on the entire system net: a token in a place p does not necessarily enable each succeeding transition t of p. Additional tokens may independently appear or disappear in other preceding places of t. Likewise, it is not guaranteed that a synchronizing transition actually occurs.

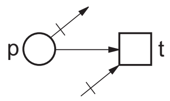


A free-choice net does not allow such structures: an elementary system net N is called a *free-choice net* if for each place p of N with $|p^\bullet| \geq 2$ and each transition $t \in p^\bullet$:

$$^\bullet t = \{p\}. \tag{1}$$

free-choice net





not both $\not\rightarrow$-arcs at the same time

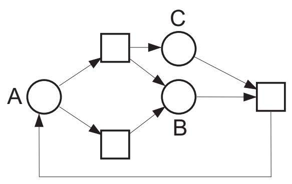
There is also a completely different motivation for this type of net: The synchronization of tokens is deterministic. The tokens required for the occurrence of a transition accumulate and do not get lost again. For each transition t of N with $|{}^\bullet t| \geq 2$ and each place $p \in {}^\bullet t$, we now require that

$$p^\bullet = \{t\}. \tag{2}$$

These two definitions, (1) and (2), actually describe the same nets! A third, symmetrical definition also describes the same nets: for each arc $p \to t$ of N, either

$${}^\bullet t = \{p\} \text{ or } p^\bullet = \{t\}. \tag{3}$$

17.2 The Trap/Cotrap Theorem for Free-Choice Nets



not live for any initial marking: the cotrap{A,C} does not contain any trap

Chapter 10 has shown that in each elementary system net with the trap/cotrap property, each reachable marking enables at least one transition. For free-choice nets with the trap/cotrap property, an even more powerful theorem holds: each transition can always become enabled again (the net is live, as defined in Chap. 10).

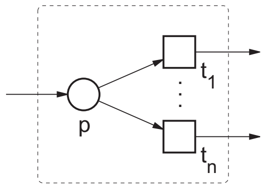
Theorem 27 (Trap/Cotrap Theorem for Free-Choice Nets)**.** *Let N be a free-choice net. Then N is live if and only if N has the trap/cotrap property.*

The proof is not easy. From this theorem, there immediately follows: if further tokens are added to the initial marking of a live free-choice net, the result is again a live net. This does not hold for common elementary system nets. A series of properties of free-choice nets can be decided comparatively efficiently.

17.3 Clusters

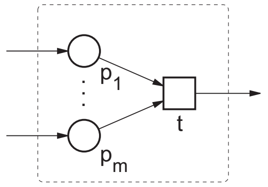
A cluster is a particularly simple subnet. It turns out that free-choice nets consist of such clusters.

A *cluster of a net* N is a subset of its places and transitions. There exist two different types:



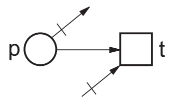
fc-cluster

1. For a place p with $p^\bullet = \{t_1, \ldots, t_n\}$ $(n \geq 0)$, let the input elements ${}^\bullet t_i$ of each transition t_i $(i = 1, \ldots, n)$ contain only p.[1] Then $\{p, t_1, \ldots, t_n\}$ is *free-choice cluster* of N: a token in p has a "free choice" between the transitions t_i.



ds-cluster

2. For a transition t with ${}^\bullet t = \{p_1, \ldots, p_m\}$ $(m \geq 0)$ let the post-set $p_j^\bullet$ of each place p_j $(j = 1, \ldots, m)$ contain only t. Then $\{t, p_1, \ldots, p_m\}$ is a *deterministic synchronization cluster* of N. The transition t occurs after it has "waited" for the places $p_1, \ldots, p_m$ to become marked.



The particularly simple structure $\{p, t\}$ with $p^\bullet = \{t\}$ and ${}^\bullet t = \{p\}$ is obviously a cluster of both types.

In Fig. 17.1, the structures $\{\mathsf{B}, \mathsf{a}, \mathsf{b}\}$ and $\{\mathsf{C}, \mathsf{c}, \mathsf{d}\}$ are free-choice clusters. $\{\mathsf{e}, \mathsf{D}, \mathsf{E}\}$ is a deterministic synchronization cluster, and $\{\mathsf{A}, \mathsf{f}\}$ is a cluster of both types.

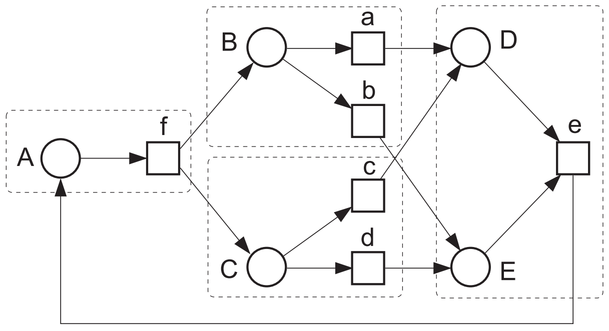
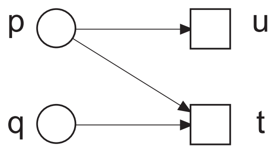


Figure 17.1: A free-choice net with four clusters

The clusters of a free-choice net N partition N:

Theorem 28 (Cluster Theorem for Free-Choice Nets)**.** *An elementary system net N is a free-choice net if and only if each place and each transition of N lies in exactly one cluster of N.*

[1]If $n = 0$, then $p^\bullet = \emptyset$.



The proof uses definition (3) of free-choice nets. If N is a free-choice net, then for each arc (p, t) there exist two possibilities: either ${}^\bullet t = \{p\}$, in which case there exists a free-choice cluster $(p, \ldots, t, \ldots)$, or $p^\bullet = \{t\}$, in which case there exists a deterministic synchronization cluster $(t, \ldots, p, \ldots)$.

If N is not a free-choice net, then there exist an arc (p, t), a transition u and a place q such that $t, u \in p^\bullet$ and $p, q \in {}^\bullet t$. Thus, p and t are not part of any cluster.

17.4 The Rank Theorem

A net that adequately models a real system is often live (each transition may always become enabled again) and bounded (no place may accumulate unboundedly many tokens). For a free-choice net N, liveness and boundedness can essentially be characterized with a connection between the clusters and the matrix of N: the rank of the matrix $\underline{N}$ of N is 1 less than the number of clusters of N. The rank of $\underline{N}$ is, as usual, the number of linearly independent columns of $\underline{N}$. This yields the following theorem:

Theorem 29 (Rank Theorem for Free-Choice Nets)**.** *For a connected free-choice net N, there exists an initial marking with which N is live and bounded if and only if*

(a) *N has a positive place invariant i, whose carrier contains each place of N*

(b) *N has a transition invariant j whose carrier contains each transition of N*

(c) *If the rank of $\underline{N}$ is k, then N has exactly $k + 1$ clusters.*

A comprehensible version of the nontrivial proof can be found in [15].

The free-choice net in Fig. 17.1 satisfies the conditions (a) and (b). However, its matrix has rank 6. The net thus violates condition (c), because it only has four clusters.

Exercises

1. Construct the clusters of the free-choice net in Fig. 17.2. Determine the type of each cluster.

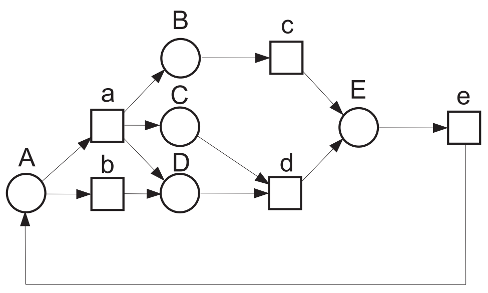


Figure 17.2: Free-choice net

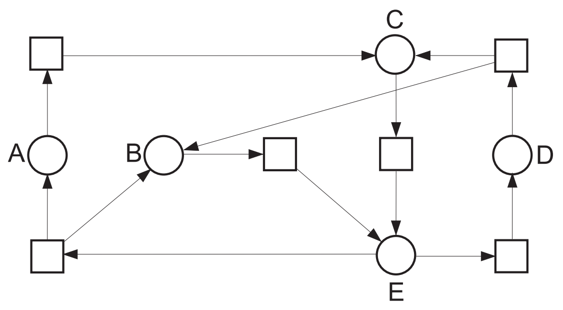


Figure 17.3: Free-choice net

2. Show that the free-choice net in Fig. 17.3 is live if its initial marking marks either of the places B, D or E.
3. Extend the answer to exercise 3(b) in Chap. 10.
4. Verify by means of the Rank Theorem whether there exists an initial marking for the free-choice net in Fig. 17.4 with which it is live and bounded.

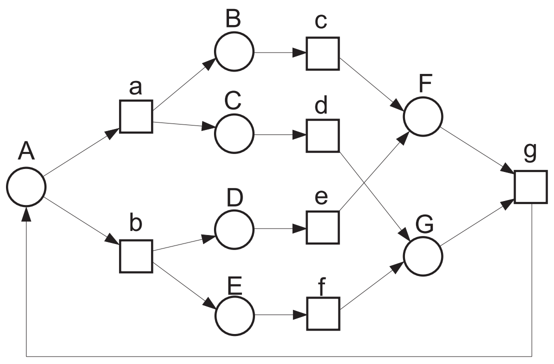


Figure 17.4: Free-choice net

Further Reading

As early as 1972, Michael H.T. Hack [33] defined free-choice nets and proposed them as schemata for the assembly and disassembly of composed objects. Furthermore, he defined the basic terms of the trap and cotrap for system nets and, with them, structurally characterized the live and safe free-choice nets (cf. Chapter 17, Theorem 28). The linear-algebraic connections with the Rank Theorem go back to the 1980s. Desel and Esparza compiled the theory of free-choice nets in [15].

Marked Graphs — Chapter 18

If the free-choice structure is further restricted such that it does not allow any choices at all, the result is a *marked graph*. This restriction greatly simplifies the analysis. In particular, for each marked graph N, there exists an initial marking such that N is live and 1-bounded.

18.1 Defining Marked Graphs

An elementary system net N is a *marked graph* if for each place p of N:

marked graph

$$|^\bullet p| = |p^\bullet| = 1.$$

Figure 18.1 shows a marked graph.

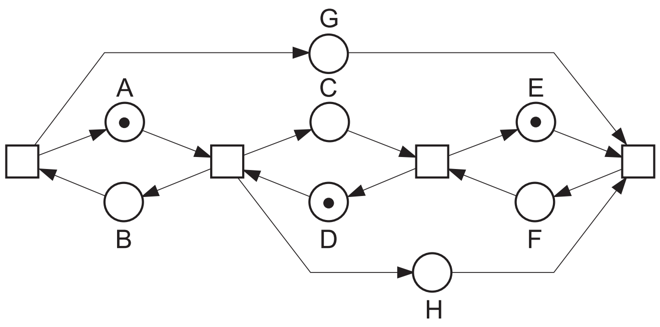


Figure 18.1: Marked graph

The central structural properties of marked graphs are paths and cycles. A *path* of N is a sequence $w = p_1 \dots p_n$ of mutually distinct places with

$$p_i{}^\bullet = {}^\bullet p_{i+1} \text{ for } (i = 1, \dots, n-1).$$

The sequence w is also a *cycle* if

cycle

$$p_n{}^{\bullet} = {}^{\bullet}p_1.$$

The marked graph in Fig. 18.1 has eight cycles: AB, CD, EF, ACDB, ACEFDB, CEFD, BGFD, HFD. It is easy to see that each cycle is a positive place invariant. Therefore:

Theorem 30 (Cycle Theorem for Marked Graphs)**.** *Let N be a marked graph and let $p_1 \dots p_n$ be a cycle of N, initially holding a total of k tokens. Then the following equation holds in N:*

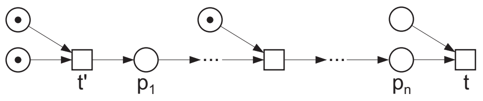
$$p_1 + \dots + p_n = k.$$

18.2 Liveness of Marked Graphs

Section 14.1 has defined the liveness of a system net N. A net N is live if for each transition t and each reachable marking M of N, the following holds: from M there can be reached a marking M' that enables t. For a marked graph, this property directly follows from its structure:

Theorem 31 (Liveness Theorem for Marked Graphs)**.** *A marked graph N is live if and only if each cycle of N contains at least one initially marked place.*

The if-part is obvious: without an initially marked place, a cycle $w = p_1 \dots p_n$ is unmarked in each reachable marking, because w is a place invariant. Thus, no transition in $p_i^{\bullet}$ can ever occur.



To show the only-if-part, let M be a reachable marking and t a transition of N. According to our assumption, each cycle of N contains at least one initially marked place. Thus, according to the above theorem, M also marks each cycle. Then there exists a path $p_1, \dots, p_n$ with $p_n^{\bullet} = t$ such that the transition t' in ${}^{\bullet}p_1$ is enabled in M. The step $M \xrightarrow{t'} M''$ reduces the length of this path. Via induction over the length of such a path, a marking that enables t is eventually reached.

Each of the eight cycles of the system net N in Fig. 18.1 contains at least one initially marked place. Thus, N is live.

18.3 1-Bounded Marked Graphs

Section 3.5 shows the important role of 1-bounded system nets. Live, 1-bounded marked graphs are easily characterized:

Theorem 32 (Theorem on Live and 1-Bounded Marked Graphs)**.** *A live marked graph N is 1-bounded if and only if each place of N is part of a cycle that initially contains exactly one token.*

We show the if-part indirectly: Let p be a place that only belongs to cycles initially containing more than one token. Because N is live, a marking M is reachable that enables the transition in $p^\bullet$ and thus marks p itself. We now temporarily delete this token from p. The net N stays live, because the resulting marking M' still marks each cycle. Therefore, another marking M'' that marks p is reachable from M'. With the temporarily deleted token, p now holds two tokens.

The only-if-part immediately follows from Theorem 30.

With the exception of the three cycles ACDB, CEFD and ACEFDB, each cycle of the system net N in Fig. 18.1 contains exactly one initially marked place. Each place of N lies on one of these cycles. Thus, N is 1-bounded.

18.4 Liveness of 1-Bounded Marked Graphs

The question arises: Which marked graphs are live and 1-bounded at the same time? The answer may be surprising: each strongly connected marked graph N can be marked such that N is live and 1-bounded. A net is strongly connected if for each two places there exists a path that connects them.

Theorem 33 (Theorem on Initial Markings of Marked Graphs)**.** *For each strongly connected marked graph N there exists an initial marking such that N is both live and 1-bounded.*

For a proof, we start with a marking M_0 that marks each cycle with at least one token: a simple task. If now there exists a place that only lies on cycles with more than one token, individual tokens are deleted according to the procedure described in the proof of Theorem 32 until a cycle with only one token remains. The graph in Fig. 18.1 is marked in this way.

Exercises

1. With the help of Theorems 31 and 32, test the marked graphs of the following figures for liveness and 1-boundedness:

   (a) Figure 18.2

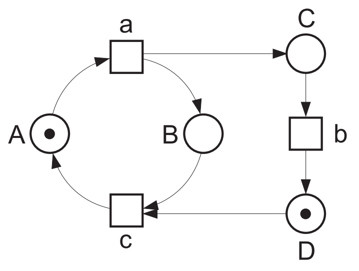


Figure 18.2: System net

   (b) Figure 18.1 with the initial marking M_0 = ADF.

2. Let N be a marked graph, let t and u be transitions and let p be a place of N. The transition t is *connected to* u *via* p if ${}^\bullet p = \{t\}$ and $p^\bullet = \{u\}$.

   N is *weakly connected* if for each two transitions t and u there exists a sequence $t_0, \ldots, t_n$ of transitions such that $t_o = t$, $t_n = u$ and, for $i = 1, \ldots, n$, either t_{i-1} is connected to t_i or t_i is connected to t_{i-1}.

   Show that for each transition invariant $m = (m_1, \ldots, m_k)$ of a weakly connected marked graph, the following holds: $m_1 = m_2 = \ldots = m_k$.

Further Reading

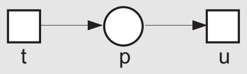
Marked graphs were introduced by Genrich in [27]. A rich theory on marked graphs was developed primarily in the 1970s.

The Customs Facilities Problem

A surprising application of Theorem 33 is the solution to the *customs facilities problem* [26]: Suppose a city has only one-way roads, designed such that a driver may reach each road from every other road. Now, *customs facilities* shall be established on some roads such that all drivers encounter at least one customs facility when driving a *circuit*, bringing them back to their initial road. Furthermore, for each road there shall exist a “Sunday circuit”: on such a circuit, a driver encounters exactly *one* customs facility.

The customs facilities problem pursues the question whether in every such city, customs facilities can be distributed such that these two conditions are met.

To solve this problem, each crossroads t is modeled as a transition, each road from a crossroads t to a crossroads u as a place p with arcs in the direction of travel, and each customs facility on a road p as a token in p.



At first, “sufficiently many” customs facilities are distributed such that each circuit has at least one. If then a road p only lies on circuits with two or more customs facilities, customs facilities are moved according to the step rule of transitions, analogously to the proof of Theorem 33. Eventually, two customs facilities will reach p. One of them is removed. This procedure is repeated until p lies on at least one circuit with exactly one customs facility. Thus, a Sunday circuit is established for p.

This example is notable for the fact that an inherently static problem is solved with a dynamic procedure.

Well-Formed System Nets — Chapter 19

Many real systems have a special state in which each individual *instance* of the system terminates. The system can then be newly instantiated.

A corresponding system net designates one of its reachable markings as the *final marking*. Furthermore, it satisfies three rather intuitive conditions:

- The final marking is reachable from any reachable marking.
- At the end of an instance, no tokens generated in between remain.
- Each transition can become enabled.

We will see how these three conditions can be verified rather easily.

19.1 Example: Models of Business Processes

A business process is a standardized process for implementing organizational procedures, as they usually occur in administrations. A business process contains activities that are connected to each other. In a Petri net model of a business process, each activity is modeled as a transition. The model itself is a *well-formed* system net.

Figure 19.1 shows a Petri net model of a business process for developing an offer. Its final marking has a token in the place stop.

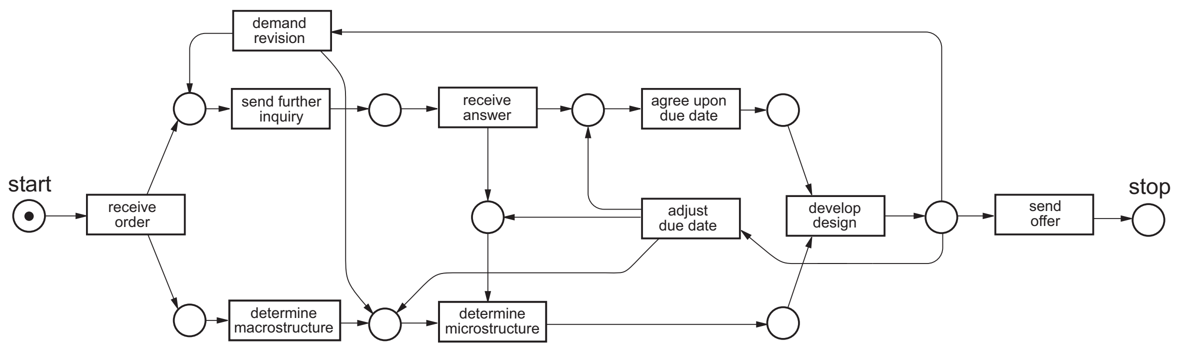


Figure 19.1: Business process for developing an offer

19.2 Well-Formed Elementary System Nets

Let N be an elementary system net and E an arbitrary reachable marking of N. Then N *together with the final marking* E *is well-formed* if N has the following three properties:

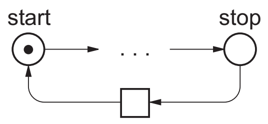
well-formed system net

- The final marking E is *unique*: no marking of the form $\underline{E} + \underline{L}$ is reachable if L marks at least one place.
- N is *weakly live*: for each transition t there exists a reachable marking that enables t.
- N is *terminable*: from each reachable marking, the final marking can be reached.

19.3 Deciding Well-Formedness

To decide whether an elementary system net N with the initial marking M_0 and the final marking E is well-formed, we expand N to a net N^* and decide whether N^* is live and bounded.

Specifically, we construct a new transition t_E. To this end, we construct for each place p an arc (p, t_E) with the arc weight $E(p)$ (cf. Section 6.2) and an arc (t_E, p) with the arc weight $M_0(p)$. Arcs with an arc weight of 0 are omitted. The addition of t_E to N yields N^*.



The final marking of the system net N in Fig. 19.1 has exactly one token in stop. Thus, the net N^* is generated from N by adding a new transition t such that ${}^\bullet t = \{\mathsf{stop}\}$ and $t^\bullet = \{\mathsf{start}\}$. The two (omitted) arc weights both have the value 1.

With the concepts of liveness (each transition can always become enabled again) and boundedness (for each place p there exists a natural number n such that $M(p) \leq n$ for each reachable marking M) the concept of well-formedness can be characterized:

Theorem 34 (Well-Formedness Theorem)**.** *An elementary system net N with a final marking E is well-formed if and only if N^* is live and bounded.*

This theorem defines well-formedness in terms of already-known concepts. The well-formedness of N in Fig. 19.1, for instance, can be verified with the Rank Theorem from Section 17.4, because N is a free-choice net.

Exercises

1. Does there exist a final marking E such that the system net in Fig. 19.2 is well-formed with E?

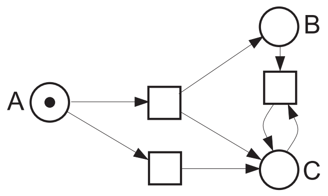


Figure 19.2: System net N

* 2. Prove the Well-Formedness Theorem.

3. Show by means of the Well-Formedness Theorem and the Rank Theorem that the system net in Fig. 19.1 with the final marking stop is well-formed.

Further Reading

The concept of well-formed elementary system nets and their usage for modeling business processes were introduced by van der Aalst [1]. He discusses additional variants and aspects of "well-formed" behavior and proves the Well-Formedness Theorem (Theorem 34).

Safety and Liveness

Since the 1970s, a system net has been called n*-safe* if each reachable marking marks each place with at most n tokens. We call such system nets n*-bounded* in this book. Back then, *liveness* and many of its variants stood for aspects of the reachability of markings and the possibility to enable transitions (starting from the initial marking, or an arbitrary reachable marking). Since the early 1980s, temporal logic has distinguished *safety properties* (intuitively: "something bad" never happens) and *liveness properties* (intuitively: eventually, "something good" happens). Each property of a system net can be composed from its safety and liveness properties [5]. A sensible description of the requirements of a system always includes properties of both types.

State and run properties, as defined in Chapters 9 and 16, are special safety and liveness properties, respectively. Usually, they are fully sufficient to describe the central requirements of a system model. Typical properties that cannot be formulated this way pertain to individual steps (for instance: "each step increases the number of tokens by 1"). However, such properties directly follow the step rule for transitions. Moreover, interesting system properties are interesting precisely because they do not only pertain to individual steps.

Part III

Case Studies

This selection of a few examples from hundreds of interesting case studies seems arbitrary and requires an explanation. We will introduce a variety of problems that cannot be solved with other modeling techniques, or at least not with the same set of principles.

In many contexts, *mutual exclusion* is one of the fundamental problems of distributed systems. Implicit assumptions often motivate a favored solution. Petri nets make these assumptions explicit.

The *Counterflow Pipeline Processor* (CFPP) is an extremely dynamic hardware architecture. Its asynchronous design becomes particularly clear in a Petri net model.

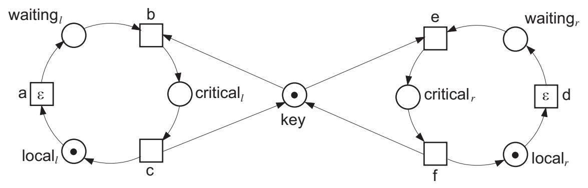
In an *agent network*, each agent only communicates with its respective neighbors without knowing the layout of the entire network. With very few exceptions, each agent uses the same local algorithm. This gives rise to the problem of how to model and verify *all* such networks with a single model. Petri nets offer adequate notations for this.

Each of the three case studies additionally introduces a new modeling or analysis technique: Mutual exclusion can only be modeled with the notion of *fairness*. For the CFPP, we analyze a solution to its synthesis problem. Agent networks use system net *schemata*.

Mutual Exclusion Chapter 20

By the 1960s, mutual exclusion was recognized as one of the central problems in the organization of computer systems. It arises whenever a resource can be accessed by only one of many processes at any one time (for instance, a processor, a printer or a communications device). While a process is using the resource, it is in its *critical* state. An agreement between the processes has to guarantee that no two processes can be in their critical states at the same time. We will introduce such an agreement in which processes can only communicate via asynchronous messages.

20.1 The Problem



With the system net in Fig. 3.2, we have already introduced a model that guarantees the mutual exclusion of the two processes l and r: they never reside in their critical states simultaneously. A *mutex algorithm* also guarantees that each process that wants to enter its critical state will eventually be able to do so. To specify this, we take another detailed look at the three states local, waiting and critical as well as the steps between them:

(1) State local: this is a state in which the process only works with its own data.

(2) Step from local to waiting: with this step a process announces its intention to use the limited resource. This step can be executed spontaneously at any time. A mutex algorithm cannot influence this step. A process is *not obliged* to execute this step. It may forever remain local. Thus, a is a cold transition.

(3) State waiting: in this step the process expects the mutex algorithm to permit it to enter its critical state. The algorithm may break down waiting into several intermediate states.

(4) Step from waiting to critical: a mutex algorithm may require certain preconditions to be met before a process can execute this step. Eventually, however, this step always has to be possible: the mutex algorithm has to allow this step (and its intermediate steps) not only for a few time intervals, but continuously until the process has reached critical. At the same time, the mutex algorithm can rely on the process to actually execute this step (and its intermediate steps). We will see that the algorithm in Fig. 3.2 does not guarantee this.

(5) State critical: this is the state in which the process uses the limited resource.

(6) Step from critical to local: the process may execute this step spontaneously at any time. A mutex algorithm cannot influence this step. A process is *obliged* to execute this step. It must not remain critical forever.

20.2 Realizability

An elementary system net that models a mutex algorithm (i.e., requirements (1)–(6) in the previous section) sensibly describes each of the three local states of each process with a 1-bounded place. (2) requires a cold transition from local to waiting for each process. (6) requires a hot transition from critical to local. Figure 20.1 outlines these connections. The “cloud” indicates the scope of the mutex algorithm.

We now assume an elementary system net N and derive a contradiction. With this, we show that no elementary system net can model a mutex algorithm.

Figure 20.2 shows a complete distributed run of the system net N outlined in Fig. 20.1. In this run, the left process l (top row) becomes critical infinitely often, while the right process r (bottom row) remains local forever. (For our purposes, r could

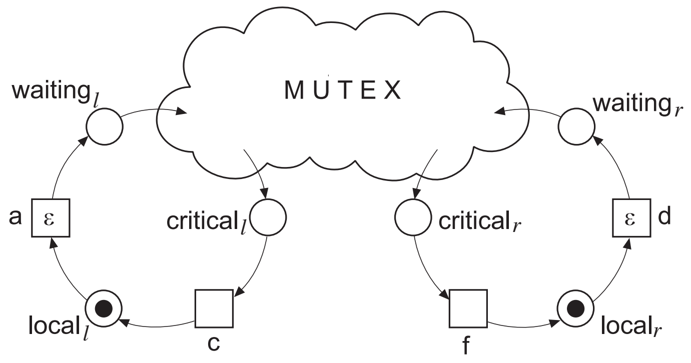


Figure 20.1: A few components of a mutex algorithm

become critical a finite number of times and then remain local.) This run meets all six requirements from Section. 20.1.

Figure 20.3 shows an extension of this run: the right process executed a step to waiting. This is possible, because of requirement (2). This run is also complete. However, it violates requirement (4). Therefore, N is not a mutex algorithm.

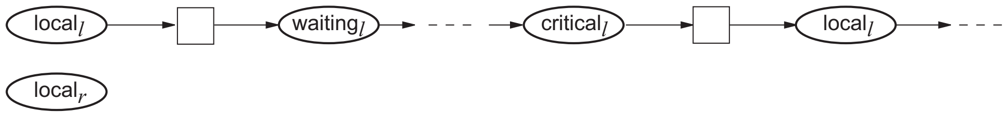


Figure 20.2: A valid run of a mutex algorithm

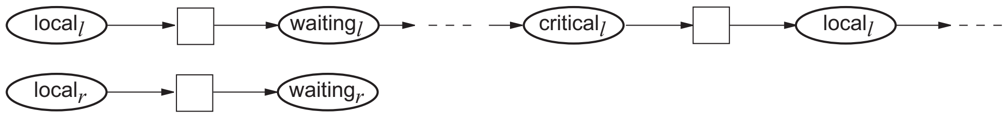


Figure 20.3: An invalid run

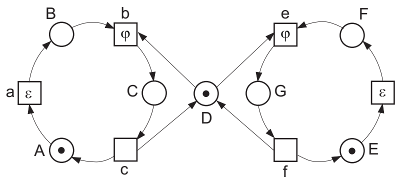
20.3 Fairness Assumptions

With the help of a new notation we can nevertheless construct mutex algorithms. To do this, we take another detailed look at the system net N in Fig. 3.2: it approximates a mutex algorithm A, because each “intended” run of A is also a run of

N. However, N also has additional, "unintended" runs, for instance (with a coherent renaming of the places and transitions), the sequential run

$$\mathsf{ADE} \xrightarrow{\mathsf{d}} \mathsf{ADF} \xrightarrow{\mathsf{a}} \mathsf{BDF} \xrightarrow{\mathsf{b}} \mathsf{CF} \xrightarrow{\mathsf{c}} \mathsf{ADF} \xrightarrow{\mathsf{a}} \mathsf{BDF} \xrightarrow{\mathsf{b}} \ldots$$

This run enables transition e an infinite number of times, without e ever actually occurring. This is (intuitively) an "unfair" treatment of e. Likewise, all runs that treat b unfairly are "unintended". This observation motivates the following definition of a sequential run $w = S_0 \xrightarrow{t_1} S_1 \xrightarrow{t_2} \ldots$ of an elementary system net N and a transition t of N: w *ignores fairness for* t if t is enabled in infinitely many markings S_i, but occurs only a finite number of times in w (for only a finite number of indices i: $t = t_i$). The run w *respects fairness for* t if w does not ignore fairness for t. A distributed run K *respects fairness for* t if each sequential run of K (as described in Section 4.1) respects fairness for t. To limit the set of runs of a system net N to those runs respecting fairness for a transition t, the transition t is labeled with "φ" in the graphical representation of N.



Thus, the system net N in Fig. 3.2 describes a correct mutex algorithm if the two transitions b and e are labeled with "φ".

Even without formally defining "implementability", it is easy to see that fairness cannot be implemented. The observation that fairness cannot be approximated with finite means makes this particularly clear: each finite initial segment of an unfair distributed run can be continued as a fair distributed run. Whether or not fairness is correctly implemented could only be decided "after an infinite amount of time." The assumption of fairness for transitions thus increases the expressive power of Petri nets.

However, a fairness requirement of the form "the transition t eventually occurs" can be implemented more strictly as "t occurs after at most k cycles." Such an algorithm regulates the flow of tokens in $^\bullet t$. For the left process this applies to transition b with the places B and D. The algorithms of both processes access the place D. Thus, they have to synchronize themselves. Instead, we look for a mutex algorithm whose transitions that require fairness have their pre-sets entirely *within* the respective processes.

20.4 Mutex with Autonomous Fairness

We construct an algorithm with a *token* that only one of the processes can possess at any one time. The process possessing the token can immediately execute its step from waiting to critical. The process not possessing the token can, while waiting, send a message to its partner to request the token. As soon as the partner yields the token, the process will become critical. A process possessing the token, when receiving such a request from its partner, will eventually yield the token.

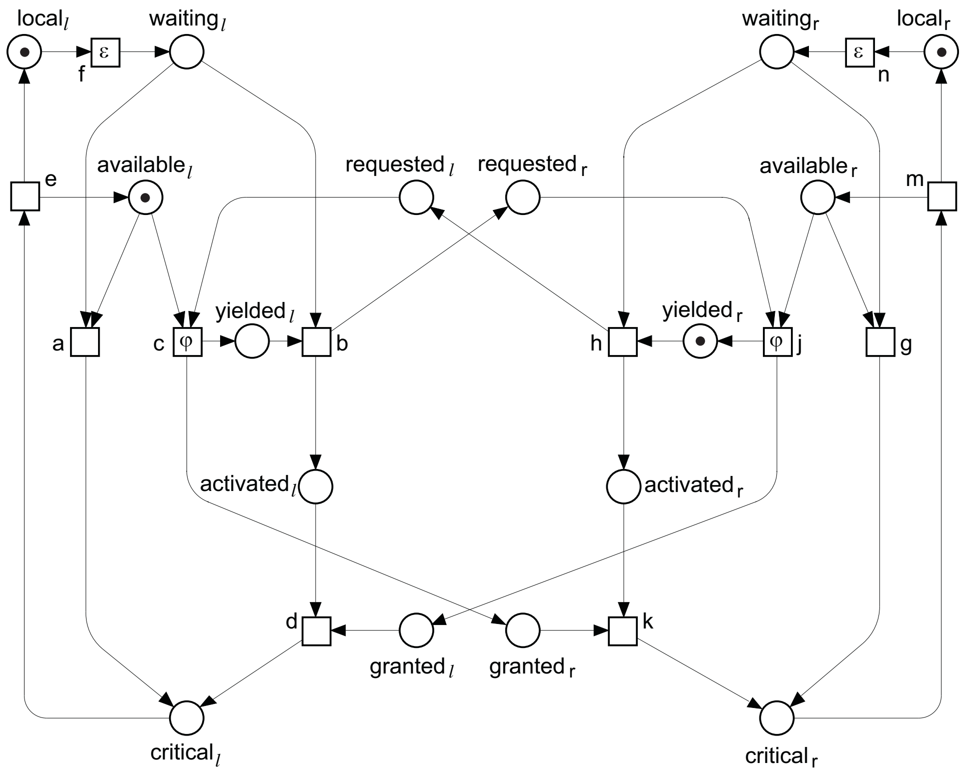


Figure 20.4: Message-based mutex

Figure 20.4 replaces the “cloud” in Fig. 20.1 and shows the algorithm. In the given initial marking, the left process is in possession of the token ($\mathsf{available}_l$). Therefore, the left process can immediately execute the step from $\mathsf{waiting}_l$ to $\mathsf{critical}_l$ via transition a. The right process does not possess the token, but can, once it reaches $\mathsf{waiting}_r$, execute the step to $\mathsf{activated}_r$ via transition h, thereby sending a message to the left process ($\mathsf{requested}_l$). The left process can respond to this message by yielding the token via transition c ($\mathsf{yielded}_r$), whereupon the right process reaches $\mathsf{critical}_r$ via transition k. The label "φ" (indicating "fairness") in transition c rules out "unfair" behavior. The left process would be unfair if it always ignored a pending request from the right process ($\mathsf{requested}_l$), i.e., the choice between transitions a and c was always decided in favor of a and thus to the disadvantage of c.

20.5 The Scenarios of the Algorithm

For a systematic understanding of the algorithm, it is helpful to examine its scenarios. In fact, the algorithm is scenario-based. With the denotations in Fig. 20.5, the scenario S_l in Fig. 20.6 describes the cycle in which the owner of the token (i.e., in Figs. 20.4 and 20.5 the left process) becomes $\mathsf{critical}$. The corresponding scenario S_r (with a token initially in M instead of N) is obvious.

Figure 20.7 shows the partial run A_r in which the right process

- requests the token via h,
- receives the token and becomes $\mathsf{critical}$ via k
- and yields the token via j.

The corresponding partial run A_l (with a token initially in C instead of B) is obvious.

Figure 20.8 shows the partial run B_l in which the left process

- yields the token via c,
- requests the token via b

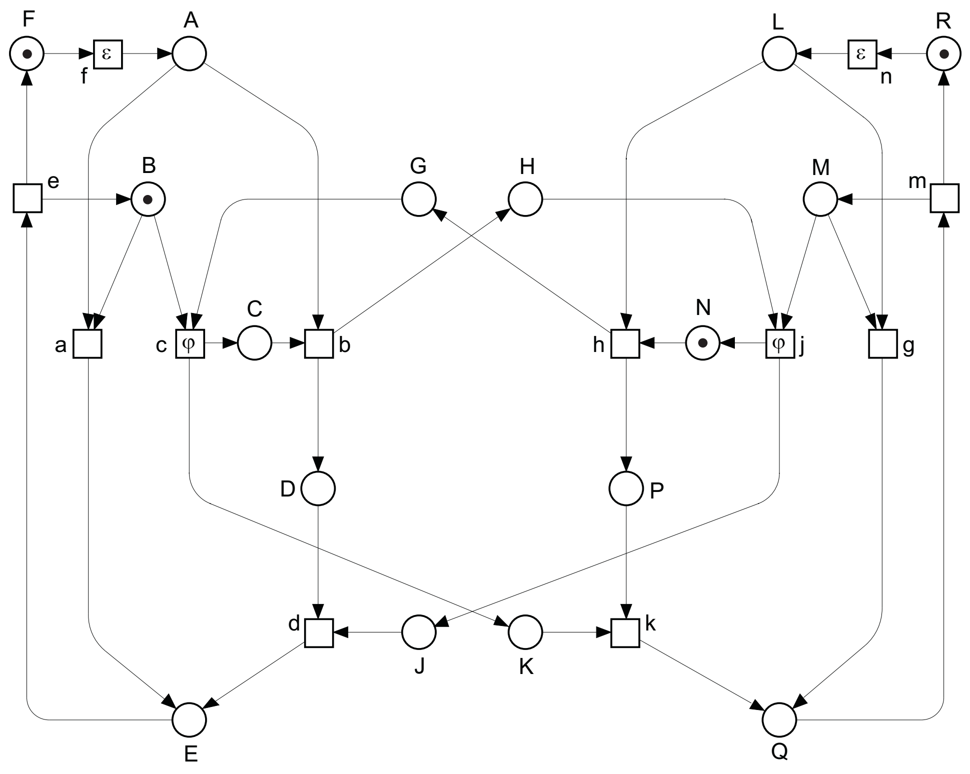


Figure 20.5: Renaming of places and transitions

- and receives the token and becomes critical via d.

The corresponding partial run B_r (with a token initially in M instead of N) is obvious.

The communication channels G, K, H and J of the algorithm occur in both A_r and B_l as labeled places. By merging those places of A_r and B_l that have the same label, a scenario, S_1, is generated in which at first the right and then the left process becomes critical. Likewise, let the scenario S_2 be composed of A_l and B_r.

Each distributed run of the system net in Fig. 20.5 consists of instances of the four scenarios S_l, S_r, S_1 and S_2. Thus, the mutex algorithm is indeed scenario-based.

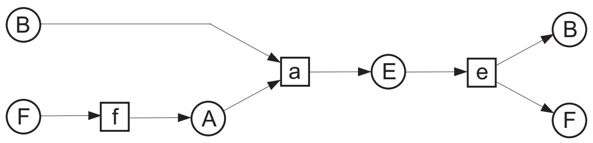


Figure 20.6: The scenario S_l

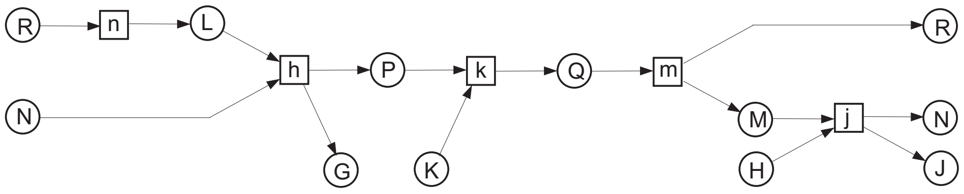


Figure 20.7: The partial run A_r

20.6 Correctness of the Algorithm

As for each mutex algorithm, we have to show that, next to the actual mutual exclusion property, the six requirements from Section 20.1 are satisfied by the algorithm in Fig. 20.4. With the exception of requirement (4) they are all obviously fulfilled.

With the denotations in Fig. 20.5, the logical expression $\neg(\mathsf{E} \wedge \mathsf{Q})$ and thus the inequality

$$\mathsf{E} + \mathsf{Q} \leq 1 \tag{1}$$

describes the mutual exclusion property. Its validity follows immediately by subtracting the canonical inequalities of B, K, M and J from the place invariant

$$\mathsf{E} + \mathsf{B} + \mathsf{K} + \mathsf{Q} + \mathsf{M} + \mathsf{J} = 1.$$

Next to the state property (1), the algorithm has to satisfy the run property (4) from Section 20.1: in each run, each waiting process will eventually become critical. In Fig. 20.5, for the left process, this is the property

$$\mathsf{A} \mapsto \mathsf{E}. \tag{2}$$

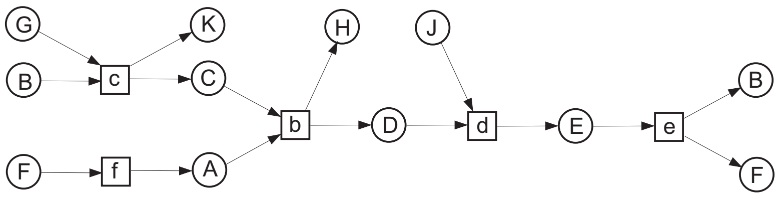


Figure 20.8: The partial run B_l

We will show (2) by means of a proof graph, as introduced in Chapter 16. Figure 20.9 shows this proof graph. The correctness of both proof graphs follows from the reading rule and place invariants, with the exception of step 5 in Fig. 20.9: HM $\mapsto$ J. This step needs a rule exploiting the fairness assumption for transition j. It prevents the infinite repetition of the cycle M $\xrightarrow{g}$ Q $\xrightarrow{m}$ M. Such a rule is described in [64].

$$\begin{tikzcd}
\text{1.A} \arrow[r] \arrow[rr, bend left] & \text{2.AB} \arrow[r, mapsto, "c"] \arrow[rrrrrr, bend right, mapsto, "a"'] & \text{3.AC} \arrow[r, mapsto, "b"] & \text{4.H} \arrow[r, mapsto, "\text{Fig. 20.10}"] & \text{5.HM} \arrow[r, mapsto, "j"] & \text{6.J} \arrow[r] & \text{7.JD} \arrow[r, mapsto, "d"] & \text{8.E}
\end{tikzcd}$$

Figure 20.9: Proof graph for $A \mapsto E$

$$\begin{tikzcd}
\text{1.H} \arrow[r] \arrow[rrr, bend left] \arrow[rrrr, bend right] & \text{2.HK} \arrow[r] & \text{3.HKP} \arrow[r, mapsto, "k"] & \text{4.HQ} \arrow[r, mapsto, "m"] & \text{5.HM}
\end{tikzcd}$$

Figure 20.10: Proof graph for $H \mapsto HM$

Exercises

1. Show the correctness of the proof graphs in Figs. 20.9 and 20.10.
2. Construct the scenarios S_1 and S_2 described in Section 20.5.

Further Reading

A series of solutions to the mutual exclusion problem originally formulated in programming languages were also reproduced as Petri nets [64]. The postscript to Chapter 4 discusses some general problems of such Petri net reproductions. Kindler and Walter show in [40] that each mutex algorithm for autonomous processes requires some form of fairness assumption.

The Problem of Mutual Exclusion

The problem of mutual exclusion occurs in many variations. Those variations differ particularly with respect to:

- the number of processes and the topology of their networking: Chapter 20 introduced a solution for *two* processes, Section 9.3 for five processes (“philosophers”). If we call two processes *neighbors* if they share a limited resource (“chopstick”), then the neighbor relation of the five philosophers forms a circle. In general, there exists an arbitrary number of processes and very different patterns of the neighbor relation. Apart from circles, other common patterns are stars, grids, trees and n-dimensional cubes (“hypercubes”). Furthermore, the number of simultaneous users of a limited resource is not necessarily two.

- the assigning of the limited resource: so far, we have only seen examples in which each limited resource has a predefined set of users. This is not always the case. For instance, it may be irrelevant to which of two available printers a process sends its documents.

- the communication between the processes: in the solution in Fig. 20.4, the processes communicate via messages. In other solutions, they access a shared variable or execute joint activities.

- the influence of a global instance: an example of such an instance is an operating system managing the processes and resources. In contrast, the solution in Chapter 20 uses homogeneous processes, which are not controlled by any global instance.

The formulations of and solutions to the different variants of the problem depend heavily on the modeling technique employed. Shared variables are commonly used. Thereby, each variable is owned by a process that can update it. Other processes may only read the variable. The necessary fairness assumptions are integrated into the modeling language itself: an unbounded sequence of reading operations of a variable x can be interrupted by the variable’s owner in order to update x. During the modeling of such variables with Petri nets (cf. Chapter 4’s postscript “Read and Write vs. Give and Take”) these fairness assumptions become obvious. In Lamport’s “Bakery Algorithm”, the fairness assumption is incorporated into a display panel of which all processes require an unimpeded view.

In practice, such fairness assumptions can often be neglected: “long-lasting” accesses to limited resources can be organized by “short” accesses to other resources.

Asynchronous Hardware Chapter 21

This case study describes the systematic construction of asynchronous hardware. It uses the example of an extremely dynamic architecture of a processor that adapts to the current availability of data packets and functions, as well as the varying durations of individual calculation steps.

21.1 The Counterflow Pipeline Processor (CFPP): The Problem

The Sprout counterflow pipeline processor is a well-known, complex, asynchronous hardware architecture. We model the abstract principle of its behavior on the level of its data and control flows: a stream $d_1 \ldots d_n$ of data packets and a stream $f_1 \ldots f_m$ of instructions both reach a processor P. The processor then sequentially applies all instructions $f_1, \ldots, f_m$ to each data packet d_i and then outputs the resulting data packets

$$e_i =_{\text{def}} f_m(\ldots f_2(f_1(d_i)) \ldots)$$

for $i = 1, \ldots, n$ in sequence. After P has processed a pair of data and instruction streams, it becomes ready for new pair.

In each case, the two streams may be of different lengths. The data packets and instructions reach P asynchronously, not in predetermined time intervals. The time it takes to apply f_j to the data packet

$$f_{j-1}(\ldots f_2(f_1(d_i)) \ldots)$$

is unknown for all i and j. The processor should work as fast as possible, that is, it should execute as many operations as possible in parallel. At the same time, it should react as flexibly as possible to different speeds and different lengths of data and instruction streams, as well as different durations of instruction calculations.

21.2 The Solution Idea

To solve this problem, the CFPP architecture utilizes a sequence $P = M_1 \ldots M_k$ of consecutively linked modules, as outlined in Fig. 21.1. The data packets $\mathsf{d}_1, \ldots, \mathsf{d}_n$ flow from

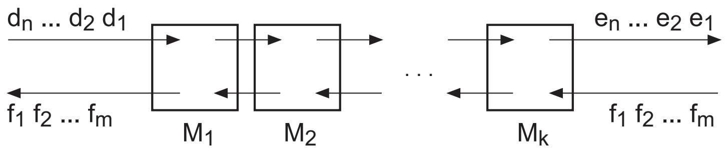


Figure 21.1: Assembly of the CFPP from modules M_i

the left, that is, via M_1, into P. The computed data packets $\mathsf{e}_1, \ldots, \mathsf{e}_n$ leave P on the right, via M_k. Vice versa, the instructions flow from the right into P and leave P via M_1. All modules work according to the same pattern. A module M_i can receive data packets from its left neighbor M_{i-1} and give out instructions to it. To its right neighbor M_{i+1}, it can give out data packets and receive instructions from it. The modules communicate synchronously: M_i can give out a data packet to M_{i+1} as M_{i+1} receives it. Likewise, M_i gives out an instruction to M_{i-1} as M_{i-1} receives it.

Figure 21.2 shows the behavior of an “inner” module $\mathsf{M}_i (i = 2, \ldots, k-1)$ as a state automaton. Initially, M_i can receive a data packet d (from M_{i-1}) and an instruction f (from M_{i+1}). When M_i has received both – in arbitrary order – it is ready for computation and applies f to d. M_i can then – in arbitrary order – give out the newly computed data packet and the instruction to M_{i+1} and M_{i-1}, respectively. M_i does not store anything, then reorganizes itself and returns to its initial state.

Of particular interest are the two states (top and bottom in Fig. 21.2) in which M_i stores either only a data packet or only an instruction: M_i can give out the unprocessed data packet to M_{i+1}, or the unused instruction to M_{i-1}.

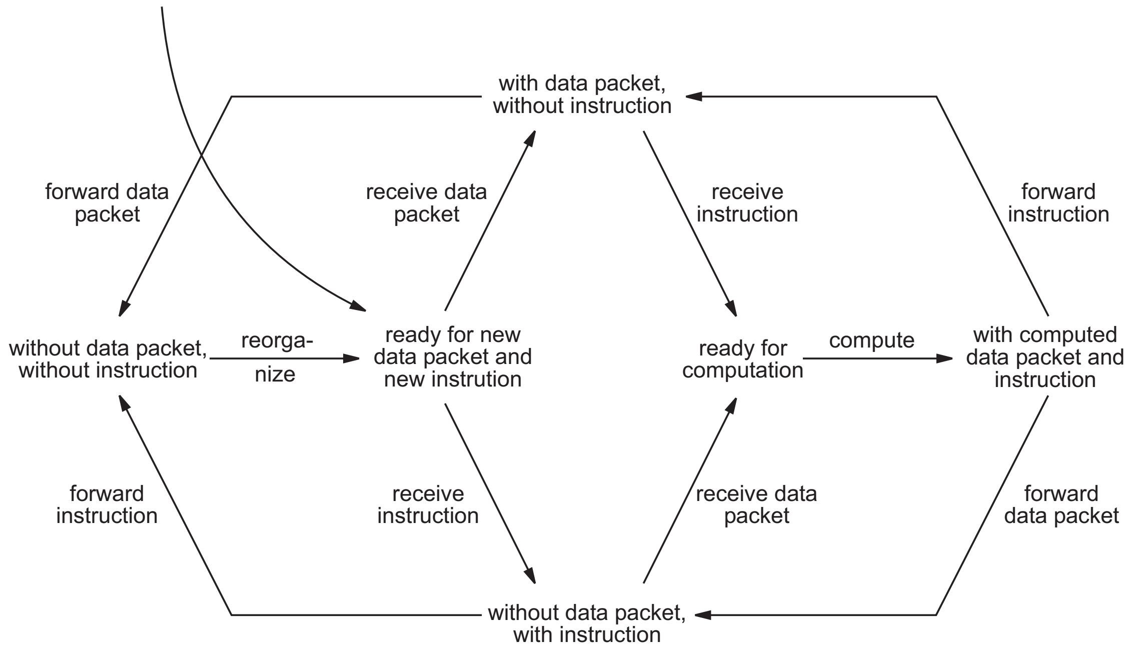

with data packet, without instruction
forward data packet
receive data packet
receive instruction
forward instruction
without data packet, without instruction
reorga-nize
ready for new data packet and new instrution
ready for computation
compute
with computed data packet and instruction
forward instruction
receive instruction
receive data packet
forward data packet
without data packet, with instruction

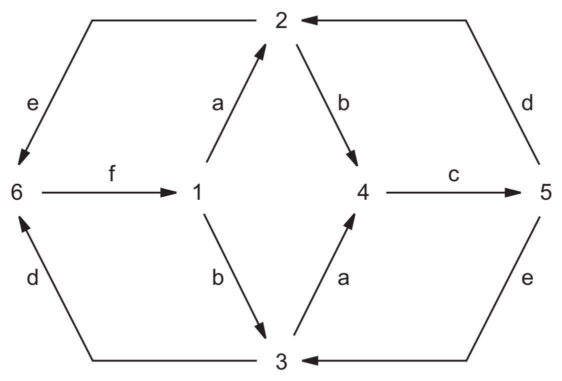
Figure 21.2: A CFPP module as state automaton

The module M_1 on the left edge of the CFPP architecture essentially behaves like an inner module. However, it only gives out an instruction f_i to the environment after f_i has been applied to the last data packet d_n. Likewise, the module M_k on the right edge only gives out a data packet e_i to the environment after the last instruction f_m has been applied to e_i. We do not explicitly model those two modules.

An architecture with k modules can process a stream $\mathsf{d}_1, \ldots, \mathsf{d}_n$ of data packets and a stream $\mathsf{f}_1, \ldots, \mathsf{f}_m$ of instructions if and only if n and m are not both greater than k: In that case, the data packets or instructions can all be stored simultaneously inside the modules. A CFPP can compensate for the different durations of instructions only if k is greater than either n or m.

21.3 The Synthesis Problem for the CFPP

The state automaton in Fig. 21.2 uses six different actions, four of which can occur in two states each. We now ask for the pre-,



$$A = \{\,2,4,5\,\},\quad B = \{\,3,4,5\,\},$$
$$C = \{\,1,3,6\,\},\quad D = \{\,1,2,6\,\},$$
$$E = \{\,1,3,5\,\},\quad F = \{\,2,4,6\,\},$$
$$G = \{\,5,1,2\,\},\quad H = \{\,3,4,6\,\}$$

Figure 21.3: The eight minimal regions $A, \ldots, H$ of a CFPP module

post- and side conditions of their occurrences and thus for the local states that organize a CFPP module. For this purpose, we solve the synthesis problem for the state automaton (cf. Chapter 7). Using the denotations in Fig. 21.3, its eight minimal regions generate the elementary system net N shown in Fig. 21.4. Its marking graph is isomorphic to the state automaton in Fig. 21.2. Therefore, N solves the synthesis problem for the CFPP.

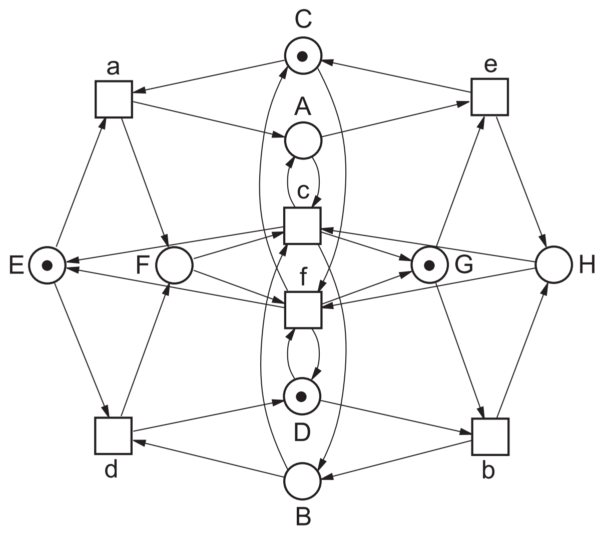


Figure 21.4: The solution to the synthesis problem for the CFPP

21.4 Structural Simplification of a Module

The structure of the system net N in Fig. 21.4 can be simplified. At first, we derive for this:

$$\begin{aligned} \mathsf{A}+\mathsf{H}+\mathsf{E}+\mathsf{D} &= 2 && \text{place invariant} \\ \mathsf{E}+\mathsf{F} &= 1 && \text{place invariant} \\ -\mathsf{B}-\mathsf{D} &= -1 && \text{place invariant} \\ -2\mathsf{E} &\leq 0 && \text{canonical inequality of } \mathsf{E} \end{aligned}$$

Their addition yields:

$$\mathsf{A}+\mathsf{F}+\mathsf{H}-\mathsf{B} \leq 2.$$

From this follows for each reachable marking that marks A, F and H, that it also marks B. This renders the loop between B and c redundant.

The argument about the loop between C and f is analogous. Thus, Fig. 21.5 shows the final version of a CFPP module.

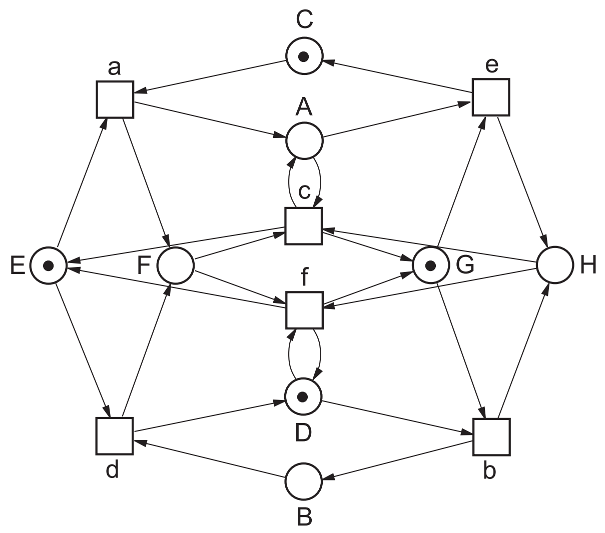


Figure 21.5: Final version of a module

21.5 The Model of the CFPP

To form the CFPP P as a sequence $\mathsf{M}_1 \ldots \mathsf{M}_k$ of modules, k instances of the module in Fig. 21.5 have to be combined. It is rather simple to combine a module M_i with its right neighbor M_{i+1}: the transition e of M_i is merged with the transition a of M_{i+1} and b of M_i with d of M_{i+1}. Figure 21.6 outlines this construction.

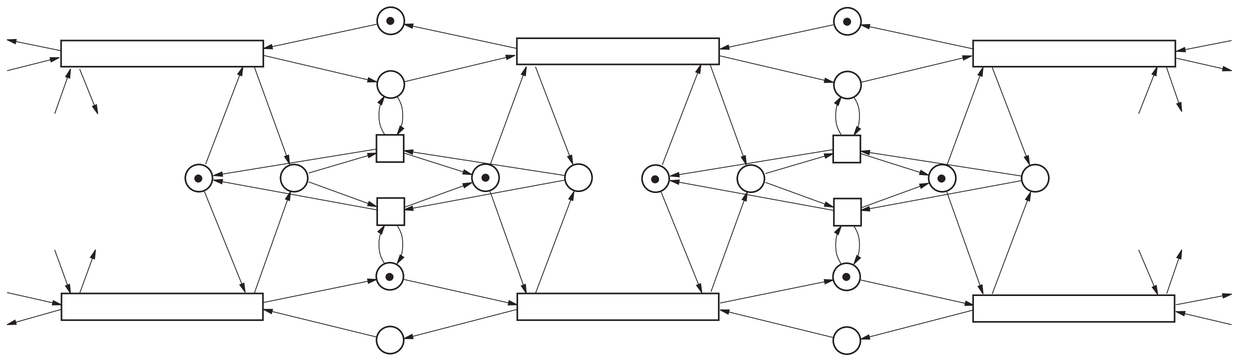


Figure 21.6: Combination of modules M_i and M_{i+1} of the CFPP

21.6 Analysis of the Model

For a better understanding of the model in Fig. 21.5, the model in Fig. 21.7 describes the intuitive meanings of some of its components. An intuitive description of the meanings of the places E, F, G and H is left to the reader.

The arcs labeled with the variable x describe the path of the data packets through the processor. Likewise, the arcs labeled with y describe the path of the instructions. The invariant $\mathsf{A} + \mathsf{C} = 1$ in Fig. 21.5 guarantees that data packets and instructions cannot overtake each other. Accordingly, $\mathsf{B} + \mathsf{D} = 1$ holds for instructions: a module M, indeed, never stores more than one data packet d and never more than one instruction f. If M_i stores both, the module computes, i.e., f is applied to d. From that follows for each reachable marking M in Fig. 21.5: if a data packet has been received, that is transition a has occurred, i.e., $\mathsf{M(A)} = \mathsf{M(F)} = 1$, and if an instruction has been received, that

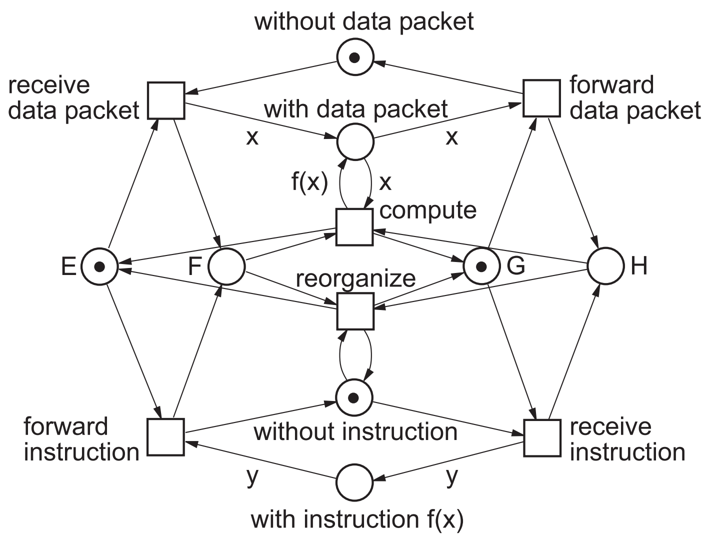


Figure 21.7: Module of the CFPP

is, transition b has occurred, i.e., $\mathsf{M(B)} = \mathsf{M(H)} = 1$, then c is the only enabled transition. Because of the invariants $\mathsf{A} + \mathsf{C} = 1$, $\mathsf{E} + \mathsf{F} = 1$, $\mathsf{G} + \mathsf{H} = 1$ and $\mathsf{B} + \mathsf{D} = 1$, the places C, E, G, D are not marked and thus the transitions a, d, e, b, f are not enabled.

If M_i stores a data packet d and M_{i+1} an instruction f, both of the shared transitions are enabled. In that case, it is decided nondeterministically whether M_i or M_{i+1} computes f(d).

Exercises

1. Describe the intuitive meanings of the places E, F, G and H in Fig. 21.7.

* 2. Construct the model of the module in Fig. 21.5 for two neighboring modules M_i and M_{i+1}. Let M_i store a data packet, but no instruction, and let M_{i+1} store an instruction, but no data packet. Show that M_i and M_{i+1} share two transitions that are in conflict with each other if both are enabled.

Further Reading

The concept of the Sprout CFPP and the modeling idea of this chapter are described by Yakovlev and Koelmans in [81]. Problems with the synthesis of distributed systems from a sequential description of their behavior often occur in hardware design. Algorithms for their solution have therefore been integrated into the *Petrify* tool [13]. Petri nets are successfully used as a modeling tool for hardware systems.

Network Algorithms

Chapter 22

Nowadays, computers often function as nodes of a network. In such a network, two nodes can be connected to each other by a *communications channel* via which they can exchange messages. Such nodes are *neighbors*.

Figure 22.1 shows two networks, N_1 and N_2, with the nodes $\alpha, \beta, \ldots$. Both are connected. In contrast to N_1, the network N_2 is *acyclic*: no sequence of edges forms a cycle.

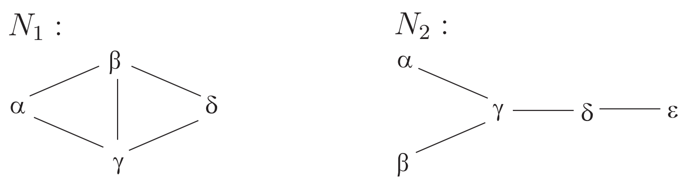


Figure 22.1: Two agent networks

In a network, there exist some typical tasks: a node sends a message over the network to all other nodes, the nodes synchronize themselves (as far as possible) in cycles or they reach mutual agreements. In this chapter, we discuss solutions to such tasks. Other typical tasks relate to the organization of limited resources, the distribution of tasks or the reinitialization after a critical error.

Such tasks are not easy to solve, because each node generally only has a small number of neighbors. No node "knows" the entire network. Furthermore, a solution is supposed to work not only for a specific network, but for infinitely many, for instance, for all connected or for all acyclic networks.

22.1 Some Conventions for the Representation of Network Algorithms

We model algorithms for the solution of the types of tasks described above as Petri net *schemata*. Such a schema is a net structure, labeled with *symbols* that can be *interpreted*. Each sensible interpretation generates its own concrete, finite agent network.

We typically use the symbol U to denote the set of agents in a network. The neighbors of a network are given as a relation

$$N \subseteq U \times U.$$

A tuple $(u, v) \in N$ thus describes a communications channel. We assume symmetrical communications channels, which means that $(u, v) \in N$ implies $(v, u) \in N$. In the network N_1 in Fig. 22.1, $U = [\alpha, \beta, \gamma, \delta]$ and $N = [(\alpha, \beta), (\beta, \alpha), (\alpha, \gamma), (\gamma, \alpha), (\beta, \delta), (\delta, \beta), (\beta, \gamma), (\gamma, \beta), (\gamma, \delta), (\delta, \gamma)]$.[1]

We also use a tuple (u, v) to denote a message from v to u. In general, a message is always of the form

$$(\textit{recipient}, \textit{sender}).$$

For an agent u, let $N(u)$ be the set of messages (v, u) to all its neighbors v and let $\overline{N}(u)$ be the set of messages (u, v) *from* all its neighbors v. Thus, in the network N_1 in Fig. 22.1

$$N(\alpha) = [(\beta, \alpha), (\gamma, \alpha)] \text{ and} \tag{1}$$

$$\overline{N}(\alpha) = [(\alpha, \beta), (\alpha, \gamma)]. \tag{2}$$

With these conventions, we first explain the idea of Petri net schemata, using the example of the *Echo Algorithm*.

22.2 The Echo Algorithm

For a connected network with a distinguished "initiator" node i, this algorithm organizes the distribution of a message from

[1]We denote U and N as well as several other sets as multisets here, in order to stay consistent with the arc labelings of the nets.

the initiator to all other nodes of the network. The initiator terminates when all nodes have confirmed the receipt of the message.

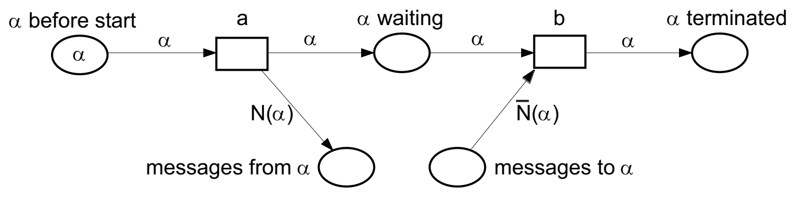


Figure 22.2: The behavior of the initiator α

As an example, let α be the initiator of the network N_1 in Fig. 22.1. Figure 22.2 shows its behavior: the transition a sends messages from α to the neighbors β and γ. According to (1), this generates the tokens (β, α) and (γ, α) in the place "messages from α." When later the place "messages to α" has received the tokens (α, β) and (α, γ) (cf. (2)), transition b can occur. In this representation, N and $\overline{\mathsf{N}}$ are merely symbols. They first have to be interpreted according to the specification in Section 22.1 before one can talk about markings.

A non-initiator node becomes active as soon as it receives its first message. It then chooses one of its neighbors as its "pivot neighbor." Figure 22.3 shows the behavior of the node γ (see N_1 in Fig. 22.1) with α as its pivot neighbor: the transition c sends messages to the two other neighbors β and δ of γ by generating the set of tokens $\mathsf{N}(\gamma) - (\alpha, \gamma) = [(\alpha, \gamma), (\beta, \gamma), (\delta, \gamma)] - (\alpha, \gamma) = [(\beta, \gamma), (\delta, \gamma)]$ in the place "messages from α." Then γ waits, with its pivot message (γ, α) in the place "γ waiting with α," for messages from the two other neighbors, that is, for tokens in the set $\overline{\mathsf{N}}(\gamma) - (\gamma, \alpha) = [(\gamma, \alpha), (\gamma, \beta), (\gamma, \delta)] - (\gamma, \alpha) = [(\gamma, \beta), (\gamma, \delta)]$. The occurrence of d then removes all tokens from the places "γ waiting with α" and "messages to γ," and terminates γ.

Instead of α, the node γ can also choose β as its pivot neighbor if the message (γ, β) reaches the place "messages to γ." In principle, each neighbor of a node can play the role of the pivot neighbor.

The net in Fig. 22.4 models this possibility by means of the variable y. The current choice of a pivot neighbor determines

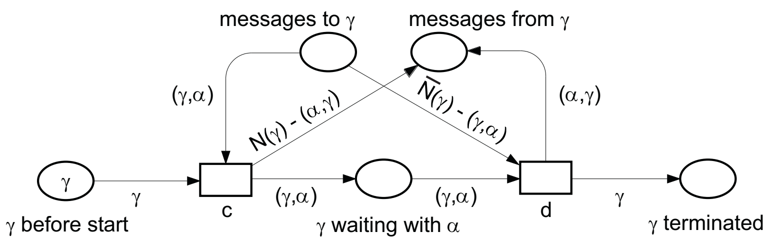


Figure 22.3: The behavior of γ with the pivot neighbor α

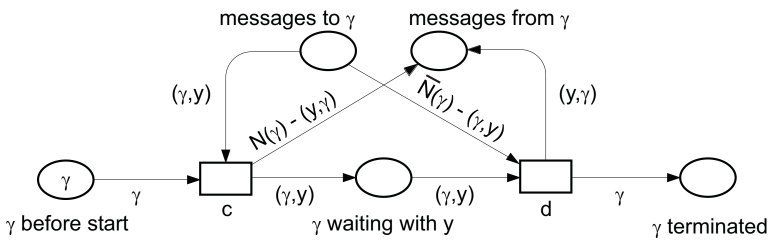


Figure 22.4: The behavior of a non-initiator agent γ

the value of y and thus the mode of transition c (cf. Sect. 2.6).

Each non-initiator node behaves according to the pattern given in Fig. 22.4 for the node γ. The net in Fig. 22.5 models this by making two modifications to the net in Fig. 22.4: first, the concrete node γ is replaced with the variable x. Second, the initial marking γ of the place "γ before start" is replaced with the symbol U'. With this, we model the behavior of arbitrary networks. U denotes the set of all nodes and $\mathsf{U}' = \mathsf{U} - \mathsf{i}$ the set of all nodes except the initiator. Thus, for N_1 in Fig. 22.1, the following holds:

$$\mathsf{U} = [\alpha, \beta, \gamma, \delta],\ \mathsf{i} = \alpha,\ \mathsf{U}' = [\beta, \gamma, \delta].$$

An agent u with only one neighbor v (for instance, α, β or ε in N_2 in Fig. 22.1) can only choose v as its pivot neighbor. In this case, the transition c in Fig. 22.5 only has the mode in which $\mathsf{x} = \mathsf{u}$ and $\mathsf{y} = \mathsf{v}$. The arc label $M(\mathsf{x}) - (\mathsf{y}, \mathsf{x})$ is then reduced to $\mathsf{N}(\mathsf{u}) - (\mathsf{v}, \mathsf{u}) = (\mathsf{v}, \mathsf{u}) - (\mathsf{v}, \mathsf{u}) = [\,]$. The transition c thus does not send any message. Accordingly, d also does not expect a message.

For the final model of the Echo Algorithm, the four places for sent, but not yet received messages in Figs. 22.2 and 22.5 are combined into a *single* place D in Fig. 22.6. Furthermore, the agent γ is replaced with the symbol i for the initiator.

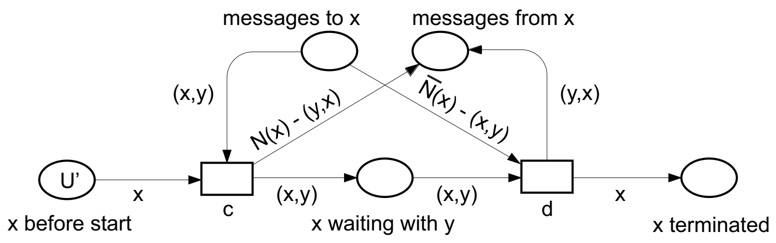


Figure 22.5: The behavior of the non-initiator nodes

Figure 22.6 does not show an example of system nets as they have been introduced in Sect. 2.7. To begin with, i, U′, N and $\overline{\mathsf{N}}$ are merely symbols. To describe a concrete agent network A, as explained in Sect. 22.1, the symbol i has to be interpreted as the initiator, U′ as the set of all other agents of A, and N as well as $\overline{\mathsf{N}}$ as functions for the generation of the network's messages. Thus, Fig. 22.6 forms a *schema* of an algorithm for an *arbitrary* agent network.

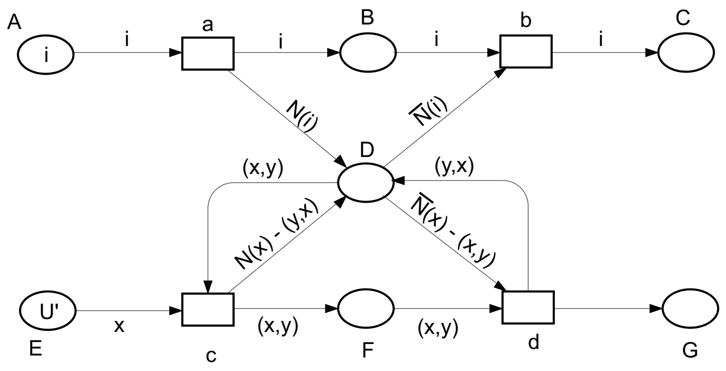


Figure 22.6: The Echo Algorithm

On this schema level, we can now show the correctness of the algorithm for each concrete connected network A. Two properties characterize the correctness of the algorithm:

1. The state property "If the initiator terminates, all other nodes

have been informed":

$$N \models \mathsf{C.i} \longrightarrow \mathsf{G.U'}.$$

2. The run property "Starting from the initial state, the initiator terminates":

$$N \models \mathsf{A.i} \wedge \mathsf{E.U'} \longmapsto \mathsf{C.i}.$$

The properties can be proved with the help of three equations:

$$\mathsf{A} + \mathsf{B} + \mathsf{C} = \mathsf{i},$$

$$\mathsf{E} + pr_1(\mathsf{F}) + \mathsf{G} = \mathsf{U'} \text{ and}$$

$$\mathsf{N}(\mathsf{A}) + \overline{\mathsf{N}}(\mathsf{C}) + \mathsf{D} + \mathsf{N}(\mathsf{E}) + \mathsf{F} + \overline{\mathsf{F}} + \overline{\mathsf{N}}(\mathsf{G}) = \mathsf{N}(\mathsf{U}).$$

In the second equation, pr_1 denotes the projection onto the first component. The validity in N can be easily asserted by means of place invariants.

22.3 Synchronization in Acyclic Networks

Network algorithms often operate in cycles: each node alternates between the states "active" and "passive" and starts each successive cycle with "active." We describe an algorithm that synchronizes the nodes as tightly as possible: simultaneously active nodes are in the same cycle. Initially, each node is active in the 0th cycle.

As in the Echo Algorithm, each node u has a pivot neighbor v in each cycle. Here, however, u initially does not expect a message from v, but from each of the *other* neighbors. As an example, Fig. 22.7 shows the second cycle of the node γ in the network N_2 in Fig. 22.1, with δ as its pivot neighbor. The transition a is enabled with the token set $\mathsf{N}(\gamma) - (\gamma, \delta) = [(\gamma, \alpha), (\gamma, \beta), (\gamma, \delta)] - (\gamma, \delta) = [(\gamma, \alpha), (\gamma, \beta)]$. The occurrence of a sends the message (γ, δ) from δ to its current pivot neighbor γ. In the state $\mathsf{passive}$, γ waits for a message (γ, δ) from δ and sends messages to its two other neighbors α and β.

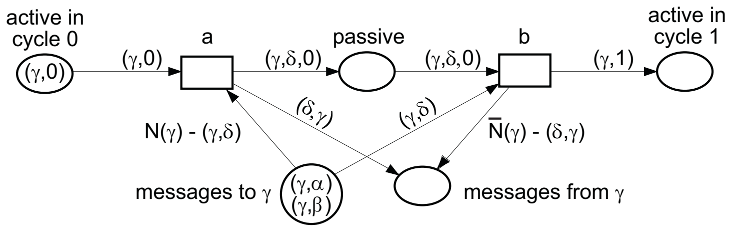


Figure 22.7: The cycle of the node γ with the pivot neighbor δ

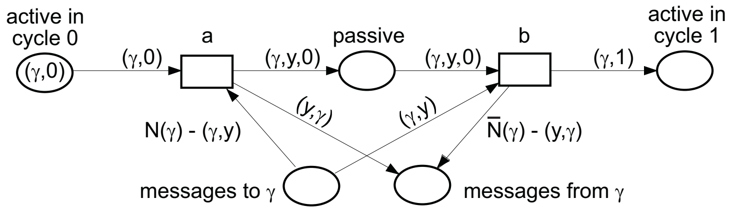


Figure 22.8: First cycle of the node γ with an arbitrary pivot neighbor

To express that, in principle, each neighbor of γ can play the role of the pivot neighbor, we replace the node δ in Fig. 22.7 with the variable y (Fig. 22.8).

Each node behaves according to the pattern shown for the node γ in Fig. 22.8. Analogously to the step from Fig. 22.4 to Fig. 22.5 for the Echo Algorithm, Fig. 22.8 is now the basis for Fig. 22.9: in the arc labelings, the node γ is replaced with the variable x and the constant cycle number 0 with the variable i. In the place "active in cycle i," the node γ with the cycle number 0 is replaced with the set of nodes U of an arbitrary network. Thereby, each node in U has the cycle number 0.[2]

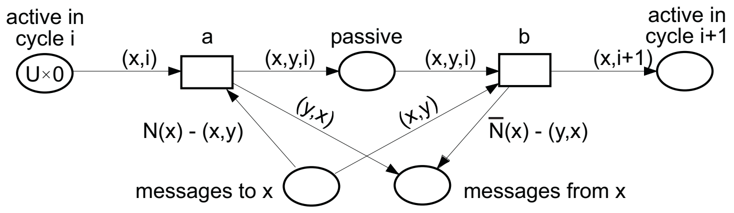


Figure 22.9: ith cycle of an agent

[2]In analogy to the Cartesian product, let $[a_1, \ldots, a_n] \times [b] =_{def} [(a_1, b), \ldots, (a_n, b)]$.

In the place "active in cycle i," a node u with only a single neighbor v enables the transition a in the mode x=u without any pending messages to u. Thus, those nodes start a new cycle. Our algorithm is correct for *acyclic*, finite networks. Each such network has at least two nodes with only one neighbor each.

We develop Fig. 22.10 from Fig. 22.9 by

- combining the places "active in cycle i" and "active in cycle i+1" into a single place A,
- combining the sent, but not yet received messages into a single place B,
- renaming the place "passive" to C.

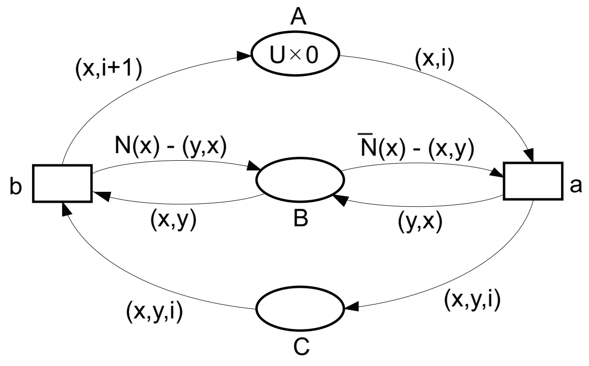


Figure 22.10: Synchronization in acyclic networks

Technically, a cycle of all nodes together forms a scenario (cf. Chap. 5) for markings M with $pr_1(M(\mathsf{A})) = \mathsf{U}$. Three properties characterize the correctness of the algorithm in Fig. 22.10:

1. the state property "two active nodes are in the same cycle":
$$\mathsf{A}.(\mathsf{u}, n) \wedge \mathsf{A}.(\mathsf{v}, m) \longrightarrow n = m,$$
2. the state property "the cycle number of a passive and an active node differs by at most 1":
$$\mathsf{A}.(\mathsf{u}, n) \wedge \mathsf{C}.(\mathsf{v}, m) \longrightarrow |n - m| \leq 1,$$

3. the run property "if the agent u has reached its ith cycle, it will eventually reach the $(i+1)$th cycle as well":

$$\mathsf{A}.(\mathsf{u}, i) \longmapsto \mathsf{A}.(\mathsf{u}, i+1).$$

The properties can be proved with the help of three equations:

1. Each node is either active or passive:

$$pr_1(\mathsf{A}) + pr_1(\mathsf{C}) = \mathsf{U},$$

   where pr_1 maps a tuple to its first component, i.e. $pr_1((u,i)) = u$ and $pr_1([(u_1,i_1),\ldots,(u_n,i_n)]) = [u_1,\ldots,u_n]$.

2. From and to passive components, messages are underway:

$$\mathsf{B} + \overline{\mathsf{B}} + \mathsf{N}(pr_1(\mathsf{C})) + \overline{\mathsf{N}}(pr_1(\mathsf{C})) = 2(pr_{1,2}(\mathsf{C}) + pr_{2,1}(\mathsf{C})),$$

   where $pr_{1,2}(a,b,c) = pr_{2,1}(b,a,c) = (a,b)$ and $(\overline{a,b}) = (b,a)$.

3. The third equation covers all the places and includes the cycle number. Furthermore, it uses four functions that map each agent and each number to a multiset of node pairs:

$$\alpha(u,n) =_{\text{def}} 2n \cdot N(u),$$

$$\overline{\alpha}(u,n) =_{\text{def}} 2n \cdot \overline{N}(u),$$

$$\beta(u,n) =_{\text{def}} 2(n+1) \cdot N(u),$$

$$\overline{\beta}(u,n) =_{\text{def}} 2(n+1) \cdot \overline{N}(u).$$

   This yields the following equation:

$$\alpha(\mathsf{A}) + \overline{\mathsf{B}} + \beta(pr_{1,3}(\mathsf{C})) = \overline{\alpha}(\mathsf{A}) + \mathsf{B} + \overline{\beta}(pr_{1,3}(\mathsf{C})).$$

As before with the Echo Algorithm, these equations are derived from place invariants.

22.4 Consensus in the Network

The nodes of a network often want to reach a mutual agreement. To do this, each node can send its neighbors a request and respond to requests from its neighbors. The algorithm terminates when all nodes have reached an agreement. To this end, each message contains not only the recipient and the sender, but also a "tag" with "?" or "!" that indicates whether the message is a request of the sender to the recipient or a response of the sender to a previous request of the recipient. Thus, a message is of the form

$$(\mathsf{recipient}, \mathsf{sender}, \mathsf{?}) \text{ or } (\mathsf{recipient}, \mathsf{sender}, \mathsf{!}).$$

Let Q(u) denote the set of all requests (v,u,?) of a node u *to* its neighbors v, and let R(u) denote the set of all responses (u,v,!) from its neighbors v to u.

For the node α of the network N_1 in Fig. 22.1, for instance, the following holds:

$$\mathsf{Q}(\alpha) = [(\beta, \alpha, ?), (\gamma, \alpha, ?)] \text{ and} \tag{3}$$

$$\mathsf{R}(\alpha) = [(\alpha, \beta, !), (\alpha, \gamma, !)]. \tag{4}$$

Figure 22.11 shows how α assents to current agreements or sends new requests. If all neighbors of α have responded to its requests (place "α is informed") and α is still negotiating, there exist two possibilities: either α agrees (transition a) or α sends new requests Q(α) (see (3)) via the place "requests from α" to its neighbors β and γ (transition c) and then waits until both neighbors have responded with tokens R(α) (see (4)) in the place "responses to α."

Figure 22.12 shows how α responds to requests from its neighbors: if α has already agreed to the present agreement and then receives a new request, it will again become ready to negotiate (transition b). If α is still negotiating anyway, it will stay in this state (transition e).

From Figs. 22.11 and 22.12, we derive the Consensus Algorithm shown in Fig. 22.13 by

- combining the two figures,

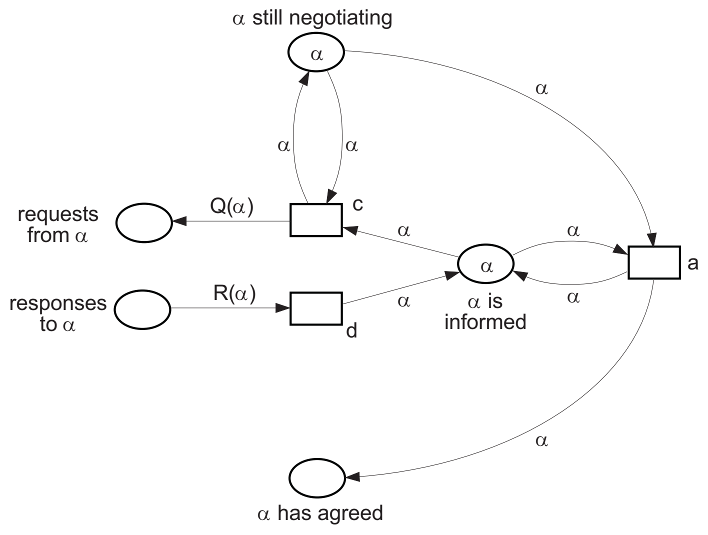


Figure 22.11: Node α agrees or sends new requests and expects responses

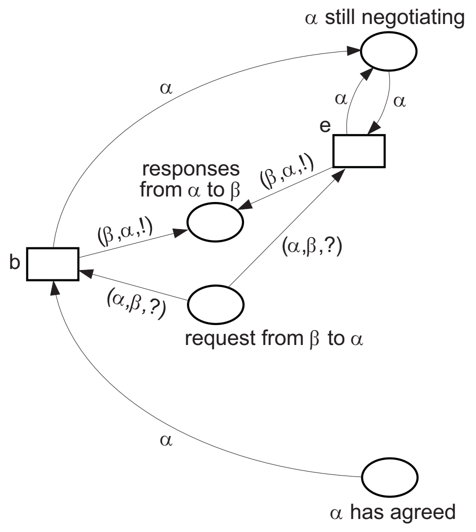


Figure 22.12: Node α receives a request from β and responds to it

- expanding the behavior to include all nodes in U, thereby replacing the constant α with the variable x,
- combining the four places with sent, but not yet received requests and responses into a *single* place D.

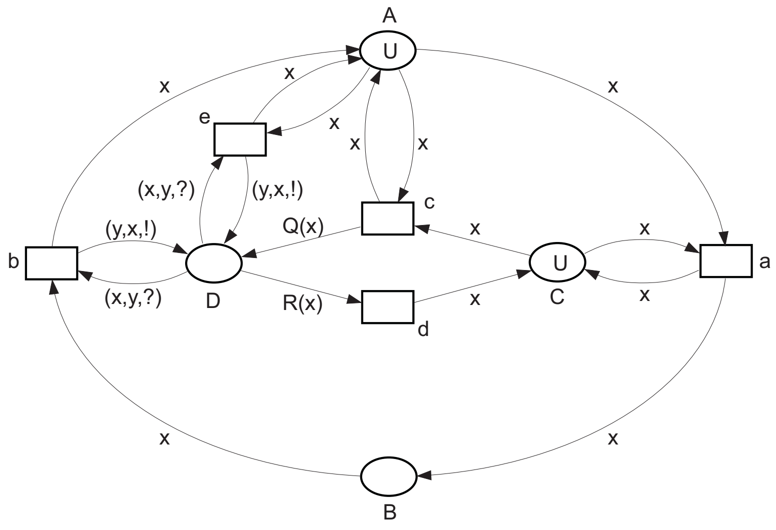


Figure 22.13: The Consensus Algorithm

Three properties characterize the correctness of the algorithm in Fig. 22.13:

1. the state property "if the node u has agreed, then u is informed":
$$\mathsf{B.u} \rightarrow \mathsf{C.u},$$
2. the state property "if all nodes have agreed, then no more messages are underway":
$$\mathsf{B.U} \rightarrow \mathsf{D} = [\,],$$
3. the run property "each node u always becomes informed again":
$$true \longmapsto \mathsf{C.u}.$$

The properties can be proved with the help of two valid equations and one inequality:

1. Each node is still negotiating or has agreed:

$$\mathsf{A} + \mathsf{B} = \mathsf{U}.$$

2. Each node is either informed, or messages from or to it are underway:

$$\mathsf{Q}(\mathsf{C}) + \mathsf{R}(\mathsf{C}) + \mathsf{D} + pr_{2,1,3}(\mathsf{D}) = \mathsf{Q}(\mathsf{U}) + \mathsf{R}(\mathsf{U}),$$

   where $pr_{2,1,3}(a, b, c) =_{\text{def}} (b, a, c)$.

3. Each node is still negotiating or is informed:

$$\mathsf{A} + \mathsf{C} \geq \mathsf{U}.$$

As with the previous two algorithms, the two equations are directly derived from place invariants. The inequality follows from an initially marked trap.

Exercises

1. For the Echo Algorithm in Fig. 22.6: construct its matrix and the initial marking in vector notation and find three place invariants that prove the equations at the end of Section 22.2.

2. For the Consensus Algorithm in Fig. 22.13:

   (a) Construct its matrix and the initial marking in vector notation and find two place invariants with the carriers $\{\mathsf{A}, \mathsf{B}\}$ and $\{\mathsf{C}, \mathsf{D}\}$, respectively.

   (b) Show that $\{\mathsf{A},\mathsf{C}\}$ is a trap.

   *(c) From the place invariants and the trap, derive the three properties discussed in Section 22.4.

Further Reading

A network algorithm describes the behavior of a whole class of networks. With a (generic) system net as in Chap. 2, it is possible to describe the behavior of, at most, a single network. Therefore, we have used *net schemata* here. Further network algorithms modeled as Petri net schemata and further distributed algorithms in general are described in [64].

Part IV

Conclusion

Closing Remarks

Cha

23.1 A Brief History of Petri Nets

In the 1960s, Carl Adam Petri's proposals for the modeling of discrete, asynchronous systems were too advanced for practical application and the "wrong" topic for the theory at that time. Accordingly, the responses were limited at first, but at least the MAC project at MIT picked up Petri nets in the late 1960s. In the 1970s, Petri nets were often used (inappropriately) to characterize formal languages. The reachability problem was long regarded as the central challenge in this area. In the early 1980s, "colored" tokens significantly increased the expressive power of Petri nets, and modeling tools enhanced their applicability in larger projects. At the same time, Petri nets entered into a fierce competition with other modeling techniques. The general interest in modeling techniques, especially graphical ones, which had been growing since the 1990s, eventually established Petri nets as an important contribution to computer science. Since 1979, an annual conference, summer schools, workshops and anthologies on specific topics have come on the scene. The number of publications on Petri nets is in the five-digit range. Historical overviews can be found in [69] and [60].

23.2 Properties of the Elementary Formalisms of Petri Nets

Every technique for the modeling of discrete, dynamic systems describes *states* and *steps*. Apart from this, their elementary formalisms can differ fundamentally. State charts, for instance, use hierarchically and parallelly composed state components. Process algebras emphasize the binary synchronization of actions in steps and the inductive buildup of models. Petri nets use multisets as state components. Steps are *linear*, *local* and *reversible*. For Petri net schemata (Chapter 22), they are also *universal*. The following explains what this means in detail, what advantages it brings and which other modeling techniques partially work according to similar principles.

Multisets are an appropriate data structure: a series of modeling techniques for discrete systems with asynchronous components use multisets as their primary data structure [14], among them *LINDA*, the *Chemical Abstract Machine* and *DNA Computing*. Figuratively speaking, there exists a "pot" and processes can add elements to and remove them from it. This pot metaphor exists in many variants. A Petri net models such "pots" as places.

The reason for choosing multisets instead of regular sets is obvious: to put an element into a "pot," it would otherwise be necessary to search the whole pot for another occurrence of the element. This would prevent the mutually independent ("asynchronous") access of multiple processes to such a pot.

Steps are "linear": Let $M_1 \xrightarrow{t} M_2$ be a step. Then $M_1 + M \xrightarrow{t} M_2 + M$ is also a step for each marking M. In particular, with the marking $n \cdot M_1$, the transition t can occur at least n times: steps behave in a *linear* fashion. All modeling techniques that use multisets as their elementary data structure also use linear steps. The linearity of steps is also essential for the calculi of place and transition invariants.

Transitions have local causes and effects. The preconditions for the occurrence of a transition t are entirely located in ${}^\bullet t$. Its effect pertains to ${}^\bullet t$ and $t^\bullet$. This locality of t immensely simplifies the intuitive understanding.

Other modeling techniques, for instance, automata and process algebras, use *actions* to model elementary steps. In such a model, one and the same action may be noted down multiple times and in different contexts. In the automaton in Fig. 7.1, for instance, the action "switch light off" occurs in the labelings of *two* arcs. (In the corresponding Petri net in Fig. 7.2, there exists only a single transition "switch light off.") To fully understand an action t in an automaton or process expression, it is necessary to take into account each and every occurrence of t: the description of the meaning of a *single* action is scattered across the entire model. This problem is even more evident in structured or hierarchical modeling techniques, such as state charts or message sequence charts (MSC).

For large systems in particular, the locality of transitions is of great value. Exponentially growing state spaces stand in contrast to transitions, whose pre- and post-sets generally do not grow significantly, even in larger systems. Thus, the matrix $\underline{N}$ of a large system N is sparse. Therefore, finding place and transition invariants is often not much more complex than in small systems.

Locality offers additional structural arguments. This is the reason that traps, cotraps and free-choice structures are possible in Petri nets and that they can be exploited in their analysis. The occurrences of two locally independent transitions t and u (i.e. ${}^\bullet t \cap {}^\bullet u = \emptyset$) are orderless in a distributed run. Locality is thus the basis for distributed runs and thus for scenarios and ultimately also for the *stubborn set* method of temporal logic, which increases efficiency.

Steps are "reversible": in each modeling technique, steps of the form $M \xrightarrow{t} M'$ are formed such that M' can be derived from M and t. A step is *reversible* if also M can be derived from M' and t. A step of a Petri net is always reversible: $\underline{M} = \underline{M'} - \underline{t}$. In contrast to this, value assignments are generally not reversible. An example of this is the value assignment $x := 0$. Intuitively speaking, given information can be returned, but deleted information cannot be brought back. The reversibility of steps is ultimately the reason that place and transition invariants are so powerful.

Petri net schemata are uninterpreted: the network algorithms in Chapter 22 are formulated as *Petri net schemata*. They contain the symbols i, u, N and $\overline{N}$, which first have to be interpreted. Only then is a Petri net generated. Considering *all* interpretations of symbols to be sets or functions is well-known from predicate logic. This technique is extremely powerful and flexible. In computer science, this idea is used for describing static structures in algebraic specifications and dynamic steps, especially in abstract state machines (ASM).

23.3 Speculative Questions

In his presentation at the 26th annual meeting of the International Conference on Applications and Theory of Petri Nets in Miami in June 2005, Carl Adam Petri praised the diversity of theory construction and areas of application that had been achieved since his dissertation. At the same time, he urged people to pose more fundamental questions.

In [59], Petri himself formulated a few such proposals and, in his presentations over recent years, emphasized physics as the basis for his motivation. His concepts always had the goal to formulate information processing independently from the current technological standards. In fact, it is likely that hitherto unknown or unused atomic or even biological effects will be employed for future calculations or conceived of as data processing units.

How can Petri nets contribute to the explanation or use of such effects? Let us first consider an analogy from chemistry and physics: Among the fundamental insights of these areas are laws of conservation. If a system does not exchange matter or energy with its environment, both can be transformed in various ways in the system's interior. The sum total, however, stays the same. Nothing is lost and nothing is added. The – not very intuitive – terms of matter and energy are defined in exactly such a way that these conservation laws hold.

Does there exist a corresponding term for a science of information transformation? What is conserved in the interior of a dynamic system that does not exchange any information with its environment? Currently used terms for "information" are obviously not very helpful here.

Petri nets may help to coin an appropriate term: their elementary dynamic concept is *reversible*. If the marking M' and the transition t are known for a step $M \xrightarrow{t} M'$, then the initial marking M can be back-calculated. For a classic value assignment, for instance, $x := 1$, the original value of x cannot be back-calculated. Reversibility in dynamic processes supports fundamental invariance from which, already in 1967, Petri constructed a reversible propositional logic [57].

The insight that the partial independence of components decisively structures a system is equally important: the causal partial order of events is in itself objective and does not need an "observer," who forces events into a "temporal" order.

It is currently not foreseeable how far-reaching these and others of Petri's proposals are.

With the currently available concepts of Petri nets, we are able to build better systems. At the same time, we should be prepared for surprises in the future.

23.4 Petri Nets in Software Engineering

Because software and computer-integrated systems are more and more often abstractly modeled before their actual implementation, the role of Petri nets grows as they successfully combine intuition and precision during the planning and configuration of IT products. The techniques introduced in this book are sufficient to represent causal interrelations in the control flow and simple structures in the data space. However, not all important aspects of complex software models can be expressed in this way. There are two groups of aspects that have not been covered here: the first group is temporal and stochastic interrelations of actions, which need to be expressible. Petri nets were extended very early to allow for these aspects to be modeled. Thus, a series of variants were created and many software tools support these aspects. [4], [41] and [7] further address these questions.

The second group of aspects that were not covered in this book relates to the appropriate handling of *large* system models. Such models have to be systematically *composed* from smaller ones. Basic ideas for this have been discussed in Chapter 8. Equally important is the *refinement* of components. In the simplest case, a transition t is replaced with an interface net whose interface consists of the places in ${}^\bullet t \cup t^\bullet$. An overview of the different variants of refinement and composition can be found in [30].

Large systems are nowadays designed in a systematic process, aided by software tools. There is a series of such design procedures that are based on Petri nets. The most universal and most common approach is based on “colored nets” [38], a special variant of system nets.

Numerous other proposals pick up the idea of incorporating the concept of object orientation into Petri nets. [51] provides an overview.

With activity diagrams, the currently dominant “Universal Modeling Language” (UML) adopted some ideas of Petri nets. A more fundamental connection between Petri nets and UML is described in [74].

23.5 Reference to Other System Models and Analysis Techniques

Next to the usual automaton models, particularly finite automata with their graph representation, there are also some models that, like Petri nets, propose some elementary formalisms and thus specify a class of systems. The formalisms of Petri nets are described in this book and summarized in Section 23.3.

Process algebras are noted for their inductive design and their focus on pairwise "handshake synchronization" of actions. [9] tries to combine the advantages of both system models.

Abstract State Machines (ASM) pursue the idea of leaving the interpretations of all data and function symbols in an algorithm's specification completely open. This results in a concept of algorithms that is more general than usual [32]. The concept of Petri net schemata in Chapter 22 is based on this idea.

For the analysis methods presented in Part II, the question of the complexity of their algorithmic solutions arises. This text addressed these issues only rarely. There are numerous studies in the literature that sometimes bring up very fundamental questions of complexity theory. [22] provides an overview.

23.6 Other Introductory Texts

The most recent comprehensive introduction to Petri nets was written by a team of authors after the *Third Advanced Course on Petri Nets* in 1998 [70]. Compared to that text, this one is shorter and focuses on fundamental terms, accurate analysis techniques and case studies. At the same time, it contains a few new items:

- As far as possible, elementary and generic systems nets are not treated separately.
- State properties written as equations and inequalities are separated from the question of how to prove their correctness.
- Scenario-based runs and run properties are emphasized.
- Hot and cold transitions are distinguished.
- In some instances, new terminology is used.

In particular, the term "system net" is new for an initially marked net structure. The prefix "elementary" denotes the special case of "black" tokens. The term "Petri net" is used to denote the entire field of study. All this is meant to emphasize the advantages of Petri nets and to intuitively, conceptually and terminologically simplify their comprehensibility.

The literature on Petri nets is vast and steadily growing. The online platform `http://www.informatik.uni-hamburg.de/TGI/GI-Fachgruppe0.0.1/` maintained by the University of Hamburg, Germany is an excellent choice for an introduction.

Formal Framework

Appe

Here we present a concise compilation of the formal framework that is used throughout this book. For easy navigation and cross-reference, the relevant terms are highlighted here and also in the margin of each chapter.

Components of a Net (☞ Sect. 2.2)

Definition 1. *Let P and T be sets and let $F \subseteq (P \times T) \cup (T \times P)$.*

(i) $N =_{def} (P, T, F)$ *is a* net structure. P, T *and* F *contain the* places, transitions *and* arcs *of* N, *respectively.*

(ii) *For* $x \in P \cup T$, ${}^{\bullet}x =_{def} \{y|yFx\}$ *is the* pre-set *and* $x^{\bullet} =_{def} \{y|xFy\}$ *is the* post-set *of* x.

(iii) $x, y \in P \cup T$ *form a* loop *iff* $x \in {}^{\bullet}y$ *and* $y \in {}^{\bullet}x$.

(iv) N *is* strongly connected *iff* $aF^{+}a$ *for each* $a \in P \cup T$.

(v) *N is* acyclic *iff* $aF^{+}a$ *for no* $a \in P \cup T$.

Definition 2. *Let* $N = (P, T, F)$ *be a net structure, let* L *be a set and let* $\ell : P \cup T \to L$*. Then* ℓ *is a* labeling of N *(*N *is* ℓ-labeled*).*

Multisets (☞ Sect. 2.3)

Definition 3.

(i) *For a set* U*, a mapping* $a : U \to \mathbb{N}$ *is a* multiset over U. $\mathcal{M}(U)$ *denotes the set of all multisets over* U*. We write* $\mathcal{M}$ *for* $\mathcal{M}(U)$ *if* U *is irrelevant or obvious from context.*

(ii) $a \in \mathcal{M}$ *is* finite *iff* $a(u) \neq 0$ *for only finitely many* $u \in U$.

(iii) $[\,] \in \mathcal{M}$ *denotes the* empty multiset, *with* $[\,](u) = 0$ *for all* $u \in U$.

(iv) *For* $a, b \in \mathcal{M}$*, the* sum $a + b \in \mathcal{M}$ *of* a *and* b *is defined for each* $u \in U$ *by* $(a + b)(u) =_{def} a(u) + b(u)$.

(v) *For* $a, b \in \mathcal{M}$, a *is* smaller or equal to b, *written* $a \leq b$*, iff for each* $u \in U$, $a(u) \leq b(u)$*. The relation* $\leq$ *is a* partial order *on* $\mathcal{M}$.

(vi) *For* $a, b \in \mathcal{M}$ *with* $b \leq a$*, the* subtraction $a - b \in \mathcal{M}$ *of* b *from* a *is defined for each* $u \in U$ *by* $(a - b)(u) =_{def} a(u) - b(u)$.

Notations A.1. *A finite multiset* a *can be written* $[a_1, \ldots, a_n]$*, where for each* $u \in U$ *there exist* $a(u)$ *indices* $1 \leq i \leq n$ *with* $a_i = u$.

Expressions (☞ Sect. 2.6)

As usual we assume constant and function symbols together with a fixed *interpretation*. This includes a set U such that each constant symbol is interpreted as an element of U and each function symbol as a function over U. The symbols can be composed to form (variable-free) *expressions*. The interpretation of symbols yields canonically an interpretation of each expression as an element of U.

Two expressions d and e may thus be interpreted as the same element of U, in which case d and e are *equivalent*, written $d \sim e$. As usual, we frequently confuse symbols and variable-free expressions with their interpretation.

Expressions may also include *variables*. With a set X including all variables occurring in an expression e, a mapping $\beta : X \to U$ *evaluates* X in U. This yields an interpretation $\beta(e) \in U$: replace each occurrence of each variable x in e by its evaluation $\beta(x)$. This returns a variable-free expression $\beta(e)$, which is interpreted as described above.

An expression e is a *condition* if $\beta(e) \in \{\text{true}, \text{false}\}$ for each evaluation β. For finite multisets, β generalizes to $\beta([e_1, \ldots, e_n]) =_{def} [\beta(e_1), \ldots, \beta(e_n)]$.

Let exp denote the set of all expressions and cond the set of all conditions as assumed in the given context.

Markings, Modes, Steps (☞ Sects. 2.4–2.6)

Definition 4.

(i) *A mapping $M : P \to \mathcal{M}$ is a* marking of *N.*

(ii) *Two markings M and M' are* ordered *(written $M \leq M'$) iff $M(p) \leq M'(p)$ for each $p \in P$.*

Definition 5. *Let $N = (P, T, F)$ be a net structure, let $t \in T$.*

(i) *A condition $\hat{t} \in \mathrm{cond}$ is a transition condition.*

(ii) *For $p \in {}^\bullet t$ and $q \in t^\bullet$, sets $\overline{pt}$, $\overline{tq} \in \mathcal{M}(\mathrm{exp})$ are arc labelings of t. For technical reasons, with $(a, b) \notin F$ let $\overline{ab} =_{def} [\,]$.*

(iii) *Let X be the set of variables occurring in given transition conditions and arc labelings of t. Then a mapping $\beta : X \to U$ is a mode of t. (t, β) is an* event *of N.*

(iv) *Let β be a mode of t. Then the markings ${}^\bullet[t, \beta]$ and $[t, \beta]^\bullet$ are defined for each $p \in P$ by ${}^\bullet[t, \beta](p) =_{def} \beta(\overline{pt})$ and $[t, \beta]^\bullet(p) =_{def} \beta(\overline{tp})$.*

(v) *Let $\hat{t}$ be a condition, let β be a mode of t and let M be a marking. Then M enables t in the mode β (or M* enables the event *(t, β)) iff $\beta(\hat{t}) = \text{true}$ and ${}^\bullet[t, \beta] \leq M$.*

(vi) *Let M enable (t, β), and let $M' = M - {}^\bullet[t, \beta] + [t, \beta]^\bullet$. Then (M, t, β, M') is a step of t in mode β or a* step *of (t, β), written $M \xrightarrow{(t,\beta)} M'$.*

System Nets (☞ Sect. 2.7)

Definition 6. *Let $S = (P, T, F)$ be a net structure, let U be a set, let M_0 be a marking of P over U, let $\tau : T \to \mathrm{cond}$, let $\ell : F \to \mathcal{M}(\mathrm{exp})$, let $C \subseteq T$.*

Then $N =_{def} (S, M_0, \ell, \tau, C)$ *is a system net over* U *with* M_0, ℓ, τ *and* C *its* initial marking, arc labeling, transition condition *and* set of cold transitions*, respectively.*

General Assumptions

In the following we generally assume a set U, called the *universe*, and a system net $N = (S, M_0, \ell, \tau, C)$ over U, with $S = (P, T, F)$. Furthermore, for each $p \in P$, $t \in T$ and $(a, b) \in F$ we assume $M_0(p)$, $\tau(t)$ and $\ell(ab)$ to be finite.

(i) We usually write $\hat{t}$ for $\tau(t)$ and $\overline{ab}$ for $\ell(a, b)$.

(ii) A *marking*, a *place*, a *transition* and an *arc of* N is sloppy for a marking, a place, a transition and an arc of S.

(iii) A transition t is *hot* iff it is not cold.

Reachable Markings, Marking Graph (☞ Sect. 2.8, 2.9)

Definition 7. *The set* R *of reachable markings of* N *is inductively defined by*

(i) $M_0 \in R$;

(ii) If $M \in R$ *and* $M \xrightarrow{u} M'$ *is a step of* N*, then* $M' \in R$.

Definition 8.

(i) A step $M \xrightarrow{u} M'$ *is* reachable *iff* M *is reachable.*

(ii) A marking M' *is* reachable from a marking M *iff there exists a sequence* $M_{i-1} \xrightarrow{t_i} M_i$ $(i = 1, \ldots, n)$ *of steps with* $M_0 = M$ *and* $M_n = M'$.

Definition 9. *Let* V *and* L *be sets, let* $v \in V$ *and let* $E \subseteq V \times L \times V$*. Then* (V, E, v) *is a* (directed, initialized, arc-labeled) graph*.* V *is the set of* vertices *and* $(e_1, \ell, e_2) \in E$ *is an* ℓ-edge*, with* ℓ *its* label.

Definition 10. *Let* R *and* E *be the sets of reachable markings and steps, respectively, of* N*. Then the graph* (R, E, M_0) *is the marking graph of* N*, written* $mg(N)$.

Definition 11. *A marking* M *of* N *is final iff* M *enables only cold transitions.*

Elementary System Nets (☞ Sects. 3.1, 3.5)

Definition 12. N *is elementary iff*

(i) $\{\bullet\}$ *is the underlying universe* U,

(ii) $\hat{t} = \mathrm{true}$ *for each* $t \in T$,

(iii) $\overline{ab} = [\bullet]$ *for each arc* $(a, b) \in F$.

Notations A.2. *Let* N *be elementary.*

(i) Each mapping $a : \{\bullet\} \to \mathbb{N}$ *can be identified with* $a(\bullet) \in \mathbb{N}$*. Hence each marking* M *of* N *can be written as* $M : P \to \mathbb{N}$.

(ii) As arc labels are all alike, they are skipped in graphical representations. This likewise applies to transition conditions.

(iii) *As no arc labeling contains a variable, each step* $M \xrightarrow{t,\beta} M'$ *can be written as* $M \xrightarrow{t} M'$.

Observation 1. *For an elementary* N, ${}^\bullet[t,\beta](p) = [t,\beta]^\bullet(q) = [\bullet]$ *iff* $p \in {}^\bullet t$ *and* $q \in t^\bullet$.

Definition 13. *Let* N *be elementary.* N *is* 1*-bounded iff for each reachable marking* M *and each* $p \in P$ *holds:* $M(p) \leq 1$.

Sequential Runs (☞ Sect. 4.1)

Definition 14.

(i) *A finite sequence of steps* $M_0 \xrightarrow{u_1} M_1 \xrightarrow{u_2} \dots \xrightarrow{u_n} M_n$ *of* N *is a* finite sequential run of N *iff* M_n *is a final marking of* N.

(ii) *An infinite sequence of steps* $M_0 \xrightarrow{u_1} M_1 \xrightarrow{u_2} \dots$ *of* N *is* incomplete *iff there exists an index* n *and a hot transition* t *such that for each* $i \geq n$ *holds:*M_i *enables* t *and* ${}^\bullet t \cap {}^\bullet u_i = \emptyset$.

(iii) *An infinite sequence of steps is an* infinite sequential run of N *iff it is not incomplete.*

Actions (☞ Sect. 4.3)

Definition 15. *An action is a labeled net* $A = (Q, \{v\}, G)$ *with* ${}^\bullet v \cap v^\bullet = \emptyset$ *and* ${}^\bullet v \cup v^\bullet = Q$.

Definition 16. *Let* Q *be a set labeled by some* $\ell : Q \to P \times U$. *Then the marking* M_Q *of* P *is defined for each* $p \in P$ *by* $M_Q(p) = [u_1, \dots, u_n]$ *iff* $q_1, \dots, q_n$ *are the elements of* Q *with* $\ell(q_i) = (p, u_i)$ *for* $i = 1, \dots, n$.

Definition 17. *An* ℓ*-labeled action* A *with transition* v represents the event (t,β) *of* N *iff* $\ell(v) = (t,\beta)$, $M_{{}^\bullet v} = {}^\bullet[t,\beta]$ *and* $M_{v^\bullet} = [t,\beta]^\bullet$.

Distributed Runs (☞ Sect. 4.4)

Definition 18. *A net* $K = (Q,V,G)$ *is a causal net iff*

(i) *for each* $q \in Q$, $| {}^\bullet q | \leq 1$ *and* $| q^\bullet | \leq 1$*;*

(ii) *The transitive closure* G^+ *of* G *is irreflexive (i.e.,* G^+ *is a strict partial order);*

(iii) *For each* $x \in Q \cup V$, $\{y \mid yG^+x\}$ *is finite.*

Definition 19. *Let* $K = (Q,V,G)$ *be a causal net.*

(i) ${}^\circ K =_{def} \{q \in Q \mid {}^\bullet q = \emptyset\}$,

(ii) $K^\circ =_{def} \{q \in Q \mid q^\bullet = \emptyset\}$.

Observation 2. *If* $V = \emptyset$ *then* $G = \emptyset$ *and* ${}^\circ K = K^\circ = Q$.

Definition 20. *An* ℓ*-labeled causal net* K *is a distributed run of* N *iff* $M_{{}^\circ K} = M_0$ *and each action of* K *represents an event of* N. K *is complete iff the marking* M_{K° *enables no hot transition.*

Definition 21. *For a causal net* $K = (Q,V,G)$, *the relation* G^+ *is the (strict, partial) causal order on* K.

Composition of Distributed Runs (☞ Sect. 4.8)

Definition 22. *For* $i = 1,2$ *let* $K_i = (Q_i, V_i, G_i)$ *be occurrence nets, labeled with*

ℓ_i. Let $(Q_1 \cup V_1) \cap (Q_2 \cup V_2) = K_1^\circ = {}^\circ K_2$ and for each $p \in K_1^\circ$ let $\ell_1(p) = \ell_2(p)$. Then the occurrence net $K =_{def} (Q_1 \cup Q_2, V_1 \cup V_2, G_1 \cup G_2)$ labeled with ℓ, with $\ell(x) = \ell_i(x)$ is the composition of K_1 and K_2, written $K_1 \circ K_2$.

Scenarios (☞ Sect. 5.1)

Definition 23. *A distributed run K is a scenario of N iff $M_{\circ K} = M_{K^\circ}$.*

More generally:

Definition 24. *A distributed run K is a scenario for a marking M iff $M \leq M_{\circ K}$ and $M \leq M_{K^\circ}$.*

Definition 25. *N is scenario based iff there exists a finite set A of scenarios and a finite set B of finite distributed runs such that*

(i) each finite concurrent run of N is shaped $S_1 \cdots S_n$ with $S_1, S_n \in B$ and $S_2, \ldots, S_{n-1} \in A$,

(ii) each infinite concurrent run of N is shaped $S_1 S_2 \cdots$ with $S_1 \in B$ and $S_2, S_3, \ldots \in A$.

Notice that S_1 and S_n may be empty concurrent runs, i.e., including no transitions.

Place Capacities (☞ Sect. 6.1)

Definition 26. *Let N be elementary, let M be a marking of N, and let $n \in \mathbb{N}$.*

(i) M is n-bounded iff $M(p) \leq n$ for all $p \in P$.

(ii) The set $R^{(n)}$ of n-reachable markings of N is inductively defined by

- *$M_0 \in R^{(n)}$ if M_0 is n-bounded;*
- *If $M \in R^{(n)}$, $M \xrightarrow{t} M'$ is a step and M' is n-bounded, then $M' \in R^{(n)}$.*

Observation 3. *There may exist reachable n-bounded markings that are not n-reachable.*

Definition 27.

(i) Let $p \in P$ and assume $M_0(p) \leq n$. Let $\widetilde{p}$ be a fresh place with ${}^\bullet\widetilde{p} = p^\bullet$, $\widetilde{p}^\bullet = {}^\bullet p$ and $M_0(\widetilde{p}) = n - M_0(p)$. Then $\widetilde{p}$ is the n-complement of p.

(ii) Assume M_0 is n-bounded. Then $N^{(n)}$ is defined as N, extended by the n-complement $\widetilde{p}$ for each place p.

Lemma 1. *Let $n \in \mathbb{N}$ such that M_0 is n-bounded. Let R' be the set of reachable markings of $N^{(n)}$. Then there exists a bijection $f : R^{(n)} \to R'$ such that for all $M, M' \in R^{(n)}$ holds: $M \xrightarrow{t} M'$ is a step on N iff $f(M) \xrightarrow{t} f(M')$ is a step of $N^{(n)}$.*

Arc Weights (☞ Sect. 6.2)

Definition 28. *Let N be elementary, let $w : (P \times T) \cup (T \times P) \to \mathbb{N}$ be a mapping with $w(a,b) = 0$ if $(a,b) \notin F$ and let M, M' be markings of N.*

(i) (N, w) is the w-generalization of N.

(ii) $M \xrightarrow{t} M'$ is a w-step iff for each $p \in P$ $w(p,t) \leq M(p)$ and $M'(p) = M(p) - w(p,t) + w(t,p)$.

Definition 29. *Let N' be an elementary system net. A finite sequential run $r = M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} M_n$ of N is an* *N-reduction* *of a finite sequential run $r' = M'_0 \xrightarrow{t_{1'}} M'_1 \xrightarrow{t_{2'}} \cdots \xrightarrow{t'_{n'}} M'_{n'}$ of N' iff $t_1, \dots, t_k$ is obtained by eliminating from $t'_1, \dots, t'_{k'}$ all transitions that are not in T.*

Lemma 2. *Let (N, w) be a w-generalization of N. Then there exists an elementary system net N' such that the N-reductions of N' coincide with the finite sequential runs of N.*

Regions (☞ Sect. 7.3)

Definition 30. *Let $Z = (V, E, v)$ be a graph, let $R \subseteq V$, let $\pi =_{def} h \xrightarrow{\ell} k \in E$.*

(i) R receives π *iff $h \notin R$ and $k \in R$.* R dispatches π *iff $h \in R$ and $k \notin R$.* R contains π *iff $h, k \in R$.*

(ii) R is a *region of Z* *iff for each edge label ℓ: R receives either each or no ℓ-edge, and R dispatches either each or no ℓ-edge.*

(iii) A region R is *minimal* *iff no proper nonempty subset of R is a region. Let $\mathrm{mreg}(Z)$ denote the set of minimal regions of Z.*

Definition 31. *Let $Z = (V, E, v)$ be a graph, let L be the set of labels occurring at the edges of Z. Then* $\mathrm{sn}(Z) =_{def} (\mathrm{mreg}(Z), L, F, M_0)$ *is the* *(elementary) system net of Z*, *where*

(i) $(\ell, R) \in F$ iff R receives or R contains each ℓ-edge,

(ii) $(R, \ell) \in F$ iff R dispatches or R contains each ℓ-edge,

(iii) $M_0(R) = 1$ if $v \in R$ and $M_0(R) = 0$ if $v \notin R$.

Theorem 1 (Synthesis Theorem)**.** *If the synthesis problem of a state automaton Z can be solved by a 1-bounded elementary system net, then the system net of Z is a solution.*

Nets with Interfaces, Communicating Nets (☞ Sects. 8.1, 8.2)

Definition 32.

(i) Let $I \subseteq P \cup T$. Then (S, I) is an *interface net*. *I is the* *interface* *of (S, I).*

(ii) A set M of interface nets is *associative* *iff no element appears in the interfaces of three or more of the nets in M.*

Definition 33. *For $i = 1, 2$ let $N_i = (S_i, I_i)$ be two interface nets such that $S_i = (P_i, T_i, F_i)$ and $(P_1 \cup T_1) \cap (P_2 \cup T_2) \subseteq I_1 \cap I_2$.*

(i) The interface net $N_1 \oplus N_2 =_{def} ((P_1 \cup P_2, T_1 \cup T_2, F_1 \cup F_2), (I_1 \cup I_2) \setminus (I_1 \cap I_2))$ is the *composition* *of N_1 and N_2.*

(ii) N_1 and N_2 *communicate* *iff for each $x \in I_1 \cap I_2$ holds:*

- *$x \in P_1 \cap P_2$,*
- *either ${}^\bullet x \subseteq T_1$ and $x^\bullet \subseteq T_2$, or ${}^\bullet x \subseteq T_2$ and $x^\bullet \subseteq T_1$.*

Theorem 2 (Composition Theorem for Interface Nets)**.** *For $i = 1, 2, 3$, let N_i be interface nets with the interfaces I_i.*

(a) $N_1 \oplus N_2 = N_2 \oplus N_1$.

(b) *If* $I_1 \cap I_2 \cap I_3 = \emptyset$, *then* $(N_1 \oplus N_2) \oplus N_3 = N_1 \oplus (N_2 \oplus N_3)$.

Decomposition into Open Subnets (☞ Sect. 8.3)

Definition 34. *Let* $P' \subseteq P$, $T' \subseteq T$, $F' = F \cap (P' \times T')$ *and* $S' = (P', T', F')$. *Furthermore, let* $I \subseteq P'$ *such that for each* $p \in I$ ${}^\bullet p \cup {}^\bullet p' \subseteq T'$. *Then* $N' = (S', I)$ *is an* open subnet *of* S. N *is* minimal *iff there exists no open subnet* $S'' = (P'', T'', F'')$ *with* $S'' \neq S'$, $P'' \subseteq P'$, $T'' \subseteq T'$ *and* $F'' \subseteq F'$.

Theorem 3 (Associativity Theorem)**.** *Let* M *be a set of pairwise communicating interface nets. Then* M *is associative.*

Equations and Inequalities (☞ Sects. 9.2, 9.4, 9.7)

Definition 35. *Let* A *be a set, let* $a \in A$, *let* $+ : A \times A \to A$ *be an operation, let* $p_1, \ldots, p_k \in P$, *and let* $f_1, \ldots, f_k : \mathcal{M} \to A$.

(i) $G\colon f_1(p_1) + \cdots + f_k(p_k) = a$ *is an* equation over N.

(ii) For a marking M *of* N, *an equation* G holds in M *(written* $M \models G$*) iff* $f_1(M(p_1)) + \cdots + f_k(M(p_k)) = a$.

(iii) G holds in N *(*G *is* valid *in* N, *written* $N \models G$*) iff* $M \models G$ *for each reachable marking* M *of* N.

(iv) Replacing in (i) and (ii) the equality "=" with "≤" or "≥" yields inequalities.

(v) For each place $p \in P$, $p \geq [\,]$ *is the* canonical inequality *of* p.

Lemma 3. *Each canonical inequality of a place* p *of* N *holds in* N.

Definition 36. *An equation or inequality over* N *is a* state property *of* N.

Theorem 5 (Validity Theorem for Propositional Properties)**.** *For a system net* N *and propositional properties* α *and* β, *the following hold:*

a) $N \models \alpha \wedge \beta$ *iff* $N \models \alpha$ *and* $N \models \beta$.

b) If not $N \models \alpha$, *then not necessarily* $N \models \neg\,\alpha$.

c) If $N \models \alpha$ *or* $N \models \beta$, *then* $N \models \alpha \vee \beta$.

d) If $N \models \alpha$, *then* $N \models \beta \to \alpha$.

e) If $N \models \neg\alpha$, *then* $N \models \alpha \to \beta$.

Traps and Cotraps of Elementary System Nets (☞ Sects. 10.1–10.3)

Definition 37. *Let* $Q \subseteq P$.

(i) ${}^\bullet Q =_{def} \bigcup_{q \in Q} {}^\bullet q$ *and* $Q^\bullet =_{def} \bigcup_{q \in Q} q^\bullet$.

(ii) Q *is a* trap *of* N *iff* ${}^\bullet Q \subseteq Q^\bullet$.

(iii) Q *is a* cotrap *of* N *iff* $Q^\bullet \subseteq {}^\bullet Q$.

(iv) Q *is* initially marked *iff for at least one* $q \in Q$, $M_0(q) \geq [\,]$.

(v) N *has the* trap/cotrap property *iff to each cotrap* R *there exists an initially marked trap* $Q \subseteq R$.

Theorem 6 (Trap Theorem)**.** *Let N be an elementary system net with an initially marked trap $Q = \{q_1, \ldots, q_r\}$. Then the following inequality holds in N:*

$$q_1 + \ldots + q_r \geq 1. \tag{1}$$

Theorem 7 (Cotrap Theorem)**.** *Let N be an elementary system net with an initially unmarked cotrap $Q = \{q_1, \ldots, q_r\}$. Then the following equation holds in N:*

$$q_1 + \ldots + q_r = 0. \tag{2}$$

Theorem 8 (Theorem on Marked Cotraps)**.** *Let N be an elementary system net and let M be a marking of N that marks each cotrap of N. Then M enables at least one transition.*

Theorem 9 (Trap/Cotrap Theorem)**.** *Let N be an elementary system net that has the trap/cotrap property. Then each reachable marking M of N enables at least one transition.*

Vector and Matrix Representations for Markings, Transitions and Elementary System Nets (☞ Sects. 11.1, 11.2)

Definition 38. *W.l.o.g. let $P = \{p_1, \ldots, p_k\}$ and $T = \{t_1, \ldots, t_\ell\}$. For $i = 1, \ldots, k$ and $j = 1, \ldots, \ell$ let*

$$z_{ij} =_{def} \begin{cases} -1, & \text{if } p_i \in {}^\bullet t_j \text{ and } p_i \notin t_j^\bullet \\ +1, & \text{if } p_i \in t_j^\bullet \text{ and } \notin {}^\bullet t_j \\ 0, & \text{otherwise.} \end{cases}$$

Then $\underline{M} = \begin{pmatrix} M(p_1) \\ \vdots \\ M(p_k) \end{pmatrix}$, $\underline{t_i} = \begin{pmatrix} z_{i1} \\ \vdots \\ z_{ik} \end{pmatrix}$ *and* $\underline{N} = \begin{pmatrix} z_{11} & \cdots & z_{\ell 1} \\ \vdots & & \vdots \\ z_{1k} & \cdots & z_{\ell k} \end{pmatrix}$ *are the vector representations of M and t, and the matrix representation of N, respectively.*

Lemma 4. *With component-wise addition and comparison of vectors holds: M enables $\underline{t}$ iff $\underline{t} \leq M$, and $M \xrightarrow{t} M'$ is a step iff M enables t and $\underline{M'} = \underline{M} + \underline{t}$.*

Place Invariants and Equations of Place Invariants of Elementary System Nets (☞ Sects. 11.3, 11.4)

Definition 39. *Let $\vec{0} = (0, \ldots, 0)$ be ℓ-dimensional and let $\underline{n} = (n_1, \ldots, n_k)$ be an integer solution of $x \cdot \underline{N} = \vec{0}$.*

(i) $\underline{n}$ *is a place invariant of N.*

(ii) $n_0 =_{def} \underline{n} \cdot M_0$ *is the constant of $\underline{n}$.*

(iii) $n_1 \cdot p_1 + \cdots + n_k \cdot p_k = n_0$ *is the equation of $\underline{n}$.*

(iv) $\underline{n}$ is positive *iff* $n_1, \ldots, n_k \in \mathbb{N}$.

(v) The carrier *of $\underline{n}$ is the set of all places p_i with $n_i > 0$.*

Theorem 10 (Elementary Place Invariants Theorem)**.** *Let N be an elementary system net with a place invariant $\underline{n}$. Then the equation of $\underline{n}$ holds in N.*

Theorem 11 (Converted Place Invariants Theorem)**.** *Let N be an elementary system net such that for each transition t there exists a* *reachable marking M that enables t. Furthermore, let equation (2) hold in N. Then $\underline{n} = (n_1, \dots, n_k)$ is a place invariant of N with the constant n_0.*

Theorem 12 (Positive Place Invariants Theorem)**.** *A place p with a positive place invariant is bounded.*

Theorem 13 (Invariant Trap/Invariant Cotrap Theorem)**.** *The carrier of a positive place invariant of a system net N is also a trap and a cotrap of N.*

Calculating with Equations and Inequalities (☞ Sects. 12.1, 12.4)

Definition 40. *Let $G_1 : n_1 \cdot p_1 + \dots + n_k \cdot p_k = n_0$ and $G_2 : m_1 \cdot p_1 + \dots + m_k \cdot p_k = m_0$ be equations and let $z \in \mathbb{Z}$. Then*

(i) $G_1 + G_2 =_{def} (n_1 + m_1) \cdot p_1 + \dots + (n_k + m_k) \cdot p_k = n_0 + m_0$ is the sum of G_1 and G_2,

(ii) $z \cdot G_1 =_{def} z \cdot n_1 \cdot p_1 + \dots + z \cdot n_k \cdot p_k = z \cdot n_0$ is the scalar product of G_1 with z.

Theorem 14 (Addition Theorem of Valid Equations and Inequalities)**.** *Let N be an elementary system net. The sum of two equations or inequalities that hold in N, as well as the product of such an equation or inequality with a factor z, again hold in N.*

Definition 41. *A state property G is stable iff for each step $M \xrightarrow{t} M'$ of N holds: If G holds in M then G holds in M'.*

Traps of a System Net (☞ Sect. 13.1)

Definition 42. *For a finite multiset $A = [a_1, \dots, a_k]$ let $|A| =_{def} k$. Let $Q = \{q_1, \dots, q_k\}$ be a trap of N. Then $|q_1| + \dots + |q_k| \geq 1$ is the inequality of Q.*

Sum Expressions (☞ Sects. 13.2, 13.3)

Definition 43. *An expression e is* unary *if at most one variable occurs in e. (This variable may occur several times).*

Definition 44. *The set SE of sum expressions is inductively defined:*

(i) $\underline{0} \in \mathrm{SE}$,

(ii) For $e_1, e_2 \in \mathrm{SE}$, also $(e_1 + e_2) \in \mathrm{SE}$ and $-e_1 \in \mathrm{SE}$,

(iii) If $f(b_1, \dots, b_n)$ is an expression and $e_1, \dots, e_n \in \mathrm{SE}$, then $f(e_1, \dots, e_n) \in \mathrm{SE}$,

(iv) If e_1 is unary then $e_1 \cdot e_2 \in \mathrm{SE}$.

Notations A.3. *A multiset $[a_1, \dots, a_n]$ is represented by the expression $a_1 + \dots + a_n$. The empty multiset $[\,]$ is represented by $\underline{0}$.*

Product and Equivalence of Sum Expressions (☞ Sect. 13.3)

Definition 45. *Let e_1 and e_2 be two sum expressions, let e_1 be unary with variable x. Then the product of e_1 with e_2 is defined as $e_1 \cdot e_2 =_{def} e_1[x \setminus e_2]$ (i.e., each occurrence of x in e_1 is replaced by e_2).*

Definition 46. *The* sum expression equivalence $\sim$ *is defined for sum expressions* e_1, e_2, e_3 *and function symbols* f *as*

- $e_1 + e_2 \sim e_2 + e_1$,
- $(e_1 + e_2) + e_3 \sim e_1 + (e_2 + e_3)$,
- $e_1 + \underline{0} \sim e_1$,
- $e_1 + (-e_1) \sim \underline{0}$,
- $f(\dots, e_1 + e_2, \dots) \sim f(\dots, e_1, \dots) + f(\dots, e_2, \dots)$,
- $f(\dots, -e_1, \dots) \sim -f(\dots, e_1, \dots)$,
- $f(\dots, \underline{0}, \dots) \sim \underline{0}$.

Definition 47. *The* calculation rules *for sum expressions are given by the above sum expression equivalence together with the equivalence as described in Sect. 2.6.*

Matrix and Place Invariants for Generic System Nets (☞ Sects. 13.5–13.8)

Definition 48. *As in Sect. 11.1, w.l.o.g let* $P = \{p_1, \dots, p_k\}$ *and* $T = \{t_1, \dots, t_\ell\}$*. With* $z_{ij} =_{def} \overline{t_j p_i} - \overline{t_i p_j}$*, let* $\underline{t_i}$ *and* $\underline{N}$ *as in Sect. 11.1 and 11.2. Then* $\underline{N}$ *is the* *matrix of* N.

Definition 49. *Let* $b_1, \dots, b_k$ *be unary sum expressions such that for* $j = 1, \dots, k$ *holds:* $b_1 \cdot \underline{N}(1, j) + \dots + b_k \cdot \underline{N}(k, j)$ *can be reduced to* $\underline{0}$ *by means of the calculation rules. Then* $b =_{def} (b_1, \dots, b_k)$ *is a* *place invariant of* N.

Definition 50. *Let* $b = (b_1, \dots, b_k)$ *be a place invariant of* N.

(i) *W.l.o.g for each place* p *assume* $M_0(p)$ *be represented as a sum expression. Then the sum expression* $b_0 =_{def} b_1(M_0(p_1)) + \dots + b_k(M_0(p_k))$ *is the* *constant of* b.

(ii) *The equation* $b_1(p_1) + \dots + b_k(p_k) = b_0$ *is the* *equation of* b.

Theorem 15 (Generic Place Invariant Theorem)**.** *Let* N *be a system net and let* b *be a place invariant of* N*. Then the equation of* b *holds in* N.

Covering Graph (☞ Sects. 14.4, 14.8)

Definition 51. *Let* $c : P \to \mathbb{N} \cup \{\omega\}$ *be a mapping.*

(i) c represents *a marking* M *of* N *iff for all* $p \in P$ *with* $c(p) \in \mathbb{N}$ *holds:* $M(p) = c(p)$.

(ii) c covers N *iff* c *is reachable or represents infinitely many reachable markings* $K_0, K_1, \dots$ *such that for all* $i = 0, 1, 2, \dots$ *and all* p *with* $c(p) = \omega$ *holds:* $M_i(p) \geq i$.

Definition 52. *Let* G *be a finite, initial, directed, arc labeled graph with initial vertex* M_0 *and markings that cover* N *as vertices, such that there exists a sequential run* $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots$ *of* N *iff there exists a path* $n_0 \xrightarrow{t_1} n_1 \xrightarrow{t_2} \dots$ *of* G *such that* n_i *represents* M_i *for* $i = 1, 2, \dots$*. Then,* G *is a* *covering graph of* N.

Theorem 19 (Theorem on Dead Transitions)**.** *Let* H *be a covering graph of an elementary system net* N*. A transition* t *is dead in* N *if and only if* H *does not have a* t*-labeled edge.*

Marking Equation, Transition Invariants (☞ Sects. 15.1–15.3)

Theorem 20 (Finiteness Theorem of Positive Place Invariants)**.** *If each place of an elementary system net N has a positive place invariant, then only a finite number of markings is reachable in N.*

Definition 53. *For two markings M, M' of N, the marking equation of M and M' is the equation $M' = M + \underline{N} \cdot x$.*

Definition 54. *Let $\sigma = M_1 \xrightarrow{u_2} M_2 \xrightarrow{u_3} \cdots \xrightarrow{u_n} M_n$ be a finite sequence of steps $M_{i-1} \xrightarrow{t_i} M_i$. For each transition t_j let $a_j =_{def} |\{i \mid u_i = t_j\}|$. Then $a = (a_1, \ldots, a_\ell)$ is the counting vector of σ.*

Theorem 21 (Theorem on the Marking Equation)**.** *Let N be an elementary system net with a step sequence σ from a marking M to a marking M'. The counting vector of σ solves the marking equation for M and M'.*

Theorem 22 (Viability Theorem)**.** *Let N be an elementary system net with markings M and M' and a solution $\underline{a}$ to the marking equation (3). Then there exists a marking L of N and a step sequence σ from $\underline{M} + \underline{L}$ to $\underline{M'} + \underline{L}$ such that $\underline{a}$ is the counting vector of σ.*

Theorem 23 (Acyclic Viability Theorem)**.** *Let N be an acyclic elementary system net with markings M and M' and let $\underline{a}$ be a solution to the marking equation for M and M'. Then $\underline{a}$ is the counting vector of a step sequence from M to M'.*

Definition 55. *Each solution $(m_1, \ldots, m_k)$ of $N \cdot x = \vec{0}$ with $m_1, \ldots, m_k \in \mathbb{N}$ is a transition invariant of N.*

Theorem 24 (Transition Invariants Theorem)**.** *Let $\underline{a}$ be a transition invariant of an elementary system net N and let σ be a step sequence from a marking M to a marking M' such that $\underline{a}$ is the counting vector of σ. Then M and M' are identical.*

Theorem 25 (Reproducibility Theorem)**.** *If an elementary system net N does not have any transition invariants, then no marking is again reachable from itself.*

Run Properties (☞ Sects. 16.1, 16.3)

Definition 56. *Let e and f be state properties. Then the formula $e \mapsto f$ is a run property.*

Definition 57. *A run property $e \mapsto f$ holds in N (written $N \models e \mapsto f$) iff for each (finite or infinite) sequential run $w = M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots$ holds: To each i with $M_i \models e$ there exists a $j \geq i$ with $M_j \models f$.*

Definition 58. *For $Q \subseteq P$ and $t \in T$, let* $\mathrm{eff}(Q, t) =_{def} (Q \setminus {}^\bullet t) \cup t^\bullet$.

Theorem 26 (Theorem on Deduced Run Properties)**.** *Let N be an elementary system net and let Q be a subset of its places such that Q enables at least one hot transition of N. Let $T = \{t_1, \ldots, t_n\} \subseteq Q^\bullet$ such that $N \models Q \to \neg{}^\bullet t$ for each $t \in Q^\bullet \backslash T$. Then*

$$N \models Q \mapsto \mathit{eff}(Q, t_1) \vee \ldots \vee \mathit{eff}(Q, t_n).$$

Lemma 5. *Let e, f, g be state properties.*

(i) If $N \models e \mapsto f$ and $N \models f \mapsto g$ then $N \models e \mapsto g$.

(ii) If $N \models e \mapsto f$ and $N \models g \mapsto f$ then $N \models (e \vee g) \mapsto f$.

(iii) If $N \models e \mapsto (f \wedge g)$ then $N \models e \mapsto f$ and $N \models e \mapsto g$.

(iv) If $N \models e \to f$ then $N \models e \mapsto f$.

Free-Choice Nets (☞ Sects. 17.1–17.4)

Definition 59. *N is free-choice iff for each $p \in P$ and each $t \in T$ with $(p,t) \in F$ holds: $p^\bullet = \{t\}$ or ${}^\bullet t = \{p\}$.*

Theorem 27 (Trap/Cotrap Theorem for Free–Choice Nets)**.** *Let N be a free-choice net. Then N is live if and only if N has the trap/cotrap property.*

Definition 60.

(i) Let $p \in P$, $p^\bullet = \{t_1, \ldots, t_n\}$ and ${}^\bullet t_1 = \cdots = {}^\bullet t_n = \{p\}$. Then $\{p, t_1, \ldots, t_n\}$ is a fc-cluster.

(ii) Let $t \in T$, ${}^\bullet t = \{p_1, \ldots, p_n\}$ and $p_1^\bullet = \cdots = p_n^\bullet = \{t\}$. Then $\{t, p_1, \ldots, p_n\}$ is a ds-cluster.

Theorem 28 (Cluster Theorem for Free–Choice Nets)**.** *An elementary system net N is a free-choice net if and only if each place and each transition of N lies in exactly one cluster of N.*

Theorem 29 (Rank Theorem for Free-Choice Nets)**.** *For a connected free-choice net N, there exists an initial marking with which N is live and bounded if and only if*

a) N has a positive place invariant i, whose carrier contains each place of N

b) N has a transition invariant j whose carrier contains each transition of N

c) If the rank of $\underline{N}$ is k, then N has exactly $k+1$ clusters.

Marked Graphs (☞ Sects. 18.1, 18.2)

Definition 61.

(i) N is a marked graph iff N is elementary and for each $p \in P$ holds: $|{}^\bullet p| = |p^\bullet| = 1$.

(ii) $\{p_1, \ldots, p_n\} \subseteq P$ is a cycle iff for $i = 1, \ldots, n$ holds: $p_{i-1}^\bullet = {}^\bullet p_i$, with $p_0 =_{def} p_n$.

Theorem 30 (Cycle Theorem for Marked Graphs)**.** *Let N be a marked graph and let $p_1 \ldots p_n$ be a cycle of N, initially holding a total of k tokens. Then the following equation holds in N:*

$$p_1 + \ldots + p_n = k.$$

Theorem 31 (Liveness Theorem for Marked Graphs)**.** *A marked graph N is live if and only if each cycle of N contains at least one initially marked place.*

Theorem 32 (Theorem on Live and 1-bounded Marked Graphs)**.** *A live marked graph N is 1-bounded if and only if each place of N is part of a cycle that initially contains exactly one token.*

Theorem 33 (Theorem on Initial Markings of Marked Graphs)**.** *For each strongly connected marked graph N there exists an initial marking such that N is both live and 1-bounded.*

Well-Formed Elementary System Nets (☞ Sects. 19.2, 19.3)

Definition 62. *Let N be elementary and let E be a reachable marking (called* final*). Then (N, E) is* well-formed *iff*

(i) For no marking $L \neq \vec{0}$, $E + L$ is reachable,

(ii) Each transition of N is enabled in at least one reachable marking,

(iii) E is reachable from each reachable marking of N.

Theorem 34 (Well-formedness Theorem)**.** *An elementary system net N with a final marking E is well-formed if and only if N^* is live and bounded.*

Fairness Assumptions (☞ Sect. 20.3)

Definition 63. *Let N be elementary, let $w = M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots$ be an infinite sequential run and let $t \in T$. Then w* neglects fairness *for t iff infinitely many markings M_i enable t, but $t = t_j$ for only finitely many indices j. The run w respects* fairness *for t iff w does not neglect fairness for t.*

Net Schemata (☞ Sect. 22.2)

In contrast to Sect. 2.6, here we assume no fixed interpretation of the symbols in expressions. Rather, a net schema covers *all* interpretations. This requires a symbolic representation of markings. To this end, a marking M assigns each place p a set $M(p)$ of variable free expressions. $M(p)$ is frequently represented as a constant symbol, to be interpreted as a finite multiset. In a net schema, different expressions are never equivalent in the sense of Sect. 2.6. The only equivalences to calculate with are those for multiset expressions, as in Sect. 13.3.

Bibliography

[1] W. van der Aalst *Verification of workflow nets.* LNCS 1248 pp. 407-426 (1997)

[2] W. van der Aalst, N. Lohmann, P. Massuthe, Chr. Stahl, K. Wolf *Multiparty contracts: agreeing and implementing interorganizational processes.* Computing Journal (2008)

[3] M. Abadi, L. Lamport *Composing Specifications.* ACM Tr. Prog. Lang. Syst. 15(1), 73-132 (1993)

[4] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, G. Frankeschinis *Modeling with Generalized Stochastic Petri Nets.* Wiley (1995)

[5] B. Alpern, F. Schneider *Defining liveness.* Information Processing Letters 21, 181-185 (1985)

[6] E. Badouel, P. Darondeau *Theory of regions.* LNCS 1492 (1998)

[7] G. Balbo *Introduction to generalized stochastic Petri nets.* LNCS 4486, 83-131 (2007)

[8] L. Bernardinello *Synthesis of net systems.* LNCS 691, 89-105 (1993)

[9] E. Best, R. Devillers, M. Koutny *Petri Net Algebra.* Springer (2001)

[10] E. Best, C. Fernandez *Nonsequential Processes.* Springer (1988)

[11] N. Busi *Analysis issues in Petri nets with inhibitor arcs.* Theoretical Computer Science 275 (1-2) pp. 127-177 (2002)

[12] J. Cortadella, M. Kishinevsky, L. Lavagne, A. Yakovlev *Deriving Petri nets from finite transition systems.* IEEE Transactions on Computers 47, 859-882 (1998)

[13] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Yakovlev *Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers.* IEICE Transactions on Information and Systems E 80-D (3), 315-325 (1997)

[14] C.S. Calude, G. Paun, G. Rozenberg, A. Salomaa (eds) *Multiset Processing.* LNCS 2235 (2001)

[15] J. Desel, J. Esparza *Free Choice Petri Nets.* Cambridge Tracts in Theoretical Computer Science, Vol. 40 (1995)

[16] J. Desel, G. Juhás *What is a Petri net? Informal answers for the informed reader.* In: H. Ehrig, G. Juhás, J. Padberg, G. Rozenberg (Eds.): Unifying Petri Nets. LNCS 2128, 1-25

[17] J. Desel, K.P. Neuendorf, M.D. Radola *Proving nonreachability by modulo-invariants.* Theoretical Computer Science 153, S.49-64

[18] J. Desel, W. Reisig, G. Rozenberg *Lectures on Concurrency and Petri Nets.* LNCS 3098 (2004)

[19] E.W. Dijkstra *Cooperating Sequential Processes.* TR EWD-123 (1965)

[20] E.W. Dijkstra *Invariance and non-determinacy.* In: C.A.R. Hoare, J.C. Sheperdson (Ed.): Math. Logic and Programming Languages, Prentice Hall, S. 157-165 (1985)

[21] A. Ehrenfeucht, G. Rozenberg *Partial (set) 2-structures.* Acta Inf. 27, S. 315-368 (1990)

[22] J. Esparza *Decidability and complexity of Petri net problems – an introduction.* LNCS 1491, 374-428

[23] J. Esparza, K. Heljanko *Unfolding – A Partial Order Approach to Model Checking.* Springer (2008)

[24] J. Esparza, M. Nielsen *Decidability issues for Petri nets – a survey.* Journal of Informatics Processing an Cybernetics 30(3), S. 143-160 (1994)

[25] A. Finkel *The minimal coverability graph for Petri nets.* LNCS 674, pp. 210-243 (1993)

[26] H.J. Genrich *Das Zollstationenproblem.* GMD St. Augustin (1971)

[27] H.J. Genrich *Einfache nicht-sequentielle Prozesse.* GMD Bonn, Report No. 37 (1971)

[28] H. Genrich, K. Lautenbach *System modelling with high-level Petri nets.* Theoretical Computer Science 13, S. 109-134 (1981)

[29] C. Girault, R. Valk *Petri Nets for Systems Engineering.* Springer (2003)

[30] L. Gomes, J.P. Barros *Structuring and composability issues in Petri net modeling.* IEEE Tr. Industr. Inf. 1(2), 112-123 (2005)

[31] G. Goos *Vorlesungen über Informatik.* Vol. 1, Springer (1995)

[32] Y. Gurevich *Sequential abstract-state machines capture sequential algorithms.* ACM Transactions on Computational Logic 1, 1 pp. 77-111 (2000)

[33] M.H.T. Hack *Analysis of Production Schemata by Petri Nets.* MIT, Cambridge (1972)

[34] R. Hamadi, B. Benatallah *A Petri Net-based model for web service composition.* 14th Australian Database Conference (2003)

[35] D. Harel, R. Marelly *Come, Let's Play* Springer (2003)

[36] A.W. Holt, H. Saint, R. Shapiro, S. Warshall *Final Report of the Information Systems Theory Project.* Griffiss Air Force Base (1968)

[37] K. Jensen *Coloured Petri nets and the invariant method.* Theoretical Computer Science 14, S.317-336 (1981)

[38] K. Jensen, L.M. Kristensen *Coloured Petri Nets.* Springer (2009)

[39] R.M. Karp, R.E. Miller *Parallel program schemata.* J. Comp. Syst. Sci. 3, pp. 147-195 (1969)

[40] E. Kindler, R. Walter *Mutex Needs Fairness.* Information Processing Letters 62, 31-39 (1997)

[41] P. Kritzinger, V. Bause *Introduction to Stochastic Petri Nets.* Vieweg Verlag (2002)

[42] L. Lamport *Specifying Systems.* Addison Wesley (2002)

[43] K. Lautenbach *Liveness in Petri Nets.* GMD St. Augustin 75-02-1 (1975)

[44] K. Lautenbach *Linear algebraic calculations of deadlocks and traps.* In: K. Voss, M.J. Genrich, G. Rozenberg (Eds.) Concurrency and Nets. Springer (1987)

[45] R. Lorenz, S. Mauser, G. Juhas *How to synthesize nets from languages: a survey.* 39th Winter Simulation Conference, IEEE Press, 637-647 (2007)

[46] Z. Manna, A. Pnueli *The Temporal Logic of Reactive and Concurrent Systems.* Springer (1992)

[47] E.W. Mayr *Ein Algorithmus für das allgemeine Erreichbarkeitsproblem bei Petrinetzten und damit zusammenhängende Probleme.* TU München (1980)

[48] E.W. Mayr *An algorithm for the general Petri net reachability problem.* SIAM J. Computing 13, 3, 441-460 (1984)

[49] K.L. McMillan *Using unfoldings to avoid the state explosion problem in the verification of asynchronous cicuits.* LNCS 663, 164-177 (1993)

[50] J. Misra, K.M. Chandy *Parallel Program Design: A Foundation.* Addison-Wesley (1988)

[51] T. Miyamoto, S. Kumagai *A survey of object oriented Petri nets and analysis methods.* IEICE E88-A, 2964-2971 (2005)

[52] U. Montanari, F. Rossi *Contextual nets.* Acta Informatica 32, 545-596 (1995)

[53] T. Murata *State equation, controllability and maximal matchings of Petri nets.* IEEE Trans Autom. Contr. 22(3), 412-416 (1977)

[54] M. Nielsen, G. Plotkin, G. Winskel *Petri nets, event structures and domains.* Theoretical Computer Science 13, 85-108 (1981)

[55] S. Owicki, L. Lamport *Proving liveness properties of concurrent programs.* ACM Trans. on Programming Languages and Systems, 4 (3) 455-495 (1982)

[56] C.A. Petri *Kommunikation mit Automaten.* Institut für Instrumentelle Mathematik, Bonn, IIM Report No. 2 (1962)

[57] C.A. Petri *Grundsätzliches zur Beschreibung diskreter Prozesse.* 3. Colloquium über Automatentheorie (3rd Colloq. on Automata Theory), Birkhäuser, 121-140 (1967)

[58] C.A. Petri *Non-sequential Processes.* GMD St. Augustin (1977)

[59] C.A. Petri *Nets, time and space.* Theoretical Computer Science 153, S. 3-48 (1996)

[60] Petri Net World: *http://www.informatik.uni-hamburg.de/TGI/PetriNets/*

[61] L. Priese, H. Wimmel *Theoretische Informatik – Petri Netze.* Springer (2003)

[62] W. Reisig *Petrinetze – eine Einführung.* Springer (1982)

[63] W. Reisig *Petri Nets – An Introduction.* Springer (1985)

[64] W. Reisig *Elements of Distributed Algorithms.* Springer (1998)

[65] W. Reisig *50 Jahre Modellbasierter Entwurf: Vom Modellieren mit Programmen zum Programmieren mit Modellen.* In: W. Reisig, J.-C. Freytag (Eds.): Informatik – Aktuelle Themen im historischen Kontext, Springer (2006)

[66] W. Reisig *The Decent Philosophers: An exercise in concurrent behaviour.* Fundamenta Informaticae 80 1-9 (2007)

[67] W. Reisig *The Scholten/Dijkstra Pebble Game.* LNCS 4800, 589-595 (2008)

[68] W. Reisig *Simple composition of nets.* LNCS 5606 pp. 23-42 2009

[69] W. Reisig, W. Brauer *Carl Adam Petri und die Petrinetze.* Informatik-Spektrum 29(5), Oktober 2006, Springer, 369-374 (2006). English version: E. Gelenbe (ed) *Fundamental Concepts in Computer Science 3.* Imperial College Press (2007)

[70] W. Reisig, G. Rozenberg *Lectures on Petri Nets I, II.* LNCS 1491, 1492 (1998)

[71] C. Reutenauer *The Mathematics of Petri Nets.* Prentice Hall (1990)

[72] K. Schmidt *Verification of siphons and traps for algebraic Petri nets.* LNCS 1248, 427-446 (1997)

[73] K. Schmidt *Model-checking with coverability graphs.* Formal Methods in System Design 15(3), 239-254 (1999)

[74] H. Störrle *Models of Software-Architecture-Design and Analysis with UML and Petri Nets.* Books on Demand (2000)

[75] R. Valk *Self-modifying nets, a natural extension of Petri nets.* LNCS 62 pp. 464-476 (1978)

[76] R. Valk *Object Petri nets.* LNCS 3098, pp. 819-848 (2004)

[77] A. Valmari *The State Explosion Problem.* In: W. Reisig, G. Rozenberg (Eds.), LNCS 1491 (1998)

[78] W. Vogler *Concurrent implementation of asynchronous transition systems.* LNCS 1639, 284-303

[79] A. Wegrzyn, A. Kavatevich, J. Bieganowski *Detection of deadlocks and traps in Petri nets by means of Thelen's prime implicant method.* Int. J. Appl. Math. Comp. Sci. 14, 1 (113-121) (2004)

[80] M. Yamauchi, T. Watanabe *Time complexity analysis of the minimal siphon extraction problem of Petri nets.* IEICE Tr. Fund. El. Com. Comp. Sci., E 82-A, 11, 2558-2565 (1999)

[81] A.V. Yakovlev, A.M. Koelmans *Petri nets and digital hardware design.* LNCS 1492, 154-236 (1998)

Index