Monographs in Theoretical Computer Science
An EATCS Series

Roberto Gorrieri

Process Algebras for Petri Nets

The Alphabetization of Distributed Systems

Springer

Monographs in Theoretical Computer Science
An EATCS Series

More information about this series at http://www.springer.com/series/776

Roberto Gorrieri

Process Algebras for Petri Nets

The Alphabetization of Distributed Systems

Springer

Roberto Gorrieri
Dipartimento di Informatica - Scienza e Ingegneria
Università di Bologna
Bologna, Italy

ISSN 1431-2654 ISSN 2193-2069 (electronic)
Monographs in Theoretical Computer Science. An EATCS Series
ISBN 978-3-319-55558-4 ISBN 978-3-319-55559-1 (eBook)
DOI 10.1007/978-3-319-55559-1

Library of Congress Control Number: 2017936136

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To Nicoletta, Lucia, Emma and Caterina

Foreword

Distributed, open-ended systems are ubiquitous in today's information and communication technology: in most cases they exhibit an independent, concurrent behavior. The process of specification, design and development requires formal models able to express interesting properties and to allow for efficient validation and verification procedures. In addition to their role as design tools in ICT engineering, distributed, concurrent models are very successful in formalizing key concepts of biology, economics, complex systems and so on.

In many cases it is convenient to develop specialized models for the classes of systems we are interested in, equipped with convenient features. For instance, while in all cases a formal semantics is required, there is a conflict between model expressiveness and complexity of verification algorithms. Another important choice is between a graphical model and a text-based model, both equipped with observation criteria and consequent semantics: the former is often more perspicuous, but less convenient for specifying large, modular systems in a compositional way.

The main achievement of this book is to present a hierarchy of six models of increasing expressiveness, each of them described both in graphical form and in textual, term-like form. Plenty of results guarantee that there is a one-to-one correspondence between the two representations, and that corresponding forms have the same interleaving and step semantics.

In reality, graphical and textual forms are not equivalent: the former is more expressive since its semantics can be enriched with observational criteria to make certain relevant properties evident, which are present, but hidden, in the latter. For instance, the presentation in graphical form of the solution of the dining philosophers problem can be shown to be correct, while the textual version, once the choice is made of observing the behavior sequentially, has too restricted observation capabilities. In other words, as soon as the textual form is translated into the graphical form, new observation and analysis methods become applicable. Thus the translation has the flavor of associating with the language a more expressive concurrent semantics.

In the book, a clear choice is made about which models to handle: process description language CCS and Place/Transition (P/T) Petri nets, with various syntactic restrictions to yield the expressiveness hierarchy. Only the top level, Nonpermissive nets, is an extension of P/T nets: the extension is needed to achieve Turing-completeness.

I fully agree. There are several reasons for the choice. CCS is equipped with a rich theory, as testified by several textbooks and monographs, some of them very recent. Also, CCS is the starting point of the most popular language for mobile systems, i.e. π-calculus, and of its extensions for security and service-oriented architectures. P/T Petri nets were the first concurrency model (Carl Petri's thesis is dated 1962), and they are maybe the model with the richest theory, the largest application experience and the most extensive collection of tools. Furthermore, (unsafe) finite P/T nets have an easy-to-understand cyclic behavior with a possibly infinite set of states, but some important properties, e.g. reachability, are still decidable.

The technical content of the book is of extremely high quality. Actually, the author personally contributed with a number of seminal, well-known papers to the historical development of the main concepts and results in the book. Also, the presentation of complex results, which appeared over a large span of years (the bibliography contains about 150 citations), required a remarkable effort to make them consistent and uniform. In addition, some important results, needed to complete the full picture, are published in the book for the first time, as appropriate in a research monograph. Finally, the presentation and justification of the conceptual developments and of the formal achievements is well articulated and explained, and supported by plenty of examples as required in an advanced textbook.

The book will be precious for those interested in the truly concurrent semantics of distributed communicating systems. A system designer, given a specification in CCS, might want to better understand its behavior by looking at the corresponding net, by checking there the given concurrency requirements, and possibly by taking advantage of existing tools. More relevantly from a methodological point of view, a computer scientist, aiming at studying a particular class of concurrent distributed systems, might define a specification language based on variants of CCS and Petri nets by applying the approach in the book.

Ugo Montanari *Pisa, June 2016*

Acknowledgements

I would like to thank Ugo Montanari and Pierpaolo Degano, who, together with Rocco De Nicola, began research relating process algebras and Petri nets just one year before I started my Ph.D. studies, under their illuminating supervision. This experience has certainly influenced my interest in this topic, and in the end led me to write this book.

Other scholars I have worked with also influenced this book. First of all, I should mention Ursula Goltz, whose idea of using unsafe nets, instead of safe nets, was important for the development of this book. Then also Arend Rensink, whose work, together with Goltz, explained that the unguarded sum is not a good operator if the goal is to get finite nets.

I would also like to thank my former Ph.D. students Nadia Busi, who unexpectedly passed away in 2007, Cristian Versari, and my former master's student Massimo Morara, for their work on distributed semantics of some process algebras, which influenced in places the formulation of the ideas presented in this book. Versari also provided some figures for this book.

Some colleagues read some chapters and gave advice; the list includes: Pierpaolo Degano, Ugo Montanari, Davide Sangiorgi, Valeria Vignudelli and Gianluigi Zavattaro. Karina Gardella is thanked for some hints on the use of English. Finally, thanks to Grzegorz Rozenberg for supervising the evaluation of this book, to an anonymous reviewer for his/her detailed comments, and, last but not least, to Ronan Nugent for his assistance in the publishing process.

Roberto Gorrieri *Bologna, November 2016*

Contents

Chapter 1
Introduction

Abstract This introductory chapter outlines the main problem dealt with in this book: finding suitable languages for representing classes of Petri nets, taking inspiration from the process algebras developed in the last four decades. The structure of the book is outlined and some hints on how to read it are presented. Finally, it is also argued that Turing-completeness is not a sufficient criterion to compare the expressive power of different process algebras, because the sets of problems in distributed computing that two languages can solve may be different, even if both include all the Turing-computable functions.

1.1 The Alphabetization of Distributed Systems

A distributed system is a computer system made of several components, implemented in hardware or software or as a combination of both, that may be located at different sites, even at a geographical distance, and that cooperate to accomplish a task or coordinate to offer a service by means of suitable communication protocols based on message passing. The most notable example of a distributed system is the *Internet*, whose most important service is the *World Wide Web*.

At a very abstract level of detail, the main feature of a distributed system is *distribution*: the global state of the system is composed of a collection of local states, physically located at different sites, and each activity that the system performs may actually involve only a subset of these local states. Another important feature is that communication takes place only by *message passing*, so that any information exchange happens by means of explicit communication primitives of *send* or *receive*; in other words, there is no global memory, shared by the components. The communication mechanism can be *synchronous* or *asynchronous*: the former when the send action and the receive one are performed by the interacting partners at the same time; the latter when the send action is decoupled from the receive one.

Many other features of distributed systems are relevant, such as heterogeneity of the components, possible independent failure of components, the absence of a

R. Gorrieri, *Process Algebras for Petri Nets*, Monographs in Theoretical Computer Science. An EATCS Series, DOI 10.1007/978-3-319-55559-1_1

global clock, scalability etc., but for the aims of this book, only distribution and communication are considered. In particular, communication is assumed to be synchronous, because this mechanism can also easily implement the asynchronous one (it is enough to put a medium, such as a buffer, between the two partners of the communication to get an asynchronous communication), while the reverse is more difficult to achieve.

Many semantic models of computation have been proposed to model distributed systems; here we give a short, not exhaustive, list:

- Petri nets [Petri62, Hack76b, Pet81, Rei85, MM90, JK95, MR95, Bus02, Rei13];
- Transition systems [Kel76, Mil80, Plo04b, Gla01, San12, GV15];
- Event structures [NPW81, Win87, Win88, BMM06];
- Causal trees [DD89, DD90, BMS15];
- Concurrent histories [DM87];
- Statecharts [Har87, HPSS87];
- Message sequence charts [RGG, DH99];
- Kahn process networks [Kah74].

Each of these has its own pros and cons. However, among them, we chose Petri nets, for the following reasons:

1. distribution is a first-class concept (which is not the case for, e.g., labeled transition systems);
2. Petri nets can model recursive behavior with a finite structure (which is not the case for, e.g., event structures);
3. they are a widely studied model (see, e.g., [RR98a, DRR04] and the references therein), equipped with a simple, precise, formal semantics, both for the so-called linear-time and branching-time semantics (which is not the case for, e.g., message sequence charts);
4. they are equipped with analysis techniques that are decidable in many cases, as described in Section 3.3 (which is not the case for, e.g., concurrent histories), and that are sometimes supported by automatic or semi-automatic software tools (see, e.g., [TA15, Tool] for surveys on Petri net tools); and, finally,
5. there is a large literature of applications of Petri nets to the modeling of real distributed systems (see, e.g., [RR98b, Rei98]) and the references therein).

From now on, at an abstract level, we take the liberty of identifying a *distributed system* with the *Petri net* which models it. Therefore, we shall consider specific classes of Petri nets as specific classes of distributed systems.

Many specification languages have been proposed to describe reactive, distributed systems, starting from the seminal work by Hoare with CSP [Hoa78, Hoa85, Ros98], and Milner with CCS [Mil80, Mil89, GV15]. These languages are usually called *process algebras*, to reflect the algebraic nature of their syntactic and semantic definitions. Many process algebras have been proposed in the literature: besides CSP and CCS, also ACP [BK84, BW90, BBR10] by Bergstra, Klop and

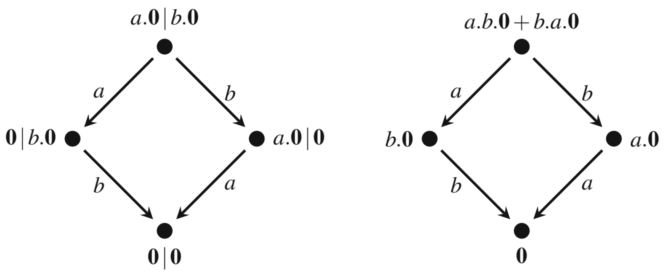

$a.\mathbf{0}\,|\,b.\mathbf{0}$
a
b
$\mathbf{0}\,|\,b.\mathbf{0}$
$a.\mathbf{0}\,|\,\mathbf{0}$
b
a
$\mathbf{0}\,|\,\mathbf{0}$
$a.b.\mathbf{0}+b.a.\mathbf{0}$
a
b
$b.\mathbf{0}$
$a.\mathbf{0}$
b
a
$\mathbf{0}$

Fig. 1.1 Interleaving law: two isomorphic LTSs

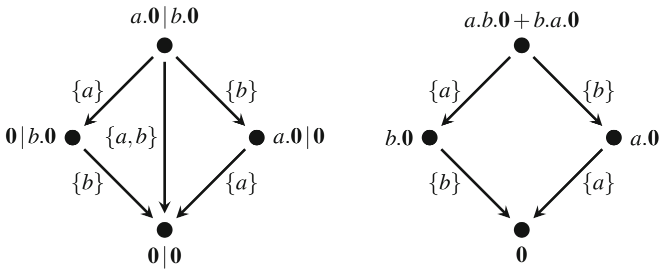

$a.\mathbf{0}\,|\,b.\mathbf{0}$
$\{a\}$
$\{b\}$
$\mathbf{0}\,|\,b.\mathbf{0}$
$\{a,b\}$
$a.\mathbf{0}\,|\,\mathbf{0}$
$\{b\}$
$\{a\}$
$\mathbf{0}\,|\,\mathbf{0}$
$a.b.\mathbf{0}+b.a.\mathbf{0}$
$\{a\}$
$\{b\}$
$b.\mathbf{0}$
$a.\mathbf{0}$
$\{b\}$
$\{a\}$
$\mathbf{0}$

Fig. 1.2 Two step transition systems

Baeten (who coined the term *process algebra*), LOTOS [BB87, BLV95] by Bolognesi and Brinksma (who included abstract data types in a process algebra), CIRCAL [Mil85] by Milne (who introduced the step semantics for a process algebra), and the π-calculus [MPW92, Mil99, Par01, SW01] by Milner, Parrow and Walker (which models mobility), just to mention a few (see [Bae05] for a historical overview, and [BPS01] for a technical survey on the many facets of process algebras). The semantics of these languages have mainly been given in an *interleaving* style, either in terms of execution traces (e.g., CSP) or in terms of labeled transition systems (e.g., CCS), but in no way are parallelism (or co-occurrence of independent actions) and distribution modeled in these semantics.

An example may help clarify the idea. The process composed of two parallel actions a and b is denoted in CCS by the term $a.\mathbf{0}\,|\,b.\mathbf{0}$, while the sequential process performing these two actions in either order is denoted by $a.b.\mathbf{0}+b.a.\mathbf{0}$. In the labeled transition system semantics of [Mil89], these two CCS processes originate the isomorphic labeled transition systems in Figure 1.1, so that they are semantically equal. This is the essence of the so-called *interleaving law*: a (finite-state) parallel process is semantically equivalent to a (finite-state) possibly nondeterministic, sequential process. This is a clear drawback of the interleaving semantics, as we will argue further in Section 1.4.

One may enrich the labeling of the transition system by using, instead of single actions, multisets of concurrently executable actions: this is the so-called *step* semantics (originally introduced in [NT84, Mil85]), which is refined enough to model

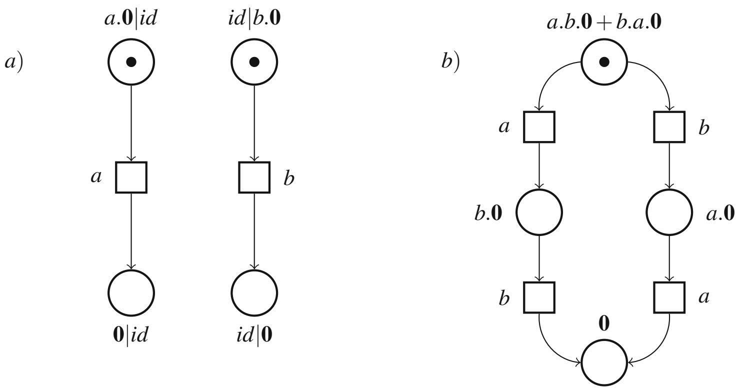


Fig. 1.3 The Petri nets for $a.\mathbf{0} \,|\, b.\mathbf{0}$ in a) and $a.b.\mathbf{0}+b.a.\mathbf{0}$ in b), according to the DDM technique

the potential parallelism of a process. According to the step semantics, $a.\mathbf{0} \,|\, b.\mathbf{0}$ and $a.b.\mathbf{0}+b.a.\mathbf{0}$ are not equivalent, as only the former can perform a transition labeled with the set $\{a,b\}$, denoting that these two actions can be performed at the same time, as outlined in Figure 1.2.

Still, no information on state distribution is actually given. To achieve this, one has to consider a more detailed model, where the monolithic, global state of the system is decomposed into a collection of local states, such that each action the system performs may be actually due to a subset of the local states, only: this is indeed what Petri nets can model, where the global state, called a *marking*, is a multiset of local states, called *places*.

Starting from the seminal work by Degano, De Nicola, Montanari and Olderog [DDM88, Old91], CCS and CCSP (i.e., a mixture of CCS and CSP) have been equipped with an operational Petri net semantics, so that distribution is a first-class concept, and parallelism and nondeterminism are modeled differently. For instance, according to this approach, the Petri nets for $a.\mathbf{0} \,|\, b.\mathbf{0}$ and $a.b.\mathbf{0}+b.a.\mathbf{0}$, depicted in Figure 1.3, show visibly that the former process is composed of two sequential subprocesses, running possibly in parallel, while the latter is just one sequential process. To be more precise, $a.\mathbf{0} \,|\, b.\mathbf{0}$ originates a distributed global state (i.e., a marking) composed of two local states: place $a.\mathbf{0}|id$, representing the sequential process $a.\mathbf{0}$ decorated with information about the context (it is the left component of a parallel process), and place $id|b.\mathbf{0}$. The number of tokens in a place denotes the number of instances of that sequential process, that are available in the current global state; in our example, only one token is present in place $a.\mathbf{0}|id$ and in place $id|b.\mathbf{0}$. A transition (represented as a labeled box) is defined by a triple (m_1,ℓ,m_2), where m_1 is its *pre-set*, i.e., the marking specifying the tokens to be consumed for its executability, ℓ is its label, and m_2 is its *post-set*, i.e., the marking specifying the token to be produced upon completion; in our example, the two transitions are $(\{a.\mathbf{0}|id\},a,\{\mathbf{0}|id\})$ and $(\{id|b.\mathbf{0}\},b,\{id|\mathbf{0}\})$. Of course, these two transitions can be performed in either order, independently of each other, and even in parallel: at the

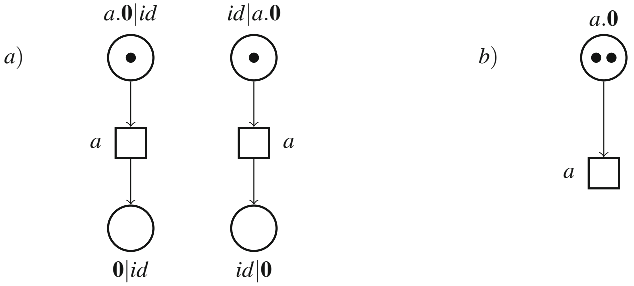


Fig. 1.4 The nets for $a.\mathbf{0}\,|\,a.\mathbf{0}$, according to the DDM technique in a) and Goltz's technique in b)

same time, the two tokens in the initial marking $\{a.\mathbf{0}|id, id|b.\mathbf{0}\}$ can perform their respective transitions and end up in the final marking $\{\mathbf{0}|id, id|\mathbf{0}\}$. On the contrary, $a.b.\mathbf{0}+b.a.\mathbf{0}$ is modeled with a marking which is composed of one single token (i.e., one single instance of $a.b.\mathbf{0}+b.a.\mathbf{0}$), which can perform its transitions only sequentially.

This technique, sometimes called the DDM technique, was easily exploited also to give a net semantics to other languages [DGM88, BG89, GV11], witnessing the great generality and wide applicability of the idea. However, this semantics has one major drawback: the exploited class of Petri nets is limited to so-called *safe* Petri nets, i.e., nets where each place may contain at most one token, so that identical parallel subprocesses are mapped to distinct places. Goltz [Gol88, Gol90] proposed a denotational *unsafe* Petri net semantics for the CCS subcalculus without restriction; this idea was also followed by [GM90b] in an operational setting. To clarify the semantic difference between the DDM technique and Goltz's, let us consider the CCS process term $a.\mathbf{0}\,|\,a.\mathbf{0}$. Figure 1.4(a) shows its associated net according to the DDM technique, where the two identical instances of the sequential process $a.\mathbf{0}$ are mapped to two distinct places, because of the contextual decoration; Figure 1.4(b) shows the net obtained according to Goltz's approach, which is an unsafe net with two tokens on the same place $a.\mathbf{0}$.

Many other researchers studied the problem of relating process algebras and Petri nets (see, e.g., [GV87, Tau89, GR94, MY94, GM95, Mey09, BBGM14] and the references therein). Of particular interest is the work by Best, Devillers and Koutny [BDK01, BDK02], who proposed a rather rich algebra of expressions, whose semantics is given, however, in terms of safe nets only. All these papers focus on the problem of defining a suitable distributed, net-based semantics for some chosen process algebra; hence, we can label this line of research with the motto

Petri Nets for Process Algebras.

The main aim of this book is to approach the reverse problem of finding the minimal set of process algebraic operators that are necessary in order to model all and only the Petri nets of a certain class, whence the title

Process Algebras for Petri Nets.

This book offers a definitive answer to this question. We single out six classes of finite Petri nets, and, for each class, we define a corresponding process algebra, such that:

- each term p of the process algebra is given a distributed semantics in terms of a net $Net(p)$ of that class (*only* nets of that class can be modeled by the process algebra); moreover,
- for each net $N(m_0)$ of that class (i.e., for each net N with initial marking m_0), we can single out a term p of the corresponding process algebra, whose semantics is a net isomorphic to $N(m_0)$ (*all* the nets of that class are represented by the process algebra, up to net isomorphism); and, finally,
- all the operators of the process algebra are necessary to get these results (the set of operators, for each proposed process algebra, is irredundant).

Therefore, since a class of finite Petri nets is meant as a class of distributed systems, our contribution amounts to *alphabetizing* these six classes of distributed systems, i.e., providing six languages, as simple as possible, each one representing all and only the distributed systems of a certain class.

1.2 The Hierarchy

Which classes of Petri nets will be considered? And which corresponding process algebras will be singled out? Here we describe the hierarchy of the six classes of nets and languages, which is also outlined in Figure 1.5, starting from the least expressive one. All the classes of nets, except the last one, are subclasses of so-called *Place/Transition* Petri nets (P/T nets, for short, see Chapter 3 for details).

1. *Sequential Finite-State Machines*: A *finite-state machine* is a finite Petri net whose transitions are strictly sequential, i.e., the pre-set and the post-set are singletons so that each transition is performed by a single sequential process that evolves into a single sequential process; such a net is *sequential* if the initial marking of the net is a singleton; therefore, a sequential finite-state machine describes a strictly sequential system composed of one single sequential process only. The corresponding process algebra is called SFM, and is essentially *finite-state CCS* [Mil89], whose operators are the empty process $\mathbf{0}$, action prefixing $\mu.p$, choice $p+q$, and constants C, equipped with a definition useful for describing recursive behavior.
2. *Concurrent Finite-State Machines*: A finite-state machine is *concurrent* when the initial marking is arbitrary, hence not necessarily a singleton. In this way, the distributed systems that nets of this sort are able to model are composed of a collection of strictly sequential processes that work in parallel, but without any form of synchronization. The corresponding process algebra is called CFM, and enriches SFM with the binary operator of *asynchronous* (i.e., without synchronization capabilities) parallel composition $p \mid q$, to be used at the top level only.

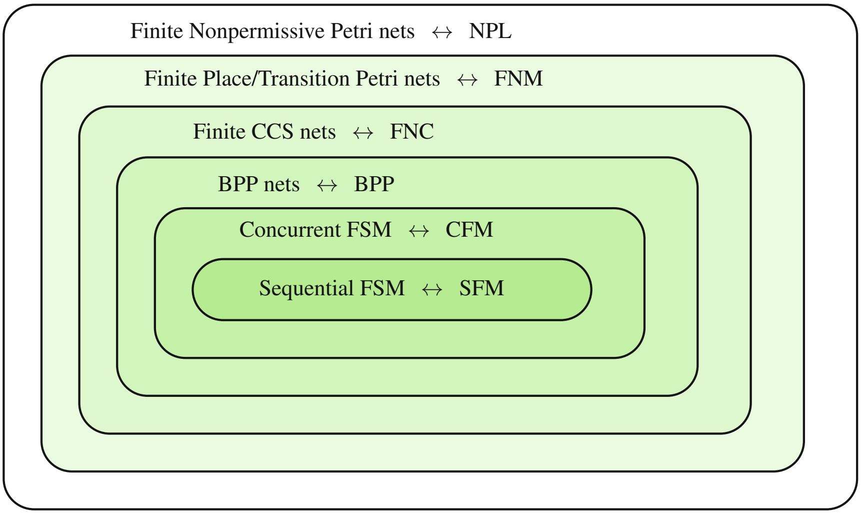

Finite Nonpermissive Petri nets ↔ NPL
Finite Place/Transition Petri nets ↔ FNM
Finite CCS nets ↔ FNC
BPP nets ↔ BPP
Concurrent FSM ↔ CFM
Sequential FSM ↔ SFM

Fig. 1.5 The hierarchy of net classes and process algebras

3. *BPP nets*: The term BPP is the acronym of *Basic Parallel Processes*, and was coined in [Chr93] to denote a simple process algebra, which essentially extends CFM by modifying the action prefixing operator: in a term $\mu.p$, the action μ may prefix not only a sequential process (as for CFM), but also a parallel process. The class of finite nets that such a language can model are called *BPP nets*, whose distinguishing feature is that the pre-set of each transition is a singleton, while the post-set can be arbitrary.
4. *Finite CCS nets*: If BPP is extended by enhancing its parallel composition operator to allow for communication (according to the binary, handshake, synchronization discipline of CCS), and by also including the CCS restriction operator $(\nu a)p$ (in order to force synchronization within p), to be used at the top level only, then the resulting calculus, called FNC, is essentially *finite-net CCS* [GV15]. The class of nets that this language can represent are called *finite CCS nets*: they are all the finite Petri nets such that the pre-set size of each transition can be at most two and, if it is two, then the labeling of this transition must be the invisible action τ.
5. *Finite P/T nets*: The only constraint on this class of Petri nets is that they must be finite; in particular, the pre-set of a transition can be of arbitrary size and with an arbitrary labeling. The process algebra FNC is to be extended with a mechanism for multi-party synchronization; this is achieved by introducing a new prefixing operator, called *strong prefixing* in opposition to the normal one and whose syntactic form is $\underline{a}.p$, which can model atomic actions; a multi-party synchronization is modeled as an atomic sequence of binary, CCS-like synchronizations. The resulting process algebra is called FNM and is a slight simplification of *finite-net Multi-CCS* [GV15].
6. *Nonpermissive Petri nets*: This class is a generalization of finite P/T nets, where a transition also specifies a set of places to be *tested for absence* of additional

tokens, called the *neg-set* of the transition. In fact, differently from finite P/T nets, a transition can be executed if and only if the required tokens in the pre-set are available, and also no additional token is present in any place belonging to the *neg-set* of the transition. Therefore, a transition that may be executable at a given marking m may be disabled at a larger marking m', because some of the additional tokens of m' may inhibit the executability of the transition. Differently from finite P/T nets, Nonpermissive Petri nets, which are a generalization of P/T nets with inhibitor arcs [FA73, Hack76a, Pet81, JK95, Bus02], are a Turing-complete formalism. The corresponding process algebra, called NPL, extends FNM with a variant of the strong prefixing operator, $\underline{\neg D}.p$, whose strong prefix $\neg D$ expresses that a transition originating from p is actually executable by $\underline{\neg D}.p$ in a parallel context, provided that no instance of constant D is currently active in this parallel context; hence, $\underline{\neg D}.p$ executes a test for absence of the testable constant D, *atomically* with the action p is able to perform.

Summing up, the set of process algebraic operators that we have singled out, in order to represent these six classes of nets, is really small: it comprises essentially the CCS operators, some of which are to be used in some constrained form, and the strong prefixing operator, originally introduced in [GMM90, GM90a], together with a suitable synchronization discipline for atomic sequences; only for NPL we have to introduce a newly conceived operator, which is, however, a variant of the strong prefixing operator.

1.3 Structure of the Book

Chapter 2 introduces the model of labeled transition systems and some behavioral equivalences over them; this part of the chapter is just a summary of Chapter 2 of [GV15]. Moreover, it includes also a description of the step transition systems model, equipped with some sensible behavioral equivalences.

Chapter 3 gives an introduction to Petri nets. Most of the material is standard, but some original contributions are also outlined. First, the concept of a statically reachable subnet is introduced because it will play a central role in the net semantics of the languages FNC, FNM and NPL. In fact, the semantics of a term p is not simply the net that can be dynamically reached from the initial marking $dec(p)$ corresponding to p, rather it is the net describing all the potential behaviors of p, as if the number of tokens in $dec(p)$ can be increased at will. For instance, consider the FNC sequential process $p = a.\mathbf{0} + \overline{a}.\mathbf{0}$; its dynamically reachable subnet, outlined in Figure 1.6(a), shows the expected behavior that p can perform a or its complementary action $\overline{a}$. On the contrary, its statically reachable subnet in (b) describes additionally the potential self-synchronization of p with another copy of itself; in this way, the net semantics for p and for the parallel term $p \,|\, p$ differ only for the form of the initial marking (one token for p and two tokens for $p \,|\, p$), but the underlying net is the same. The statically reachable subnet is always algorithmically computable, also for Nonpermissive nets, even though they are a Turing-complete

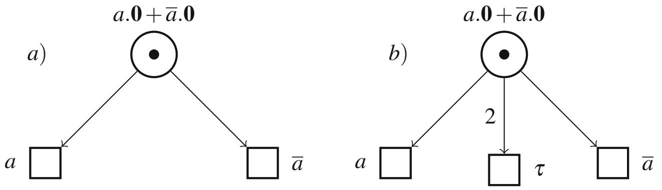

$a.\mathbf{0}+\overline{a}.\mathbf{0}$
$a)$
a
$\overline{a}$
$a.\mathbf{0}+\overline{a}.\mathbf{0}$
$b)$
2
a
τ
$\overline{a}$

Fig. 1.6 The dynamically reachable subnet (a) and the statically reachable subnet (b) for $a.\mathbf{0}+\overline{a}.\mathbf{0}$

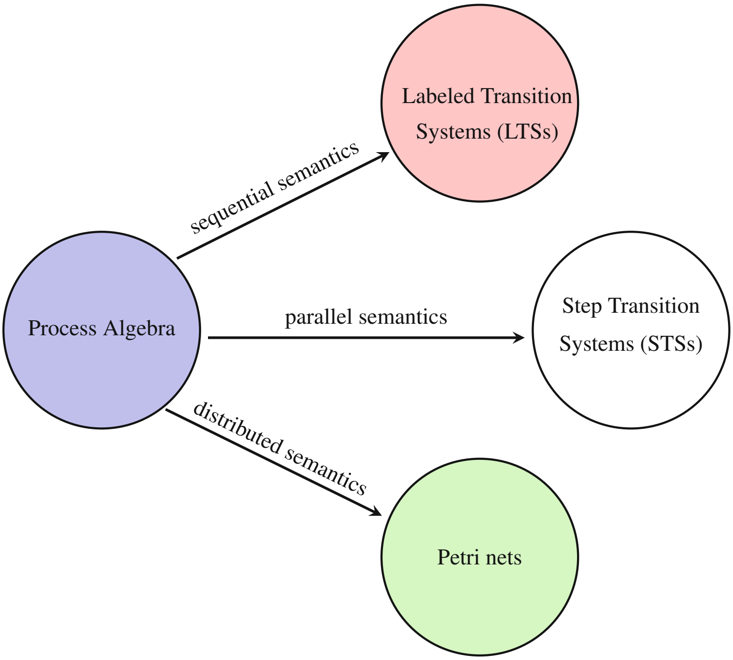


Fig. 1.7 Three semantics for each process algebra

model of computation, and so for them, as the reachability problem is undecidable, it is not always possible to compute the dynamically reachable subnet. As a matter of fact, another original contribution of this chapter is the introduction of the class of Nonpermissive Petri nets (a proper extension of P/T nets with inhibitor arcs), equipped with nonstandard behavioral equivalences, deviating from the traditional definitions for P/T nets with inhibitor arcs.

Chapters 4-8 present the six calculi: SFM in Chapter 4, CFM and BPP in Chapter 5, FNC in Chapter 6, FNM in Chapter 7 and NPL in Chapter 8. Each calculus is equipped with three different semantics, as illustrated in Figure 1.7: an *interleaving* (or *sequential*) semantics by means of labeled transition systems, a *step* (or *parallel*) semantics by means of step transition systems, and a *net* (or *distributed*) semantics by means of finite Petri nets. For each process algebra, if two terms are equivalent according to the net semantics, then they are equivalent in the step semantics; and,

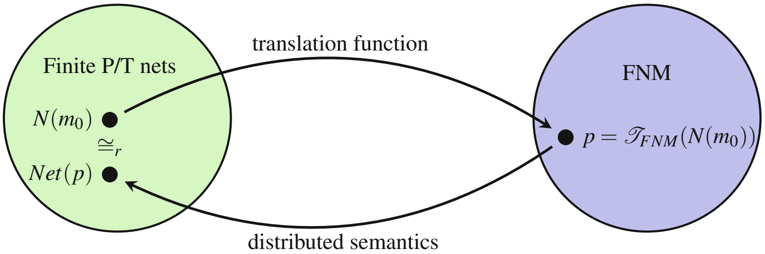


Fig. 1.8 Graphical description of the representability theorem for finite P/T nets and FNM

similarly, if two terms are equivalent according to the step semantics, then they are equivalent in the interleaving semantics. Therefore, these three semantics are ordered from the most abstract (the interleaving/sequential one) to the most concrete (the net/distributed one).

These five chapters all have the same structure. First, the syntax of the calculus is given, followed by the interleaving/sequential semantics, defined in an operational style, according to Plotkin's SOS technique [Plo04b]. Then, the step/parallel semantics is introduced by means of an obvious generalization of the SOS technique; this parallel semantics is followed by the operational Petri net semantics, which is described in tight correspondence with the style of the transition system semantics, in order to show the similarities of the two. As a matter of fact, the SOS approach can also easily be adapted to generate Petri nets instead of labeled transition systems. This distributed semantics is followed by the *representability theorem*: for any marked net $N(m_0)$ of the class, we can single out a term p of the corresponding language, whose semantics is the net $Net(p)$ isomorphic to $N(m_0)$; in this sense, we can say that p *represents* $N(m_0)$, up to isomorphism. Figure 1.8 describes graphically this result for the case of finite P/T nets and FNM: given a finite P/T marked net $N(m_0)$, the *translation function* $\mathscr{T}_{FNM}(-)$ maps $N(m_0)$ to the FNM process term $p = \mathscr{T}_{FNM}(N(m_0))$; then, the distributed semantics associates $Net(p)$ with p, such that $Net(p) \cong_r N(m_0)$, where $\cong_r$ denotes (marked) net isomorphism equivalence. The soundness of the net semantics w.r.t. the step semantics is also proved; in fact, the step semantics is mainly used as a sort of sanity check for the distributed semantics, in order to prove that the distributed semantics preserves all the intended parallel behavior. Finally, an unsafe net semantics in denotational style is introduced and proved to be coherent with the operational net semantics (in most cases, the two semantics coincide). For the cases of FNC, FNM and NPL, we were able to define these denotational semantics only because we chose to give semantics to a process p by means of its statically reachable subnet instead of its dynamically reachable subnet; this is an original contribution of this book, too.

Some repetitions in these chapters are inevitable, because each one builds on top of the previous one; this is particularly true for Chapters 4 and 5, and for Chapters 6, 7 and 8. However, as the Romans said, *repetita iuvant!* In fact, the presentation style is rather didactic, as the expected audience is composed of Ph.D. students and scholars who are not well versed in the area of concurrency theory, or who are not experts in both Petri nets and process algebras.

The final chapter describes side issues, related to possible generalizations or variations of the theory presented in this book. In particular, we hint how to extend our technique to calculi such as CCS, where the restriction operator is not used at the top level only, but can even occur inside the body of recursively defined constants.

1.4 Interleaving vs True Concurrency

Traditionally, the semantics of process algebras have been given in interleaving style, usually by means of action labeled transition systems. There are some philosophical and practical reasons for doing so, but we argue that none of them is really convincing.

Interaction vs structure: It is often assumed that the only relevant part of the behavior of a system is given by its *interaction capabilities*, i.e., the actions the system performs, with which an observer can interact. This argument is based on the assumption that a system, like a vending machine, is actually a *black box*, and that the observer of the system can only interact with this system by means of the externally visible buttons or slots, but without any knowledge of the internal structure of the machine. This idea is quite appealing and has an obvious consequence that an interleaving model, showing which visible actions can be performed and when, is more than enough to completely describe the behavior of a system. However, by modeling a reactive, distributed system with a transition system, we are forced to make one fundamental abstraction on its structure: the states of the reactive system are monolithic entities that cannot be inspected and each transition from, say, state q to state q', with label a, transforms the state q as a whole, even if only some components of the system are actually involved in the execution of action a. As the structure of the system is abstracted away in interleaving semantics, distributed systems with very different structures and very different properties can be equated. For instance, a deterministic, symmetric, fully distributed, deadlock-free solution to the well-known dining philosophers problem (originally proposed by Dijkstra [Dij71], and then elaborated on by Hoare [Hoa85] in its current formulation) can be provided in FNM (see Chapter 6 of [GV15]). Let p be the FNM solution to this problem; its interleaving semantics is a finite-state labeled transition system, which is also the semantics of an SFM process q, i.e., of a purely sequential process; since p and q are interleaving equivalent, we may wrongly conclude that q is also a deterministic, symmetric, fully distributed, deadlock-free solution to the dining philosophers problem! Of course, this is not the case. Since the properties of interest for a distributed system are often related to the structure of the system, the semantics cannot abstract away from this aspect of the system. *Truly concurrent* semantics, such as step semantics or net semantics, may often offer more adequate behavioral equivalences, when properties of distributed systems are to be investigated.

Compositionality: Most behavioral equivalences over labeled transition systems are congruences w.r.t. the operators of many process algebras, notably CCS, so that compositional reasoning and compositional analysis can be performed profitably and easily. However, we will show that interleaving semantics is too abstract semantics to be a congruence for the parallel composition operators of FNM and NPL. These languages afford multi-party communication, a linguistic mechanism that can be modeled in a compositional way only by means of a truly concurrent semantics. Hence, if compositionality is a requisite of the semantics for FNM and NPL, then a truly concurrent model is necessary.

Simplicity of the semantics definition: Another argument in favor of interleaving semantics is the simplicity and elegance of its mathematical formulation. However, the aim of this book is to show, by means of six case studies, that operational net semantics is often very similar and only slightly more complex than transition system semantics. For instance, for SFM these two semantics are exactly the same, while for FNC/FNM/NPL it is only necessary to handle the bound names in a suitable manner in order to get a simple definition of the markings and of the net transitions. Moreover, each operational net semantics is coupled with a corresponding denotational net semantics, which is also not too complex.

1.5 Beyond Turing-Completeness

In the theory of sequential computation, the computable entities are mathematical functions from $\mathbb{N}$ to $\mathbb{N}$, the set of natural numbers. A formalism is *Turing-complete* if it can compute all the functions that can be computed by means of Turing machines [Tur36]. The famous *Church-Turing thesis* states that any function that is algorithmically computable is actually computable by a Turing machine.

In the theory of concurrency and distribution, the situation is a bit different. The problems that are relevant in distributed computing include the Turing-computable functions, but also include other problems that have nothing to do with functions. In Section 3.5, we present a problem in distributed computing, called the *Last Man Standing* problem (LMS, for short, originally introduced in [VBG09]), that can be solved with a *dynamically acyclic*, finite Nonpermissive Petri net (a class of nets which is not Turing-complete), but that cannot be solved in CCS, a well-known Turing-complete language [Mil89, GV15] (see also Section 9.3). Therefore, what is the analog of computable function for concurrency? And what is the analog of Turing-completeness for concurrency? New definitions are necessary. Here we give our own proposal as a possible new foundation for *distributed* computability theory – as a generalization of classic, *sequential* (or Turing) computability theory.

A *process* is a semantic model, up to some behavioral equivalence. For instance, if the chosen models are labeled transition systems and the chosen equivalence is (weak completed) trace equivalence (see Chapter 2 for details), then a process is nothing but a formal language [HMU01]; if, instead, the chosen models are Petri nets and the chosen equivalence is net isomorphism, then a process is a Petri net, up

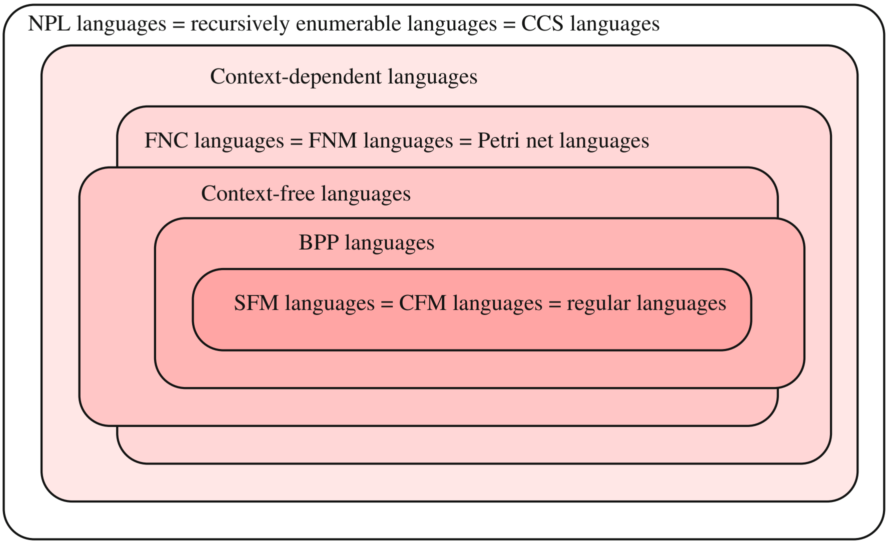

NPL languages = recursively enumerable languages = CCS languages
Context-dependent languages
FNC languages = FNM languages = Petri net languages
Context-free languages
BPP languages
SFM languages = CFM languages = regular languages

Fig. 1.9 The hierarchy of languages — interleaving semantics

to net isomorphism. In other words, the notion of computable function on the natural numbers for sequential computation is to be replaced by a model with an associated equivalence relation, which we call a process, for concurrent computation.

A process algebra is *complete* w.r.t. a given class of models if it can represent all the elements of that class, up to the chosen behavioral equivalence. A class of models is Turing-complete if it can compute all the computable functions; if a process algebra is complete w.r.t. that class of models, then it is also Turing-complete.

Different process algebras usually have different expressive power. In this book we present a list of six, increasingly more and more expressive, process algebras, each one *complete* w.r.t. some class of Petri nets, up to *net isomorphism*, as described in Figure 1.5. The most expressive one is NPL, which is the only language studied in this book to be Turing-complete, as it can represent all finite Nonpermissive Petri nets, and which can also solve the LMS problem. However, there are other process algebras, such as CCS [Mil89, GV15] (only sketched in Section 9.3), that are Turing-complete, but that can neither represent all the finite Nonpermissive Petri nets, nor solve the LMS problem. Nonetheless, CCS can represent many infinite P/T nets, and so NPL and CCS are incomparable, at least if the considered semantic equivalence is net isomorphism.

It is interesting to note that, if the chosen semantic models are labeled transition systems and the chosen behavioral equivalence is (weak completed) trace equivalence, then we remain within the reassuring confines of classic, sequential (or Turing) computability theory, as illustrated in Figure 1.9. NPL, being Turing-complete, models all the recursively enumerable languages, and the same holds for CCS; FNC and FNM model the class of Petri net languages (as shown in Sections 7.2.1 and

7.5.1), and this is included in the class of context-dependent languages; BPP models a class of languages which includes all the regular languages and some context-free languages as well as some context-dependent ones; finally, SFM and CFM model the class of regular languages. Of course, this picture is not very convincing, as we have already observed that NPL and CCS are not equally expressive, as only the former can solve the LMS problem; moreover, FNC and FNM are also not equally expressive, as only the latter can solve the dining philosophers problem in a symmetric, fully distributed, deadlock-free manner (as shown in Chapter 6 of [GV15], based on [LR81, RL94]). In other words, if the observer wears interleaving glasses, then he cannot observe anything more than ordinary formal languages or computable functions; on the contrary, if the observer wears truly concurrent glasses, then he can observe phenomena that go beyond the limits of Turing-completeness.

Chapter 2
Labeled Transition Systems

Abstract Labeled transition systems are presented as a suitable interleaving semantic model for distributed systems. Some notions of behavioral equivalence are discussed, such as isomorphism equivalence, trace equivalence, and bisimulation equivalence. The non-interleaving model of step transition systems is also introduced.

2.1 Labeled Transition Systems

A simple model for reactive, distributed systems is based on (possibly infinite) edge-labeled directed graphs, where the nodes are the *states* of the system and the edges are the *transitions*, each one labeled with one action, describing how the system evolves upon the occurrence of that action. This class of models, called *Labeled Transition Systems* (LTSs for short), was introduced by Keller in [Kel76].

Definition 2.1. (Labels) Let *Lab* be a countable set of *labels* (or *actions*), ranged over by ℓ, possibly indexed, which includes a special action τ denoting an invisible, internal activity. □

Definition 2.2. (Labeled transition system) A labeled transition system (LTS for short) is a triple $TS = (Q, A, \rightarrow)$ where

- Q is the nonempty, countable set of *states*, ranged over by q (possibly indexed);
- $A \subseteq Lab$ is the countable set of *labels* (or *actions*), ranged over by ℓ (possibly indexed);
- $\rightarrow \subseteq Q \times A \times Q$ is the *transition relation*.

Given a transition $(q, \ell, q') \in \rightarrow$, q is called the *source*, q' the *target* and ℓ the *label* of the transition. For economy's sake, we assume that for any $\ell \in A$ there exists a transition $(q, \ell, q') \in \rightarrow$.

A *rooted* labeled transition system is a pair (TS, q_0) where $TS = (Q, A, \rightarrow)$ is an LTS and $q_0 \in Q$ is the *initial state* (or *root*). Sometimes we write $TS = (Q, A, \rightarrow, q_0)$ for a rooted LTS. □

R. Gorrieri, *Process Algebras for Petri Nets*, Monographs in Theoretical
Computer Science. An EATCS Series, DOI 10.1007/978-3-319-55559-1_2

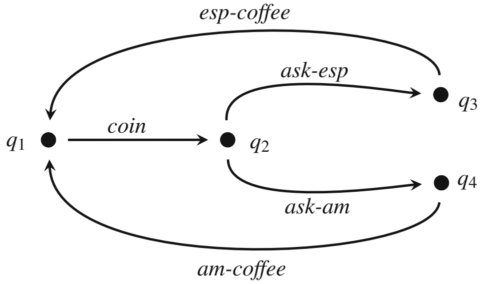


Fig. 2.1 A simple model for a coffee-vending machine

Notation: In the following, given an LTS $(Q,A,\rightarrow)$, a transition $(q,\ell,q') \in \rightarrow$ is denoted by $q \xrightarrow{\ell} q'$. Moreover, we also use the abbreviations below:

$$\begin{array}{lll} q \overset{\ell}{\rightarrow} & \text{if and only if} & \exists q'.\, q \xrightarrow{\ell} q', \\ q \overset{\ell}{\nrightarrow} & \text{if and only if} & \nexists q'.\, q \xrightarrow{\ell} q', \\ q \rightarrow & \text{if and only if} & \exists \ell.\, q \overset{\ell}{\rightarrow}, \\ q \nrightarrow & \text{if and only if} & \forall \ell\; q \overset{\ell}{\nrightarrow}. \end{array}$$

Example 2.1. A simple coffee-vending machine — which can input one coin, allows for the selection of espresso (action *ask-esp*) or American coffee (action *ask-am*), and then delivers the chosen beverage — may be modeled by the LTS (rooted in q_1) depicted in Figure 2.1, with the obvious graphical convention. Formally, the set Q of states is $\{q_1,q_2,q_3,q_4\}$, the set A of labels is $\{$*coin*, *ask-esp*, *ask-am*, *esp-coffee*, *am-coffee*$\}$ and the transition relation $\rightarrow$ is the set $\{(q_1,coin,q_2),(q_2,ask\text{-}esp,q_3),$ $(q_2,ask\text{-}am,q_4),$ $(q_3,esp\text{-}coffee,q_1),$ $(q_4,am\text{-}coffee,q_1)\}$. □

Definition 2.3. (Path) Given an LTS $TS = (Q,A,\rightarrow)$, and two states $q,q' \in Q$, a *path* of length n from q to q' is a sequence of n transitions $q_1 \xrightarrow{\ell_1} q_1'$ $q_2 \xrightarrow{\ell_2} q_2'$ $\ldots$ $q_n \xrightarrow{\ell_n} q_n'$ such that $q = q_1$, $q' = q_n'$ and $q_i' = q_{i+1}$ for $i = 1,\ldots,n$, usually denoted by

$$q_1 \xrightarrow{\ell_1} q_2 \xrightarrow{\ell_2} \ldots q_n \xrightarrow{\ell_n} q_{n+1}.$$

When $n = 0$, the path is *empty* and $q = q' = q_1$. If $q_i \neq q_j$ for all $i \neq j$ $(i,j \in \{1,\ldots,n+1\})$, then the path is *acyclic*, otherwise it is *cyclic*. The rooted LTS (TS,q_0) is *acyclic* if it contains no cyclic path starting from q_0. The LTS TS is *acyclic* if it contains no cyclic path. A path may also be infinite: the infinite sequence $q_1,q_2,q_3,\ldots$, such that $q_i \xrightarrow{\ell_i} q_{i+1}$ for each $i \in \mathbb{N}$, yields the infinite path $q_1 \xrightarrow{\ell_1} q_2 \xrightarrow{\ell_2} q_3 \ldots$. □

Definition 2.4. (Reachability relation) Let A^*, ranged over by σ, be the set of all the strings on A, including the empty string ε. The concatenation of σ_1 and σ_2

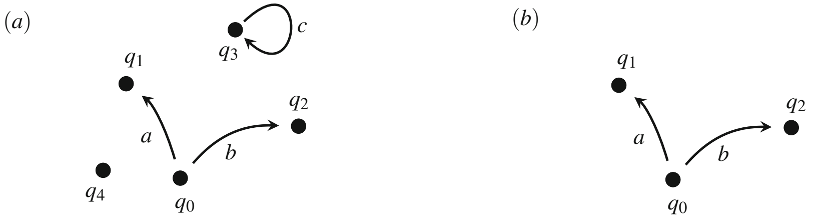


Fig. 2.2 Reachable LTS

yields $\sigma_1\sigma_2$, with the proviso that $\varepsilon\sigma = \sigma = \sigma\varepsilon$. We define the *reachability relation* $\rightarrow^* \subseteq Q \times A^* \times Q$ as the least relation induced by the following axiom and rule:

$$\frac{}{q \xrightarrow{\varepsilon}{}^{*} q} \qquad\qquad \frac{q_1 \xrightarrow{\sigma}{}^{*} q_2 \qquad q_2 \xrightarrow{\ell} q_3}{q_1 \xrightarrow{\sigma\ell}{}^{*} q_3}$$

We simply write $q_1 \rightarrow^* q_2$ to state that q_2 is *reachable* from q_1 when there exists a string σ such that $q_1 \xrightarrow{\sigma}{}^{*} q_2$. □

Definition 2.5. Given an LTS $TS = (Q, A, \rightarrow)$ and a state $q \in Q$, we define the sort of q as the set $sort(q) = \{\ell \in A \mid \exists q'.q \rightarrow^* q' \xrightarrow{\ell}\}$. We define the rooted LTS $TS_q = (Q_q, sort(q), \rightarrow_q, q)$, called the *reachable LTS from q*, where

- Q_q is the set of the states reachable from q, i.e., $Q_q = \{q' \in Q \mid q \rightarrow^* q'\}$, and
- $\rightarrow_q$ is the restriction of $\rightarrow$ on $Q_q \times sort(q) \times Q_q$. □

Let us consider Figure 2.2. It is not difficult to see that the reachable LTS from state q_0 of the LTS (a) on the left is indeed the LTS (b) on the right.

Definition 2.6. A rooted LTS $TS = (Q, A, \rightarrow, q_0)$ is *reduced* if TS is exactly the reachable LTS from q_0, i.e., $TS = TS_{q_0}$. □

The LTS in Figure 2.2(a) is not reduced, while the one in (b) is reduced. Note that a rooted LTS $TS = (Q, A, \rightarrow, q_0)$ is reduced when all the states are reachable from the initial state (i.e., when $Q = Q_{q_0}$).

We briefly introduce some classes of LTSs we will use in the following.

Definition 2.7. (Classes of LTSs) An LTS $TS = (Q, A, \rightarrow)$ is

- *finite* if it is acyclic and Q and A are finite sets;
- *finite-state* if Q and A are finite sets;
- *boundedly branching* if $\exists k \in \mathbb{N}$ such that $\forall q \in Q$ the set $T_q = \{(q, \ell, q') \mid \exists \ell \in A\ \exists q' \in Q.\ \ q \xrightarrow{\ell} q'\}$ has cardinality at most k; the least k satisfying the above condition is called the *branching degree* of the LTS;

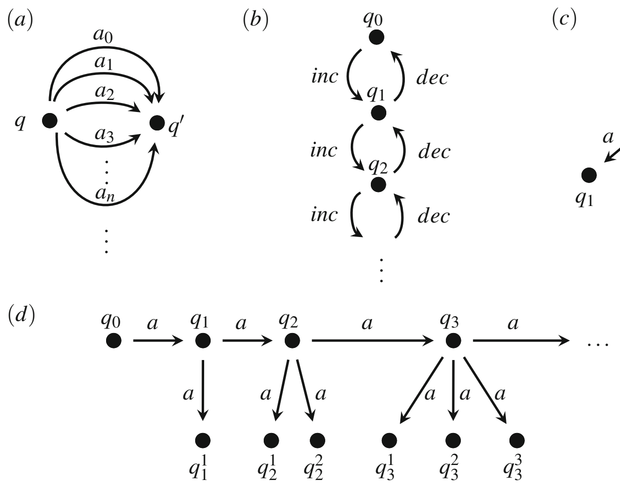


Fig. 2.3 Some labeled transition systems

- *finitely branching* if the set $T_q = \{(q,\ell,q') \mid \exists \ell \in A\ \exists q' \in Q.\ q \xrightarrow{\ell} q'\}$ is finite for all $q \in Q$; if this is not the case, the LTS is *infinitely branching*;
- *image-finite* if the set $T_{q,\ell} = \{(q,\ell,q') \mid \exists q' \in Q.\ q \xrightarrow{\ell} q'\}$ is finite for all $q \in Q$ and for all $\ell \in A$;
- *deterministic* if $q \xrightarrow{\ell} q'$ and $q \xrightarrow{\ell} q''$ imply that $q' = q''$, for all $q \in Q$ and for all $\ell \in A$. □

The LTS in Figure 2.2(*b*) is finite. Note that a finite LTS may offer only finitely many different paths. Of course, any finite LTS is also finite-state. However, the LTS in Figure 2.2(*a*) is finite-state but not finite, because it is not acyclic. Note that a finite-state LTS may offer infinitely many different paths. Note also that there are necessarily finitely many transitions when both Q and A are finite.

In Figure 2.3(*a*), an LTS is depicted that has finitely many states but that is not finite-state because A is infinite; note that this LTS is not even finitely branching. Formally, this LTS is the triple $(Q, A, \rightarrow)$, where $Q = \{q, q'\}$, $A = \{a_i \mid i \in \mathbb{N}\}$ and $\rightarrow = \{(q, a_i, q') \mid a_i \in A\}$.

A finite-state LTS is boundedly branching, while the converse does not hold: Figure 2.3(*b*) depicts a boundedly branching LTS (with branching degree $k = 2$), which is not finite-state; formally, it is the triple $(Q, A, \rightarrow)$, where $Q = \{q_i \mid i \in \mathbb{N}\}$, $A = \{inc, dec\}$ and $\rightarrow = \{(q_i, inc, q_{i+1}) \mid i \in \mathbb{N}\} \cup \{(q_{i+1}, dec, q_i) \mid i \in \mathbb{N}\}$. This LTS represents a semi-counter (i.e., a counter without the ability to test for zero) that can increment (action *inc*) or decrement (action *dec*) the value n stored in the counter state q_n (see also Example 5.9).

A boundedly branching LTS is finitely branching, while the converse is false; Figure 2.3(*d*) depicts an LTS where $Q = \{q_i \mid i \in \mathbb{N}\} \cup \{q_{i+1}^j \mid i \in \mathbb{N}, 1 \leq j \leq i+1\}$, $A = \{a\}$ and $\rightarrow = \{(q_i, a, q_{i+1}) \mid i \in \mathbb{N}\} \cup \{(q_{i+1}, a, q_{i+1}^j) \mid i \in \mathbb{N}, 1 \leq j \leq i+1\}$.

Clearly, for all $i \in \mathbb{N}$, exactly $i+1$ a-labeled transitions start from state q_i, hence, this LTS is not boundedly branching, even if it is finitely branching.

Of course, a finitely branching LTS with finitely many states is boundedly branching. Moreover, an LTS that is not boundedly branching but that is finitely branching cannot be finite-state. Finally, a finitely branching LTS is image-finite; the converse does not hold, as the LTS in Figure 2.3(a) is image-finite but not finitely branching. The LTS in Figure 2.3(c) is not image-finite. Examples of nondeterministic LTSs are shown in Figure 2.3(c) and (d).

2.2 Behavioral Equivalences

When are two systems to be considered equivalent? There is not an obvious answer to this question: many different notions of behavioral equivalence have been proposed in the literature on LTSs, some of which are motivated by peculiar technical reasons. Here we briefly present only the main ones. The reader interested in a more detailed, didactical presentation of this topic may consult [GV15], while an even more comprehensive overview may be found in [Gla01, Gla93, San12].

2.2.1 Strong Equivalences

In this section, we consider the internal action τ as observable as any other action. Equivalences of this kind are called *strong* to reflect this strict requirement. In the next section we will discuss the problem of abstracting from the internal action τ. Equivalences discussed there are called *weak* as they turn out to be less discriminating than the corresponding strong ones, discussed in this section.

As LTSs are basically edge-labeled directed graphs (even if with possibly infinitely many states), we can consider graph isomorphism as our first choice.

Definition 2.8. (Isomorphism) Let $TS_1 = (Q_1, A, \rightarrow_1)$ and $TS_2 = (Q_2, A, \rightarrow_2)$ be two LTSs. An *isomorphism* is a bijection $f : Q_1 \rightarrow Q_2$ which preserves transitions:

$$q \xrightarrow{\ell}_1 q' \quad \text{if and only if} \quad f(q) \xrightarrow{\ell}_2 f(q')$$

for all $q, q' \in Q_1$ and for all $\ell \in A$. If there exists an isomorphism between TS_1 and TS_2, then we say that TS_1 and TS_2 are *isomorphic*, denoted $TS_1 \cong TS_2$.

This definition can be adapted to rooted labeled transition systems by requiring that the isomorphism f preserve also the initial states, i.e., $f(q_1) = q_2$, if q_1 and q_2 are the initial states of TS_1 and TS_2, respectively. □

Proposition 2.1. *LTS isomorphism $\cong$ is an equivalence relation, i.e., reflexive, symmetric and transitive.* □

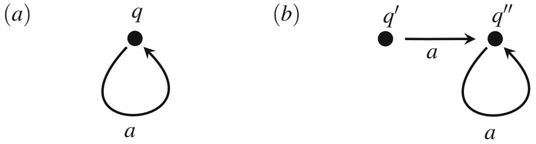


Fig. 2.4 Two equivalent, yet not isomorphic LTSs

Two isomorphic LTSs are of course indistinguishable by any observer: whatever is done on the first can be replicated on the second, and vice versa. However, isomorphism equivalence is too discriminating as it distinguishes between systems that are to be equated intuitively. For instance, consider the two LTSs in Figure 2.4, which are clearly not isomorphic because there is no bijection between the two sets of states. Nonetheless, no observer can tell them apart as, after all, both can only do any sequence of a's. Moreover, checking graph isomorphism over finite-state LTSs is an NP problem that is not known to be in P, and the best known algorithm is exponential in the number of states [BKL83]; therefore, verification based on isomorphism equivalence is, in general, not viable in practice.

Labeled transition systems are also very similar to automata [HMU01], and so we can take inspiration from the notion of equivalence defined over automata. An automaton, besides an initial state, has a designated set of *accepting* (or *final*) states. A sequence of symbols (a *string* or *word*, in automata terminology) is recognized if there is a path in the automaton starting from the initial state and ending in one of its final states by reading that string. Two automata are equivalent if they recognize the same strings (*language equivalence*). Intuitively, LTSs differ in one important aspect from automata: while we check whether string σ can drive the automaton to a final state, for LTSs we check whether it is able to perform that string; hence, if the LTS performs a string σ, necessarily it is also able to perform any prefix of σ; this implies that we should consider an analogous definition of equivalence over LTSs, which implicitly assumes that all the states are final.

Definition 2.9. (Trace equivalence) Let $(Q, A, \rightarrow)$ be an LTS and let $q \in Q$. A *trace* of q is a string $\sigma \in A^*$ such that $q \xrightarrow{\sigma}{}^* q'$ for some $q' \in Q$. The set of *traces* of q is

$$Tr(q) = \{\sigma \in A^* \mid \exists q' \in Q.\, q \xrightarrow{\sigma}{}^* q'\}.$$

Two states $q_1, q_2 \in Q$ are *trace equivalent* if $Tr(q_1) = Tr(q_2)$, and this is denoted by $q_1 =_{tr} q_2$. This definition can be extended to rooted LTSs as follows. The set $Tr(TS)$ of traces of the rooted LTS $TS = (Q, A, \rightarrow, q_0)$ is $Tr(q_0)$. Two rooted LTSs, TS_1 and TS_2, are *trace equivalent* if $Tr(TS_1) = Tr(TS_2)$. □

Is trace equivalence useful for reactive systems? Compare the impolite vending machine in Figure 2.5 with the intuitively correct model in Figure 2.1. It is not difficult to see that the two are trace equivalent, as both offer the same traces. However, we cannot declare them equivalent, as an observer can really detect some difference in their behavior. Observe that in the model of Figure 2.1, after inserting a *coin*, the

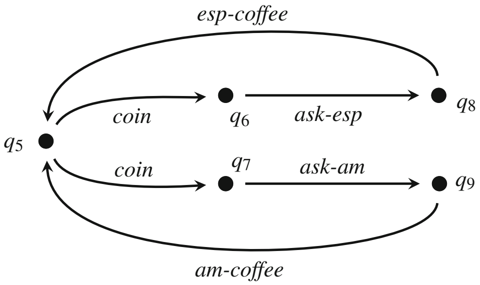


Fig. 2.5 An impolite coffee-vending machine

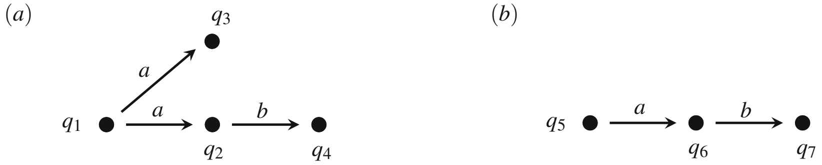


Fig. 2.6 Two trace equivalent LTSs with different deadlock behavior

user can choose between asking for an espresso or for an American coffee. On the contrary, in Figure 2.5, upon insertion of the coin, the machine nondeterministically chooses to reach either q_6, where only *ask-esp* is possible, or q_7, where only *ask-am* is possible; hence, in this case it is the machine that makes the choice and not the user! This example explains that in concurrency theory the timing of a choice can be a crucial aspect of the behavior and cannot be neglected, as in trace equivalence.

Another good reason for rejecting trace equivalence is that it equates LTSs with different deadlock behavior.

Definition 2.10. (Deadlock) A state q is a *deadlock* if there is no transition starting from it, i.e., $\nexists(\ell, q')$ such that $q \xrightarrow{\ell} q'$, usually abbreviated as $q \not\to$. An LTS $TS = (Q, A, \to)$ is *deadlock-free* if for all $q \in Q$, q is not a deadlock. □

Indeed, trace equivalence is not sensitive to deadlock. Consider the two LTSs in Figure 2.6. Both can perform the same set of traces, namely $\{\varepsilon, a, ab\}$, so they are trace equivalent. However, the LTS on the left is nondeterministic and, after a, it can reach the deadlock q_3, while q_5, after a, reaches q_6, which is not a deadlock. We can slightly refine trace equivalence in order to get an equivalence that is sensitive to deadlock.

Definition 2.11. (Completed trace equivalence) Let $TS = (Q, A, \to)$ be an LTS. A *completed trace* for $q \in Q$ is a (possibly empty) string $\sigma \in A^*$ such that $q \xrightarrow{\sigma}{}^* q'$ and $q' \not\to$ for some $q' \in Q$. Hence, the set of *completed traces* of a state $q \in Q$ is

$$CTr(q) = \{\sigma \in A^* \mid \exists q' \in Q.\, q \xrightarrow{\sigma}{}^* q' \wedge q' \not\to\}.$$

Two states, $q_1, q_2 \in Q$, are completed trace equivalent if $Tr(q_1) = Tr(q_2)$ and $CTr(q_1) = CTr(q_2)$, and this is denoted by $q_1 =_{ctr} q_2$. □

The two states q_1 and q_5 in Figure 2.6 are not completed trace equivalent, as the set of the completed traces of q_1 is $\{a, ab\}$, while for q_5 it is $\{ab\}$.

Even if sensitive to deadlock, is completed trace equivalence satisfactory? Unfortunately, this is not the case: the two vending machines of Figures 2.1 and 2.5, which we expect to be not equivalent, are completed trace equivalent: as these two LTSs are deadlock-free, it turns out that $CTr(q_1) = CTr(q_5) = \emptyset$. So it is necessary to find some finer notion of equivalence that is able to capture the timing of choices. Moreover, language equivalence over automata is PSPACE-complete [SM73, HRS76], and so is also trace equivalence (as well as completed trace equivalence) for finite-state LTSs. It is therefore advisable to find a finer notion of equivalence that is more easily checkable. The natural candidate equivalence relation is bisimulation equivalence, originated in [Park81, Mil89].

Definition 2.12. (Simulation and bisimulation) Let $TS = (Q, A, \rightarrow)$ be an LTS. A *simulation* is a relation $R \subseteq Q \times Q$ such that if $(q_1, q_2) \in R$ then for all $\ell \in A$

- $\forall q_1'$ such that $q_1 \xrightarrow{\ell} q_1'$, $\exists q_2'$ such that $q_2 \xrightarrow{\ell} q_2'$ and $(q_1', q_2') \in R$.

State q is simulated by q', denoted $q \lesssim q'$, if there exists a simulation R such that $(q, q') \in R$. Two states q and q' are *simulation equivalent*, denoted $q \simeq q'$, if $q \lesssim q'$ and $q' \lesssim q$.

A *bisimulation* is a relation $R \subseteq Q \times Q$ such that R and its inverse R^{-1} are both simulation relations. More explicitly, a bisimulation is a relation R such that if $(q_1, q_2) \in R$ then for all $\ell \in A$

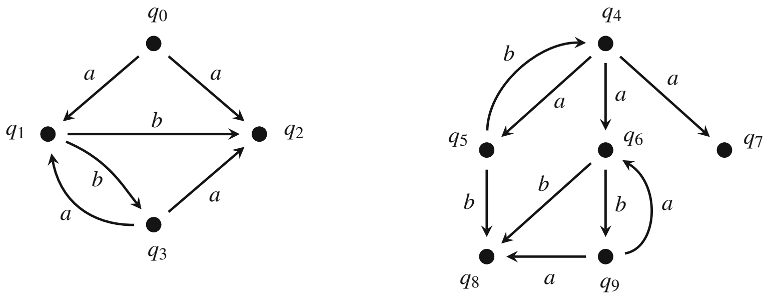
- $\forall q_1'$ such that $q_1 \xrightarrow{\ell} q_1'$, $\exists q_2'$ such that $q_2 \xrightarrow{\ell} q_2'$ and $(q_1', q_2') \in R$,
- $\forall q_2'$ such that $q_2 \xrightarrow{\ell} q_2'$, $\exists q_1'$ such that $q_1 \xrightarrow{\ell} q_1'$ and $(q_1', q_2') \in R$.

Two states q and q' are *bisimilar* (or *bisimulation equivalent*), denoted $q \sim q'$, if there exists a bisimulation R such that $(q, q') \in R$. □

Remark 2.1. The definition above covers also the case of a bisimulation between two LTSs, say, $TS_1 = (Q_1, A, \rightarrow_1)$ and $TS_2 = (Q_2, A, \rightarrow_2)$ with $Q_1 \cap Q_2 = \emptyset$, because we may consider just one single LTS $TS = (Q_1 \cup Q_2, A, \rightarrow_1 \cup \rightarrow_2)$: A bisimulation $R \subseteq Q_1 \times Q_2$ is also a bisimulation on $(Q_1 \cup Q_2) \times (Q_1 \cup Q_2)$. We say that a rooted LTS $TS_1 = (Q_1, A, \rightarrow_1, q_1)$ is bisimilar to the rooted LTS $TS_2 = (Q_2, A, \rightarrow_2, q_2)$ if there exists a bisimulation $R \subseteq Q_1 \times Q_2$ containing the pair (q_1, q_2). □

Example 2.2. Consider the LTSs in Figure 2.7. A bisimulation relation proving that $q_0 \sim q_4$ is $R = \{(q_0, q_4), (q_1, q_5), (q_2, q_7), (q_1, q_6), (q_2, q_8), (q_3, q_4), (q_3, q_9)\}$. □

Let us consider again the two vending machines of Figure 2.1 and Figure 2.5. We can prove that it is not possible to find a bisimulation R containing the pair (q_1, q_5). Suppose, towards a contradiction, we have a bisimulation R such that



(a) (b)

Fig. 2.7 Two bisimulation equivalent systems

$(q_1, q_5) \in R$. By definition of bisimulation, to transition $q_1 \xrightarrow{coin} q_2$, q_5 has to respond with $q_5 \xrightarrow{coin} q_6$ or with $q_5 \xrightarrow{coin} q_7$; this means that at least one of the two pairs (q_2, q_6) or (q_2, q_7) must belong to bisimulation R. But this is impossible: if $(q_2, q_6) \in R$, then q_6 cannot respond to transition $q_2 \xrightarrow{ask\text{-}am} q_4$, invalidating the assumption that R is a bisimulation; similarly, if $(q_2, q_7) \in R$, then q_7 cannot respond to transition $q_2 \xrightarrow{ask\text{-}esp} q_3$. Therefore, the two vending machines are not bisimulation equivalent.

We now list some useful properties of bisimulation relations.

Proposition 2.2. *For any LTS $TS = (Q, A, \rightarrow)$, the following hold:*

1. *the identity relation $\mathscr{I} = \{(q,q) \mid q \in Q\}$ is a bisimulation;*
2. *the inverse relation $R^{-1} = \{(q', q) \mid (q, q') \in R\}$ of a bisimulation R is a bisimulation;*
3. *the relational composition $R_1 \circ R_2 = \{(q, q'') \mid \exists q'.(q, q') \in R_1 \wedge (q', q'') \in R_2\}$ of two bisimulations R_1 and R_2 is a bisimulation;*
4. *the union $\bigcup_{i \in I} R_i$ of bisimulations R_i is a bisimulation.*

Remember that $q \sim q'$ if there exists a bisimulation containing the pair (q, q'). This means that $\sim$ is the union of all bisimulations, i.e.,

$$\sim = \bigcup \{R \subseteq Q \times Q \mid R \text{ is a bisimulation}\}.$$

By Proposition 2.2(4), $\sim$ is also a bisimulation, hence the largest such relation.

Proposition 2.3. *For any LTS $TS = (Q, A, \rightarrow)$, relation $\sim \subseteq Q \times Q$ is the largest bisimulation relation.* □

Observe that the bisimulation relation we have presented in Example 2.2 is not reflexive, not symmetric, and not transitive. Nonetheless, the largest bisimulation relation $\sim$ is an equivalence relation. As a matter of fact, as the identity relation $\mathscr{I}$ is a bisimulation by Proposition 2.2(1), we have that $\mathscr{I} \subseteq \sim$, and so $\sim$ is reflexive. Symmetry derives from the following argument. For any $(q,q') \in \sim$, there exists a bisimulation R such that $(q,q') \in R$; by Proposition 2.2(2), relation R^{-1} is a bisimulation containing the pair (q',q); hence, $(q',q) \in \sim$ because $R^{-1} \subseteq \sim$. Transitivity also holds for $\sim$. Assume $(q,q') \in \sim$ and $(q',q'') \in \sim$; hence, there exist two bisimulations R_1 and R_2 such that $(q,q') \in R_1$ and $(q',q'') \in R_2$; by Proposition 2.2(3), relation $R_1 \circ R_2$ is a bisimulation containing the pair (q,q''); hence, $(q,q'') \in \sim$, because $R_1 \circ R_2 \subseteq \sim$. Summing up, we have the following.

Proposition 2.4. *For any LTS $TS = (Q,A,\rightarrow)$, relation $\sim \subseteq Q \times Q$ is an equivalence relation.* □

Bisimulation equivalence over a finite-state LTS with n states and m transitions can be computed in $O(m \log n)$ time [PT87]. Differently from all the other equivalences, it is even decidable over one class of infinite-state systems we will introduce in the following (notably over BPP in Chapter 5).

It is sometimes convenient to write a bisimulation compactly, by removing those pairs that differ from others only up to the use of bisimulation equivalent alternatives. The resulting relation is *not* a bisimulation, rather a bisimulation up to $\sim$. We denote by $\sim R \sim$ the relational composition $\sim \circ R \circ \sim$; in other words, by $q \sim R \sim q'$ we mean that there exist two states q_1 and q_2 such that $q \sim q_1$, $(q_1,q_2) \in R$ and $q_2 \sim q'$.

Definition 2.13. (Bisimulation up to $\sim$) A bisimulation up to $\sim$ is a binary relation R on Q such that if $(q_1,q_2) \in R$ then for all $\ell \in A$

- $\forall q_1'$ such that $q_1 \xrightarrow{\ell} q_1'$, $\exists q_2'$ such that $q_2 \xrightarrow{\ell} q_2'$ and $q_1' \sim R \sim q_2'$,
- $\forall q_2'$ such that $q_2 \xrightarrow{\ell} q_2'$, $\exists q_1'$ such that $q_1 \xrightarrow{\ell} q_1'$ and $q_1' \sim R \sim q_2'$. □

Lemma 2.1. *If R is a bisimulation up to $\sim$, then $\sim R \sim$ is a bisimulation.*

Proof. *Assume $q \sim R \sim q'$, i.e., there exist q_1 and q_2 such that $q \sim q_1$, $(q_1,q_2) \in R$ and $q_2 \sim q'$. We have to prove that for any $q \xrightarrow{\ell} q^1$ there exists $q' \xrightarrow{\ell} q^2$ such that $q^1 \sim R \sim q^2$ (the symmetric case when q' moves first is omitted). Since $q \sim q_1$, there exists q_1' such that $q_1 \xrightarrow{\ell} q_1'$ with $q^1 \sim q_1'$. Since $(q_1,q_2) \in R$, there exists q_2' such that $q_2 \xrightarrow{\ell} q_2'$ with $q_1' \sim R \sim q_2'$. Since $q_2 \sim q'$, there exists q^2 such that $q' \xrightarrow{\ell} q^2$ with $q_2' \sim q^2$. Summing up, $q^1 \sim q_1'$ and $q_1' \sim R \sim q_2'$ and $q_2' \sim q^2$ can be shortened to $q^1 \sim R \sim q^2$, because $\sim \circ \sim = \sim$ by Proposition 2.2. Hence, we have proved that if $q \sim R \sim q'$ then for any q^1 such that $q \xrightarrow{\ell} q^1$ there exists q^2 such that $q' \xrightarrow{\ell} q^2$ with $q^1 \sim R \sim q^2$, as required by the definition of bisimulation.* □

Proposition 2.5. *If R is a bisimulation up to $\sim$, then $R \subseteq \sim$.*

Proof. *By Lemma 2.1, $\sim R \sim$ is a bisimulation, hence $\sim R \sim \subseteq \sim$ by definition of $\sim$. As the identity relation $\mathscr{I} \subseteq \sim$ by Proposition 2.2(1), we have that relation $R = \mathscr{I} \circ R \circ \mathscr{I} \subseteq \sim R \sim$, hence $R \subseteq \sim$ by transitivity.* □

The above proposition states the correctness of the proof principle based on the above up-to technique: the fact that $R \subseteq \sim$ ensures that no erroneous equalities are introduced. Hence, in order to prove that $p \sim q$, it is enough to exhibit a bisimulation up to $\sim$ that contains the pair (p,q). The up-to technique is very useful when we consider LTSs generated by process algebra terms, as we will see in the next chapters. In that setting, the states are terms, and on terms many algebraic properties hold (e.g., associativity and commutativity of parallel composition) for bisimulation equivalence; hence, we can economize on the number of pairs in R by replacing one process algebra term by some other equivalent one, according to some algebraic laws.

On process algebra terms, it is also possible to define a *structural congruence* relation $\equiv$. To be more precise, given a set of axioms E (typically, by assuming the parallel operator to be associative and commutative), the structural congruence $\equiv$ on terms is the congruence induced by the axioms in E. In other words, $p \equiv q$ if and only if p can be proved equal to q by means of an equational deductive proof, using the axioms in E (see, e.g., [GV15], Section 4.3.1, for a brief introduction to equational deduction). In Chapters 7 and 8, a structural congruence $\equiv$ will be useful in properly defining the semantics of FNM and NPL, respectively. If $\equiv \subseteq \sim$, then also the following proof technique is available.

Definition 2.14. (Bisimulation up to $\equiv$) A bisimulation up to $\equiv$ is a binary relation R on Q such that if $(q_1,q_2) \in R$ then for all $\ell \in A$

- $\forall q_1'.\ q_1 \xrightarrow{\ell} q_1'$, $\exists q_2'$ such that $q_2 \xrightarrow{\ell} q_2'$ and $q_1' \equiv R \equiv q_2'$,
- $\forall q_2'.\ q_2 \xrightarrow{\ell} q_2'$, $\exists q_1'$ such that $q_1 \xrightarrow{\ell} q_1'$ and $q_1' \equiv R \equiv q_2'$. □

Proposition 2.6. *If R is a bisimulation up to $\equiv$ and $\equiv \subseteq \sim$, then $R \subseteq \sim$.*

Proof. *It is easy to check that if R is a bisimulation up to $\equiv$, then $\equiv R \equiv$ is a bisimulation. Hence $\equiv R \equiv \subseteq \sim$ by definition of $\sim$. As the identity relation $\mathscr{I} \subseteq \equiv$, we have that relation $R = \mathscr{I} \circ R \circ \mathscr{I} \subseteq \equiv R \equiv$, hence $R \subseteq \sim$ by transitivity.* □

We conclude this section by making comparisons among the various (strong) equivalences we have introduced.

Given two labeled transition systems $TS_1 = (Q_1, A, \rightarrow_1)$ and $TS_2 = (Q_2, A, \rightarrow_2)$, it is easy to see that if TS_1 and TS_2 are isomorphic via bijection f, then $R = \{(q_1,q_2) \in Q_1 \times Q_2 \mid f(q_1) = q_2\}$ is a bisimulation. Hence, bisimulation equivalence is coarser than isomorphism.

Moreover, bisimulation equivalence is finer than completed trace equivalence. If R is a bisimulation containing the pair (q_1,q_2), then $q_1 =_{ctr} q_2$, i.e., $Tr(q_1) = Tr(q_2)$ and $CTr(q_1) = CTr(q_2)$. As a matter of fact, the bisimulation definition ensures that whatever action ℓ is performed by one of the two, the other can reply with ℓ, reaching a pair of states that are still in the bisimulation relation R. By iterating this single step, one can prove that whatever sequence σ is performed by one of the two, say $q_1 \xrightarrow{\sigma}{}^* q_1'$, the very same sequence can be performed also by the other, $q_2 \xrightarrow{\sigma}{}^* q_2'$, reaching a pair of states (q_1',q_2') that are still in R; hence, if σ is a trace of q_1, then σ is also a trace of q_2. If the reached state q_1' is a deadlock, then also the other one, say

q_2', must be a deadlock, in order to satisfy the bisimulation condition; hence, if σ is a completed trace for q_1, then σ is also a completed trace for q_2. Symmetrically, if σ is a completed trace for q_2, then σ is also a completed trace for q_1.

To conclude the comparison with the various equivalence relations we have presented, note that bisimulation equivalence is finer than simulation equivalence: $q_1 \sim q_2$ if there exists a relation S, with $(q_1, q_2) \in S$, such that S and its inverse S^{-1} are both simulations; instead, $q_1 \simeq q_2$ holds if $q_1 \lesssim q_2$ and $q_2 \lesssim q_1$, i.e., if there exist *two* simulations R_1 and R_2 such that $(q_1,q_2) \in R_1$ and $(q_2,q_1) \in R_2$, but R_2 may be not R_1^{-1}. Moreover, simulation equivalence implies trace equivalence, but it is incomparable w.r.t. completed trace equivalence.

In case of *deterministic* LTSs, trace equivalence and bisimulation equivalence coincide, as well as all the other equivalences in between. Hence, only two distinct equivalences are available for deterministic systems: trace equivalence and isomorphism equivalence.

2.2.2 Weak Equivalences

All the equivalences we have discussed so far are usually called *strong* to denote that all the actions of the LTS are equally observable. In fact, in real life this is not the case. A lot of activities of a system are completely internal and cannot be influenced by any interacting user and so, to some extent, are not observable. It is usually assumed that all the internal activities are equally represented by the same action τ, to express that the actual content of what an internal action does is unobservable. For simplicity's sake, here we introduce only some basic weak equivalences. The reader interested in more detail can consult [GV15, Gla93, San12].

Definition 2.15. For any LTS $TS = (Q, A, \to)$, with $B = A \setminus \{\tau\}$, we define relation $\Longrightarrow \ \subseteq Q \times B^* \times Q$ as the least relation induced by the following axiom and rules, where ε is the empty trace:

$$\frac{}{q \overset{\varepsilon}{\Longrightarrow} q} \qquad \frac{q_1 \overset{\sigma}{\Longrightarrow} q_2 \quad q_2 \xrightarrow{\tau} q_3}{q_1 \overset{\sigma}{\Longrightarrow} q_3} \qquad \frac{q_1 \overset{\sigma}{\Longrightarrow} q_2 \quad q_2 \xrightarrow{\ell} q_3 \quad \ell \neq \tau}{q_1 \overset{\sigma\ell}{\Longrightarrow} q_3}$$

□

Note that a path $q_1 \xrightarrow{\tau} q_2 \xrightarrow{\tau} \dots q_n \xrightarrow{\tau} q_{n+1}$ (with $n \geq 0$) yields that $q_1 \overset{\varepsilon}{\Longrightarrow} q_{n+1}$. Moreover, it can be proved that $q \overset{\ell}{\Longrightarrow} q'$ if and only if there exist two states q_1 and q_2 such that $q \overset{\varepsilon}{\Longrightarrow} q_1 \xrightarrow{\ell} q_2 \overset{\varepsilon}{\Longrightarrow} q'$. Finally, if $\sigma = \ell_1 \ell_2 \dots \ell_n$, then $q_1 \overset{\sigma}{\Longrightarrow} q_{n+1}$ if and only if there exist $q_2, \dots, q_n$ such that $q_1 \overset{\ell_1}{\Longrightarrow} q_2 \overset{\ell_2}{\Longrightarrow} \dots q_n \overset{\ell_n}{\Longrightarrow} q_{n+1}$.

Definition 2.16. (Weak trace equivalence) Let $(Q, A, \to)$ be an LTS, and let $B = A \setminus \{\tau\}$. A *weak trace* of $q \in Q$ is a string $\sigma \in B^*$ such that $q \overset{\sigma}{\Longrightarrow} q'$ for some q'. Hence, the set $WTr(q)$ of weak traces of q is

$$WTr(q) = \{\sigma \in B^* \mid \exists q' \in Q.\, q \overset{\sigma}{\Longrightarrow} q'\}.$$

Two states $q_1, q_2 \in Q$ are *weak trace equivalent* if $WTr(q_1) = WTr(q_2)$, and this is denoted by $q_1 =_{wtr} q_2$. This definition can be adapted to rooted LTSs: the set $WTr(TS)$ of weak traces of the rooted LTS $TS = (Q, A, \rightarrow, q_0)$ is $WTr(q_0)$. Two rooted LTSs, TS_1 and TS_2, are *weak trace equivalent* if $WTr(TS_1) = WTr(TS_2)$. □

Following Definition 2.9, given an LTS $TS = (Q, A, \rightarrow)$, a (strong) trace is any $\sigma \in A^*$ such that $q \overset{\sigma}{\longrightarrow}{}^* q'$ for some $q' \in Q$. It is easy to see that two (strong) trace equivalent states are also weak trace equivalent, i.e., $Tr(q_1) = Tr(q_2)$ implies $WTr(q_1) = WTr(q_2)$ for any pair of states q_1 and q_2.

Definition 2.17. (Weak completed traces) Given an LTS $TS = (Q, A, \rightarrow)$, with $B = A \setminus \{\tau\}$, and a state $q \in Q$, the set of *weak completed traces* of q is

$$WCTr(q) = \{\sigma \in B^* \mid \exists q' \in Q.\, q \overset{\sigma}{\Longrightarrow} q' \land q' \overset{\ell}{\not\Longrightarrow} \text{ for all } \ell \in B\}.$$

Note that state q' above need not be a deadlock state, as it may still perform silent, τ-labeled transitions. Two states $q_1, q_2 \in Q$ are *weak completed trace equivalent* if $WTr(q_1) = WTr(q_2)$ and $WCTr(q_1) = WCTr(q_2)$, denoted by $q_1 =_{wctr} q_2$.

This definition can be extended to rooted LTSs: the set $WCTr(TS)$ of weak completed traces of the rooted LTS $TS = (Q, A, \rightarrow, q_0)$ is $WCTr(q_0)$. Two rooted LTSs, TS_1 and TS_2, are weak completed trace equivalent if $WTr(TS_1) = WTr(TS_2)$ and $WCTr(TS_1) = WCTr(TS_2)$. □

For finite-state LTSs, one can compute weak (completed) trace equivalence by means of strong (completed) trace equivalence: one has first to compute the (partial) transitive closure $\Rightarrow' = \{(q, \ell, q') \mid \ell \in B \land (q, \ell, q') \in \Rightarrow\}$ of the transition relation $\rightarrow$ (a procedure, based on the classic Floyd-Warshall algorithm [Flo62], that takes time at most $O(n^3)$ with n the number of states), and then to check (completed) trace equivalence, which is PSPACE-complete.

Remark 2.2. **(All regular languages – and only these – are representable by finite state LTSs)** In automata theory [HMU01], the languages recognized by finite automata are called *regular* languages. We want to show that *all the regular languages* can be represented by finite-state LTSs.

Let L be a regular language. Then, there exists a DFA $M = (Q, B, \delta, F, q_0)$ — where $Q = \{q_0, q_1, \ldots, q_n\}$, and δ has type $\delta : Q \times B \rightarrow Q$ — such that L is the language recognized by M. To be precise, a configuration is a pair (q, w) with $q \in Q$ and $w \in B^*$. Configurations can evolve according to the following rules:

$$\frac{}{(q,w) \longrightarrow^* (q,w)} \qquad \frac{(q,w) \longrightarrow^* (q',aw') \qquad q'' = \delta(q',a)}{(q,w) \longrightarrow^* (q'',w')}$$

A DFA $M = (Q, B, \delta, F, q_0)$ recognizes (or accepts) a string $w \in B^*$ if there exists a final state $q \in F$ such that $(q_0, w) \longrightarrow^* (q, \varepsilon)$, i.e., there is a path starting from the initial state q_0 in the automaton that, by reading w, leads to a final state. The automaton M recognizes the language $L[M] = \{w \in B^* \mid \exists q \in F.(q_0, w) \longrightarrow^* (q, \varepsilon)\}$.

Starting from M, we can build a rooted LTS $TS_M = (Q \cup F', B \cup \{\tau\}, \to, q_0)$, where $F' = \{q'_i \mid q_i \in F\}$ is a set of copies of the final states of M, and the transition relation is defined as the least relation generated by the following rules:

$$\frac{\delta(q_i,a) = q_j}{q_i \xrightarrow{a} q_j} \qquad \frac{\delta(q_i,a) = q_j \wedge q_j \in F}{q_i \xrightarrow{a} q'_j} \qquad \frac{q_0 \in F}{q_0 \xrightarrow{\tau} q'_0}$$

Note that for a transition $\delta(q_i,a) = q_j$ with $q_j \in F$ in M, we have two transitions in TS_M: $q_i \xrightarrow{a} q_j$ and $q_i \xrightarrow{a} q'_j$. Hence, TS_M is a nondeterministic LTS, even if M is a deterministic automaton. Note that the states in F' are deadlocks.

It is not difficult to prove that $(q_0,w) \longrightarrow^* (q_k,\varepsilon)$ in M iff $q_0 \xrightarrow{w}{}^* q_k$ in TS_M, for all $q_k \in Q$ and $w \in B^*$. The proof is by induction on the length of w; the base case is $w = \varepsilon$; in this case, $(q_0,\varepsilon) \longrightarrow^* (q_0,\varepsilon)$ as well as $q_0 \xrightarrow{\varepsilon}{}^* q_0$, as required. Now, assume $w = va$. In M, $(q_0,va) \longrightarrow^* (q_k,\varepsilon)$ is derivable iff there exists a state $q_i \in Q$ such that $(q_0,va) \longrightarrow^* (q_i,a)$ and $\delta(q_i,a) = q_k$. It can easily be proved that $(q_0,va) \longrightarrow^* (q_i,a)$ in M iff $(q_0,v) \longrightarrow^* (q_i,\varepsilon)$; hence, by induction, we can conclude that $q_0 \xrightarrow{v}{}^* q_i$ in TS_M and, by definition of the transition relation, also $q_i \xrightarrow{a} q_k$; therefore, $q_0 \xrightarrow{va}{}^* q_k$ by Definition 2.4, as required.

As a consequence, we can easily prove that $w \in WCTr(TS_M)$ if and only if $w \in L[M]$ for all $w \in B^*$. The empty trace ε belongs to $WCTr(TS_M)$ iff transition $q_0 \xrightarrow{\tau} q'_0$ is present in TS_M; in turn, this is possible iff $q_0 \in F$, and so iff $\varepsilon \in L[M]$. Now, let us assume that $va \in WCTr(TS_M)$. This is possible iff there exist $q_i \in Q$ and $q'_j \in F'$ such that $q_0 \xrightarrow{v}{}^* q_i \xrightarrow{a} q'_j$. By the argument above, we have that $(q_0,v) \longrightarrow^* (q_i,\varepsilon)$; moreover, by definition of $\to$, we have $\delta(q_i,a) = q_j$ with $q_j \in F$. It can easily be proved that $(q_0,v) \longrightarrow^* (q_i,\varepsilon)$ in M iff $(q_0,va) \longrightarrow^* (q_i,a)$; therefore, $(q_0,va) \longrightarrow^* (q_j,\varepsilon)$ in M, i.e., $va \in L[M]$.

Conversely, we can also show that *only regular languages* can be represented by finite-state LTSs. Let us consider a rooted LTS $TS = (Q, B \cup \{\tau\}, \to, q_0)$, with $\tau \notin B$. Starting from TS, we can build an NFA $M = (Q,B,\delta,F,q_0)$, where the set of final states is $F = \{q \in Q \mid q \text{ is a deadlock}\}$, and $\delta \subseteq Q \times (B \cup \{\varepsilon\}) \times Q$ is defined as the least relation generated by the following rules:

$$\frac{q_i \xrightarrow{a} q_j}{(q_i,a,q_j) \in \delta} \qquad \frac{q_i \xrightarrow{\tau} q_j}{(q_i,\varepsilon,q_j) \in \delta}$$

It is easy to prove, with reasoning similar to the above, that $w \in WCTr(TS)$ if and only if $w \in L[M]$ for all $w \in B^*$; this because the two automata-like structures are essentially isomorphic and a state is final for M iff it is a deadlock for TS.

Summing up, we have shown that a language L is regular if and only if there exists a finite-state, rooted LTS TS such that $L = WCTr(TS)$. □

Definition 2.18. (Weak simulation) For any LTS $TS = (Q,A,\to)$, with $B = A \setminus \{\tau\}$, a *weak simulation* is a relation $R \subseteq Q \times Q$ such that if $(q_1,q_2) \in R$ then for all $\ell \in B$

- $\forall q'_1$ such that $q_1 \xrightarrow{\ell} q'_1$, $\exists q'_2$ such that $q_2 \xRightarrow{\ell} q'_2$ and $(q'_1,q'_2) \in R$,
- $\forall q'_1$ such that $q_1 \xrightarrow{\tau} q'_1$, $\exists q'_2$ such that $q_2 \xRightarrow{\varepsilon} q'_2$ and $(q'_1,q'_2) \in R$.

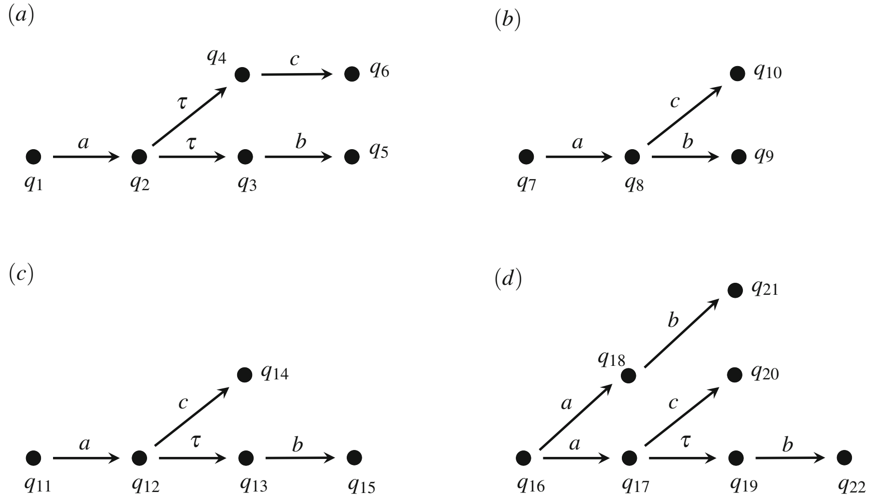


Fig. 2.8 Some weakly simulation equivalent labeled transition systems

State q is *weakly simulated* by q', denoted $q \lessapprox q'$, if there exists a weak simulation R such that $(q,q') \in R$. Two states q and q' are *weakly simulation equivalent*, denoted $q \approxeq q'$, if $q \lessapprox q'$ and $q' \lessapprox q$. □

Hence, the weak simulation preorder $\lessapprox$ is the union of all the weak simulations:

$$\lessapprox = \bigcup \{R \subseteq Q \times Q \mid R \text{ is a weak simulation}\}.$$

It is not difficult to prove that, for any LTS $(Q,A,\rightarrow)$, relation $\lessapprox \subseteq Q \times Q$ is a preorder (i.e., reflexive and transitive), while relation $\approxeq \subseteq Q \times Q$ is an equivalence relation (hence, also symmetric).

Weak similarity $\approxeq$ is incomparable with weak completed trace equivalence $=_{wctr}$, because $\approxeq$ may equate systems with different deadlock behaviors. Moreover, weak simulation equivalence is unable to sense the timing of choices in a complete way, as explained below by considering the LTSs in Figure 2.8. On the one hand, it is easily seen that all four LTSs are weakly simulation equivalent. In particular, relation

$$R = \{(q_1,q_7),(q_2,q_8),(q_3,q_8),(q_4,q_8),(q_5,q_9),(q_6,q_{10})\}$$

is a weak simulation proving that the LTS in Figure 2.8(a) is weakly simulated by the LTS in Figure 2.8(b). On the other hand, by performing a, q_1 reaches q_2, while q_7 reaches q_8; however, it is unreasonable to consider q_2 and q_8 equivalent: q_2 can silently move to q_3, hence discarding action c, while q_8 can reply to this move only by idling, but q_8 is still able to perform c.

A natural strengthening of the weak simulation preorder is weak bisimulation equivalence.

Definition 2.19. (Weak bisimulation) For any labeled transition system $(Q,A,\rightarrow)$, with $B = A \setminus \{\tau\}$, a *weak bisimulation* is a relation $R \subseteq Q \times Q$ such that both R and its inverse R^{-1} are weak simulations. More expicitly, a weak bisimulation is a relation R such that if $(q_1, q_2) \in R$ then for all $\ell \in B$

- $\forall q_1'$ such that $q_1 \xrightarrow{\ell} q_1'$, $\exists q_2'$ such that $q_2 \overset{\ell}{\Longrightarrow} q_2'$ and $(q_1', q_2') \in R$,
- $\forall q_1'$ such that $q_1 \xrightarrow{\tau} q_1'$, $\exists q_2'$ such that $q_2 \overset{\varepsilon}{\Longrightarrow} q_2'$ and $(q_1', q_2') \in R$,

and, symmetrically,

- $\forall q_2'$ such that $q_2 \xrightarrow{\ell} q_2'$, $\exists q_1'$ such that $q_1 \overset{\ell}{\Longrightarrow} q_1'$ and $(q_1', q_2') \in R$,
- $\forall q_2'$ such that $q_2 \xrightarrow{\tau} q_2'$, $\exists q_1'$ such that $q_1 \overset{\varepsilon}{\Longrightarrow} q_1'$ and $(q_1', q_2') \in R$.

States q and q' are *weakly bisimilar* (or *weak bisimulation equivalent*), denoted by $q \approx q'$, if there exists a weak bisimulation R such that $(q, q') \in R$. □

Hence, weak bisimulation equivalence is the union of all weak bisimulations:

$$\approx = \bigcup \{R \subseteq Q \times Q \mid R \text{ is a weak bisimulation}\}.$$

For any LTS $(Q,A,\rightarrow)$, relation $\approx \subseteq Q \times Q$ is an equivalence relation. Moreover, $q \approx q'$ implies $q \approxeq q'$.

Consider again Figure 2.8. To show that the LTSs (c) and (d) are weakly bisimilar, it is enough to exhibit a suitable weak bisimulation, e.g.,

$$R = \{(q_{11}, q_{16}), (q_{12}, q_{17}), (q_{13}, q_{19}), (q_{13}, q_{18}), (q_{14}, q_{20}), (q_{15}, q_{21}), (q_{15}, q_{22})\}.$$

It is possible to offer an alternative, yet equivalent, definition of weak bisimulation as follows: a weak bisimulation is a relation R such that if $(q_1, q_2) \in R$ then for all $\delta \in B \cup \{\varepsilon\}$

- $\forall q_1'$ such that $q_1 \overset{\delta}{\Longrightarrow} q_1'$, $\exists q_2'$ such that $q_2 \overset{\delta}{\Longrightarrow} q_2'$ and $(q_1', q_2') \in R$,
- $\forall q_2'$ such that $q_2 \overset{\delta}{\Longrightarrow} q_2'$, $\exists q_1'$ such that $q_1 \overset{\delta}{\Longrightarrow} q_1'$ and $(q_1', q_2') \in R$,

which is exactly the definition of strong bisimulation on the LTS defined by the transition relation $\Rightarrow'' = \{(q, \delta, q') \mid \delta \in B \cup \{\varepsilon\} \wedge (q, \delta, q') \in \Rightarrow\}$.

As a consequence, from a complexity point of view, computing weak bisimulation equivalence over finite-state LTSs is just a bit harder than computing (strong) bisimulation equivalence: as mentioned above, one has first to compute the (partial) transitive closure $\Rightarrow''$ of the transition relation $\rightarrow$ (by means of the classic Floyd-Warshall algorithm [Flo62], which runs in $O(n^3)$, where n is the number of states) and then to check (strong) bisimulation equivalence, which is in $O(m \log n)$ time [PT87], where m is the number of transitions.

A peculiar aspect of weak bisimulation equivalence, as well as of all the weak behavioral equivalences we have introduced so far, is that it equates systems with different divergent behavior.

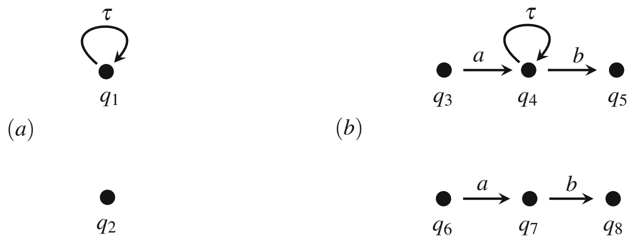


Fig. 2.9 Comparing some divergent systems

Definition 2.20. (Divergent state and livelock) A state q is *divergent* if there exists an infinite path $q_1 \xrightarrow{\tau} q_2 \xrightarrow{\tau} \dots$ of τ-labeled transitions with $q_1 = q$. An LTS $TS = (Q, A, \to)$, with $B = A \setminus \{\tau\}$, is *divergence-free* if no state $q \in Q$ is divergent.

A state q is a *livelock* if for all reachable q', i.e., for all q' such that $q \longrightarrow^* q'$, q' can do at least one τ-labeled transition and cannot do any observable transitions, i.e., $q' \xrightarrow{\tau}$ and $q' \overset{\ell}{\nrightarrow}$ for all $\ell \in B$. □

Clearly, a livelock state is also divergent. Consider the LTSs in Figure 2.9. States q_1 and q_4 are divergent, even if only q_1 is a livelock.

Now we discuss in what sense weak bisimilarity is not sensitive to divergence. Consider the two LTSs in Figure 2.9(a). They are weak trace equivalent and also weak bisimulation equivalent; hence, these weak behavioral equivalences do not distinguish a divergent state (as well as a livelock) from a deadlock! So, weak behavioral equivalences intend to abstract not only from finite amounts of internal work, but also from infinite amounts (i.e., divergences).

Similarly, the two LTSs in Figure 2.9(b) are equated, despite the fact that the upper one, in principle, may diverge and never execute b. The intuition behind this identification is that τ-cycles cannot be taken forever when an alternative is present, i.e., weak bisimilarity assumes that any computation is *fair*: if b is possible infinitely often, then b will be eventually chosen and executed.

2.3 Step Transition Systems

As discussed in Chapter 1, LTSs are an interleaving model of computation, where the states are monolithic and transitions are labeled by one single action. A generalization of LTSs that allows for the modeling of *parallel* activities is the model of Step Transition Systems (STSs, for short), which are LTSs whose transitions are labeled with a *multiset* of concurrently executed actions. Therefore, the states are still monolithic, but the transitions show visibly the potential parallelism of the system. An example may help clarify the idea. Consider the STSs in Figure 2.10. The STS rooted in q_1 models a strictly sequential system, as all the transitions are labeled with a singleton. On the contrary, the STS rooted in q_5 models a concurrent system

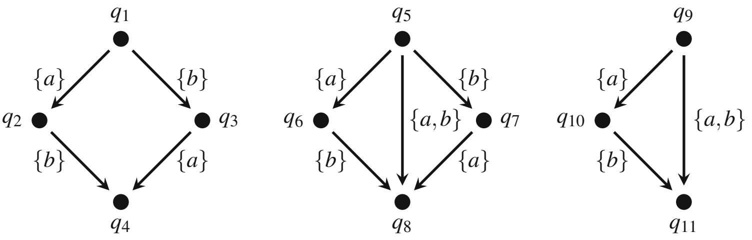


Fig. 2.10 Three step transition systems

because transition $q_5 \xrightarrow{\{a,b\}} q_8$ indicates that q_5 is able to perform two actions, namely a and b, at the same time. Therefore, even if the STS does not show the components of state q_5, still it enables us to observe that these two actions have been performed at the same time by distinct subcomponents.

Definition 2.21. (Multiset) A *multiset* M over a set A is an unordered, possibly infinite, list of elements of A, where no element of A can occur infinitely many times. This is usually represented as a set with *multiplicities*; for instance, $M = \{\ell, \tau, \ell, \tau, \tau\}$ is a multiset over the set $A = \{\ell, \tau\}$ with two occurrences of action ℓ and three occurrences of action τ. It can be represented formally as a function $M : A \to \mathbb{N}$ such that $M(x)$ is the number of instances of element $x \in A$ in M. A multiset M over A is *finite* if $M(x) > 0$ for only finitely many $x \in A$. Of course, if A is finite, then any multiset over A is a finite multiset. The set of all finite multisets over a set A is denoted by $\mathscr{M}_{fin}(A)$. The operator of multiset union $\oplus$ is defined as follows: $(M_1 \oplus M_2)(x) = M_1(x) + M_2(x)$. □

Definition 2.22. (Step labels) Given a set Lab of labels, as in Definition 2.1, the set of *step labels* is $SLab = \mathscr{M}_{fin}(Lab) \setminus \{\emptyset\}$, i.e., the set of all finite, nonempty multisets over Lab. □

Definition 2.23. (Step transition systems) A step transition system (STS for short) is a triple $STS = (Q, \mathscr{B}, \to)$ where

- Q is the nonempty, countable set of *states*, ranged over by q (possibly indexed);
- $\mathscr{B} \subseteq SLab$ is the countable set of *step labels*, ranged over by M (possibly indexed);
- $\to \subseteq Q \times \mathscr{B} \times Q$ is the *step transition relation.*

For economy's sake, we assume that for any $M \in \mathscr{B}$ there exists a step transition $(q, M, q') \in \to$. A *rooted* step transition system is a pair (STS, q_0) where $STS = (Q, A, \to)$ is an STS and $q_0 \in Q$ is the *initial state* (or *root*). Sometimes we write $STS = (Q, \mathscr{B}, \to, q_0)$ for a rooted STS. □

Of course, one can adapt all the strong equivalences we have defined over LTSs also to STSs. In particular, step bisimulation equivalence, defined below, is just ordinary bisimulation equivalence over an STS.

$\dfrac{\ell \in Lab}{o(\{\ell\}) = \{\ell\}}$	$\dfrac{o(M) = \{\tau\}}{o(M \cup \{\ell\}) = \{\ell\}}$	$\dfrac{o(M) \neq \{\tau\}}{o(M \cup \{\tau\}) = o(M)}$	$\dfrac{o(M) \neq \{\tau\} \quad \ell \neq \tau}{o(M \cup \{\ell\}) = o(M) \cup \{\ell\}}$

Table 2.1 Observable content of a step label

Definition 2.24. (Step bisimulation) Let $STS = (Q, \mathscr{B}, \to)$ be a step transition system. A *step bisimulation* is a relation $R \subseteq Q \times Q$ such that if $(q_1, q_2) \in R$ then for all $M \in \mathscr{B}$

- $\forall q_1'$ such that $q_1 \xrightarrow{M} q_1'$, $\exists q_2'$ such that $q_2 \xrightarrow{M} q_2'$ and $(q_1', q_2') \in R$,
- $\forall q_2'$ such that $q_2 \xrightarrow{M} q_2'$, $\exists q_1'$ such that $q_1 \xrightarrow{M} q_1'$ and $(q_1', q_2') \in R$.

Two states q and q' are *step bisimilar* (or *step bisimulation equivalent*), denoted $q \sim_{step} q'$, if there exists a step bisimulation R such that $(q, q') \in R$. □

Clearly, the STSs in Figure 2.10 are not step bisimilar: $q_1 \not\sim_{step} q_5$ because transition $q_5 \xrightarrow{\{a,b\}} q_8$ cannot be matched by q_1; analogously, $q_1 \not\sim_{step} q_9$ because only q_9 can perform the step $\{a, b\}$; finally, $q_5 \not\sim_{step} q_9$ because transition $q_5 \xrightarrow{\{b\}} q_7$ cannot be matched by q_9.

Definition 2.25. (Fully concurrent STS) An STS $(Q, \mathscr{B}, \to)$ is *fully concurrent* if whenever $q \xrightarrow{M_1 \oplus M_2} q'$, with $M_i \neq \emptyset$ for $i = 1, 2$, then there exists two states q_1 and q_2 such that

$$q \xrightarrow{M_1} q_1 \xrightarrow{M_2} q' \quad \text{and} \quad q \xrightarrow{M_2} q_2 \xrightarrow{M_1} q'. \qquad \square$$

Therefore, a fully concurrent STS $(Q, \mathscr{B}, \to)$ is such that, whenever $q \xrightarrow{M} q'$, with $M = \{\ell_1, \ell_2, \ldots, \ell_n\}$, then for any linearization $\ell_1' \ell_2' \ldots \ell_n'$ of the step M (i.e., for any permutation of the string $\ell_1 \ell_2 \ldots \ell_n$), there is a path

$$q_1 \xrightarrow{\{\ell_1'\}} q_2 \xrightarrow{\{\ell_2'\}} \ldots q_n \xrightarrow{\{\ell_n'\}} q_{n+1}$$

where $q_1 = q$, $q_{n+1} = q'$. By looking at Figure 2.10, it is clear that the STSs rooted in q_1 and q_5 are fully concurrent, while the STS rooted in q_9 is not.

In other words, a fully concurrent STS has the property that the parallel behavior of the system can be simulated sequentially: if two actions can be performed in parallel, then they can occur in either order (as for state q_5 in Figure 2.10). However, note that it is not true that if two actions can be performed in either order, then they can be performed in parallel (see state q_1 in Figure 2.10), and this explains the superior expressive power of (fully concurrent) STSs w.r.t. LTSs.

Weak equivalences can also be defined over STSs. The observable content of a step label M, denoted wth $o(M)$, is the multiset where additional occurrences of the invisible action τ are removed. Formally, this is defined in Table 2.1 by induction

$$\frac{}{q \overset{\{\varepsilon\}}{\Longrightarrow} q} \qquad \frac{q_1 \overset{M_1}{\Longrightarrow} q_2 \quad q_2 \xrightarrow{M_2} q_3 \quad o(M_2) = \{\tau\}}{q_1 \overset{M_1}{\Longrightarrow} q_3}$$

$$\frac{q_1 \overset{\{\varepsilon\}}{\Longrightarrow} q_2 \quad q_2 \xrightarrow{M_2} q_3 \quad o(M_2) \neq \{\tau\}}{q_1 \overset{o(M_2)}{\Longrightarrow} q_3}$$

Table 2.2 Weak step transition relation

on the size of the multiset. For instance, $o(\{\tau,\ell,\tau\}) = \{\ell\}$, $o(\{\ell,\ell,\tau\}) = \{\ell,\ell\}$ and $o(\{\tau,\tau\}) = \{\tau\}$.

Also the weak step transition $\overset{M}{\Longrightarrow}$ is to be adapted, as defined in Table 2.2; to be precise, the label M of a weak step transition $\overset{M}{\Longrightarrow}$ does not range over *SLab*, rather over $\{\{\varepsilon\}\} \cup (\mathscr{M}_{fin}(Lab \setminus \{\tau\}) \setminus \{\emptyset\})$.

Note that a path $q_1 \xrightarrow{M_1} q_2 \xrightarrow{M_2} \ldots q_n \xrightarrow{M_n} q_{n+1}$ (with $n \geq 0$), such that $o(M_i) = \{\tau\}$ for $i = 1, \ldots, n$, yields that $q_1 \overset{\{\varepsilon\}}{\Longrightarrow} q_{n+1}$. Moreover, it can be proved that, for $M \neq \{\varepsilon\}$, the weak step transititon $q \overset{M}{\Longrightarrow} q'$ is derivable if and only if there exist two states q_1 and q_2 and a step label M' such that $q \overset{\{\varepsilon\}}{\Longrightarrow} q_1 \xrightarrow{M'} q_2 \overset{\{\varepsilon\}}{\Longrightarrow} q'$, with $o(M') = M$.

Definition 2.26. (Weak step simulation and weak step bisimulation) For any step transition system $STS = (Q, \mathscr{B}, \rightarrow)$, a *weak step simulation* is a relation $R \subseteq Q \times Q$ such that if $(q_1, q_2) \in R$ then for all $M \in \mathscr{B}$

- if $o(M) \neq \{\tau\}$, then $\forall q_1'$ such that $q_1 \xrightarrow{M} q_1'$, $\exists q_2'$ such that $q_2 \overset{o(M)}{\Longrightarrow} q_2'$ and $(q_1', q_2') \in R$;
- if $o(M) = \{\tau\}$, then $\forall q_1'$ such that $q_1 \xrightarrow{M} q_1'$, $\exists q_2'$ such that $q_2 \overset{\{\varepsilon\}}{\Longrightarrow} q_2'$ and $(q_1', q_2') \in R$.

State q is *weakly step simulated* by q', denoted by $q \lesssim_{step} q'$, if there exists a weak step simulation R such that $(q, q') \in R$.

A *weak step bisimulation* is a relation $R \subseteq Q \times Q$ such that R and its inverse R^{-1} are weak step simulations. States q and q' are *weakly step bisimilar* (or *weak step bisimulation equivalent*), denoted by $q \approx_{step} q'$, if there exists a weak step bisimulation R such that $(q, q') \in R$. □

Chapter 3
Petri Nets

Abstract Finite Place/Transition Petri nets are introduced as a suitable semantic model for distributed systems. As this semantic model is not Turing-complete, some interesting properties, such as reachability, boundedness and liveness, are decidable. Also some behavioral equivalences, such as interleaving bisimulation equivalence and step bisimulation equivalence, are presented; these are decidable only for bounded nets. We also discuss a decidable behavioral equivalence based on the structure of finite nets, i.e., net isomorphism. Finally, a Turing-complete class of finite Petri nets, called Nonpermissive Petri nets, is introduced.

3.1 Introduction

In 1962 Carl Adam Petri proposed in his doctoral dissertation [Petri62] a novel semantic model, later called *Petri nets* in his honor, that generalizes automata, hence also transition systems.[1] The basic idea is to describe the global state of a system as composed of a collection of local states; a transition does not represent a global transformation, rather it applies only to a subset of local states. In Petri net terminology (see, e.g., [Pet81, Rei13] for good introductory textbooks on Petri nets), a local state s is called a *place* and it is graphically represented by a circle; a global state m, called a *marking*, is a multiset of local states, where the actual number n of copies of a certain local state s in m is denoted by the presence of n *tokens* (graphically represented as little bullets) inside place s; a net *transition*, graphically represented by a small box, is connected to all the places that take part in the local operation that the transition wants to model. Traditionally, a place is seen as a resource type; the tokens in such a place denote the number of instances of resources of that type; a transition represents an activity which consumes and produces resources. In our process algebraic view, the interpretation of places and transitions is a bit differ-

[1] It seems that Petri first devised such a model in 1939 when he was 13, for the purpose of modeling chemical reactions.

R. Gorrieri, *Process Algebras for Petri Nets*, Monographs in Theoretical Computer Science. An EATCS Series, DOI 10.1007/978-3-319-55559-1_3

ent: a place represents a sequential process type; the tokens in such a place denote the number of instances of processes of that type; a transition represents the evolution of some processes that interact by synchronizing. This will become clear in the following chapters.

There are several different classes of Petri nets in the literature, depending on some peculiar features, such as the constraint of admitting at most one token in places (e.g., *Condition/Event Systems* [Rei85] or *Elementary Net Systems* [RE98]), or the constraint of having a particular net structure (e.g., *Free Choice Nets* [DE95]) or allowing for the capability of transitions to test a place for zero (so called *Nets with inhibitor arcs* — see, e.g., [Pet81, Bus02]).

Among these many variations, we focus our attention on the most prominent and best-studied class of finite Petri nets, with the specific features that a place may hold an unbounded number of tokens, that the tokens on a place are indistinguishable and that a transition may consume from (or produce into) a certain place multiple tokens. These nets are called finite *Place/Transition Petri nets*, usually abbreviated as finite P/T nets. For this class of Petri nets, some useful subclasses are singled out, some interesting decidable properties are presented, some behavioral equivalences are defined and their decidability is discussed. The final part of this chapter describes a novel class of finite Petri nets, we call *Nonpermissive* Petri nets (NP/T, for short), which constitutes a generalization of P/T nets with inhibitor arcs. This class is interesting because it is expressive enough to be Turing-complete; nonetheless, the behavioral equivalence of net isomorphism is decidable, as the nets in this class are finite.

3.2 Place/Transition Petri Nets

We use here a nonstandard notation (also inspired by [Gol88, MM90, Old91]) that better suits our needs and that better reflects that P/T nets are a generalization of labeled transition systems.

Continuing Definition 2.21, we give here more formal definitions of multiset, of operations on multisets and of representation conventions.

Definition 3.1. (Multiset) Let $\mathbb{N}$ be the set of natural numbers. Given a countable set S, a *finite multiset* over S is a function $m : S \to \mathbb{N}$ such that the *support* set $dom(m) = \{s \in S \mid m(s) \neq 0\}$ is finite. The set of all finite multisets over S, denoted by $\mathscr{M}_{fin}(S)$, is ranged over by m, possibly indexed. (The set of all finite subsets of S is denoted by $\mathscr{P}_{fin}(S)$.) We write $s \in m$ if $m(s) > 0$. The *multiplicity* of s in m is given by the number $m(s)$. The *cardinality* of m, denoted by $|m|$, is the number $\sum_{s \in S} m(s)$, i.e., the total number of its elements. A multiset m such that $dom(m) = \emptyset$ is called *empty* and is denoted by $\emptyset$, with abuse of notation. We write $m \subseteq m'$ if $m(s) \leq m'(s)$ for all $s \in S$. We also write $m \subset m'$ if $m \subseteq m'$ and $m(s) < m'(s)$ for some $s \in S$.

The operator $\oplus$ denotes *multiset union* and is defined as follows: $(m \oplus m')(s) = m(s) + m'(s)$; the operation $\oplus$ is commutative, associative and has $\emptyset$ as neutral element. If $m_2 \subseteq m_1$, then we can define *multiset difference*, denoted by the operator $\ominus$, as follows: $(m_1 \ominus m_2)(s) = m_1(s) - m_2(s)$. The *scalar product* of a number j with a multiset m is the multiset $j \cdot m$ defined as $(j \cdot m)(s) = j \cdot (m(s))$.

A finite multiset m over $S = \{s_1, s_2, \ldots\}$ can be represented as $k_1 \cdot s_{i_1} \oplus k_2 \cdot s_{i_2} \oplus \ldots \oplus k_n \cdot s_{i_n}$, where $dom(m) = \{s_{i_1}, \ldots s_{i_n}\} \subseteq S$ and $k_j = m(s_{i_j}) > 0$ for $j = 1, \ldots, n$. If S is finite, i.e., $S = \{s_1, \ldots, s_n\}$, then a finite multiset can be represented also as $k_1 \cdot s_1 \oplus k_2 \cdot s_2 \oplus \ldots \oplus k_n \cdot s_n$, where $k_j = m(s_j) \geq 0$ for $j = 1, \ldots, n$. □

Definition 3.2. (P/T Petri net) A labeled *Place/Transition* Petri net (P/T net for short) is a tuple $N = (S, A, T)$, where

- S is the countable set of *places*, ranged over by s (possibly indexed),
- $A \subseteq Lab$ is the countable set of *labels*, ranged over by ℓ (possibly indexed), and
- $T \subseteq (\mathscr{M}_{fin}(S) \setminus \{\emptyset\}) \times A \times \mathscr{M}_{fin}(S)$ is the countable set of *transitions*, ranged over by t (possibly indexed), such that, for each $\ell \in A$, there exists a transition $t \in T$ of the form (m, ℓ, m').

Given a transition $t = (m, \ell, m')$, we use the notation:

- $^\bullet t$ to denote its *pre-set* m (which cannot be an empty multiset) of tokens to be consumed;
- $l(t)$ for its *label* ℓ, and
- $t^\bullet$ to denote its *post-set* m' of tokens to be produced.

Hence, transition t can be also represented as $^\bullet t \xrightarrow{l(t)} t^\bullet$. We also define pre-sets and post-sets for places as follows: $^\bullet s = \{t \in T \mid s \in t^\bullet\}$ and $s^\bullet = \{t \in T \mid s \in {}^\bullet t\}$. Note that while the pre-set (post-set) of a transition is, in general, a multiset, the pre-set (post-set) of a place is a set. □

Remark 3.1. **(Constraints on the definition of P/T net)** Our definition of T as a set of triples ensures that the net is *transition simple*, i.e., for any $t_1, t_2 \in T$, if $^\bullet t_1 = {}^\bullet t_2$ and $t_1^\bullet = t_2^\bullet$ and $l(t_1) = l(t_2)$, then $t_1 = t_2$. Note also that we are assuming that each transition has a nonempty pre-set: for our interpretation of net models, where a transition can only be performed by some sequential processes, this requirement is strictly necessary (see also Remark 3.3). These are the only two constraints we impose over the definition of a P/T net (*cf* [Pet81, Rei85, DR98]). The additional condition that the set A of labels is covered by T (i.e., for each $\ell \in A$ there exists $t \in T$ with label ℓ) is just for economy. □

Remark 3.2. **(LTSs form a subclass of Petri nets)** If we compare the definition above of labeled P/T Petri net with Definition 2.2 of labeled transition system, we note that the former is a generalization of the latter: a transition system is just a special case of Petri net, where each net transition $t = (m, a, m')$ is such that m and m' are singletons. This issue will be discussed further in Chapter 4. □

Example 3.1. An example of a P/T net is $N = (S, A, T)$, where $S = \{s_1, s_2, s_3, s_4\}$, $A = \{a, b, c\}$ and $T = \{(2 \cdot s_1 \oplus s_2, a, s_1), (s_2 \oplus s_3, b, 3 \cdot s_1 \oplus s_2 \oplus 2 \cdot s_3), (s_1 \oplus 2 \cdot s_3, c, \emptyset)\}$. The graphical representation of this P/T net is in Figure 3.1, where the

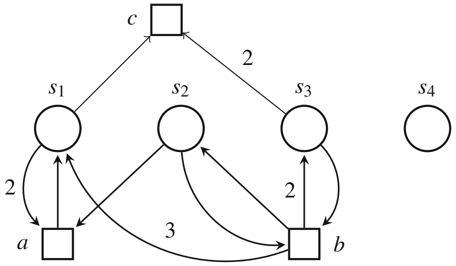


Fig. 3.1 A simple P/T Petri net

arcs connecting circles (i.e., places) to boxes (i.e., transitions), as well as boxes to circles, may be labeled with the natural number representing the number of tokens of that type that are to be removed from (or produced into) that place; no label on the arc is interpreted as the number one, i.e., one token flowing on the arc. □

In a transition system the current global state of the system is just one of its states (for instance, the initial state at the beginning of the computation). In a Petri net, the generalization of the current global state is given by means of a *marking*, i.e., a distribution of tokens over the places of the net, mathematically represented by a finite multiset m over the set S of places.

Definition 3.3. (Marking, token, P/T net system) A finite multiset over S is called a *marking*. Given a marking m and a place s, we say that the place s contains $m(s)$ *tokens*, graphically represented by $m(s)$ bullets inside place s. A *P/T net system* $N(m_0)$ is a tuple (S,A,T,m_0), where (S,A,T) is a P/T net and m_0 is a marking over S, called the *initial marking*. We also say that $N(m_0)$ is a *marked* net. □

Once a net is marked with an initial marking, we can start the so-called *token game*, i.e., we can see which of the net transitions can be *executed* (or *fired*, in Petri net terminology); the effect of performing a transition t is to remove tokens from the pre-set of t and to add tokens to the post-set of t.

Definition 3.4. (Token game) Given a labeled P/T net $N = (S,A,T)$, we say that a transition t is *enabled* at marking m, denoted by $m[t\rangle$, if ${}^\bullet t \subseteq m$. The execution (or *firing*) of t enabled at m produces the marking $m' = (m \ominus {}^\bullet t) \oplus t^\bullet$. This is written $m[t\rangle m'$. □

Example 3.2. Continuing Example 3.1 about Figure 3.1, assume we assign an initial marking $m_0 = s_1 \oplus s_2 \oplus s_3 \oplus s_4$, i.e., each place has exactly one token in it. Clearly, transition $t_1 = (2 \cdot s_1 \oplus s_2, a, s_1)$ is not enabled at m_0, because it needs two tokens in s_1; similarly, transition $t_3 = (s_1 \oplus 2 \cdot s_3, c, \emptyset)$ is not enabled at m_0. On the contrary, transition $t_2 = (s_2 \oplus s_3, b, 3 \cdot s_1 \oplus s_2 \oplus 2 \cdot s_3)$ is enabled at m_0. The firing of t_2 produces the new marking $m_1 = (m_0 \ominus {}^\bullet t_2) \oplus t_2^\bullet = (s_1 \oplus s_4) \oplus (3 \cdot s_1 \oplus s_2 \oplus 2 \cdot s_3) = 4 \cdot s_1 \oplus s_2 \oplus 2 \cdot s_3 \oplus s_4$. Note that at m_1 all of t_1, t_2 and t_3 are enabled. In particular, if t_3 is fired, $m_1[t_3\rangle m_2$, then the reached marking $m_2 = 3 \cdot s_1 \oplus s_2 \oplus s_4$ is such that transitions t_2 and t_3 cannot be fired. □

Remark 3.3. **(Empty post-set or pre-set of a transition)** In the definition of a transition $t = (m, \ell, m')$, it may be the case that its post-set m' is $\emptyset$. This is, for instance, the case in the net in Figure 3.1, where the firing of c produces no tokens. In our interpretation of tokens as sequential processes, we think this situation is quite acceptable: in performing c, the involved sequential processes dissolve.

If we allowed that the pre-set of t may be empty, i.e., $m = \emptyset$, then the transition t would always be enabled and could be always fired by ... no sequential process. In our process algebraic interpretation of net models, it is not possible to accept transitions that are not performed by sequential processes and so in Definition 3.2 we have imposed that net transitions have nonempty pre-set. □

Remark 3.4. **(Permissive nature of P/T Petri nets)** A distinguishing feature of P/T nets is their so-called *permissiveness* [Pet81]: if t is enabled at m, then t is enabled also by any other marking m' covering m, i.e., by any m' such that $m \subseteq m'$. This is in contrast with *P/T nets with inhibitor arcs* [FA73, Hack76a, Pet81, JK95, Bus02], where a transition t enabled at m may be disabled at m' because m' can contain a token in an inhibiting place for t. We will discuss this issue further in Section 3.5, where we propose a generalization of P/T nets with inhibitor arcs, called *Nonpermissive Petri nets*. □

Example 3.3. **(Bounded producer/consumer** $2PC$**)** Consider the Petri net system in Figure 3.2(a) with initial marking $m_0 = 2 \cdot P \oplus C$. Place P represents a producer, and place C a consumer; the overall system is composed of two producers and one consumer. A producer first produces some good — transition $t = (P, prod, P')$ — and then forwards it to the consumer — transition $t' = (C \oplus P', \tau, P \oplus C')$ — which can then consume the good — transition $t'' = (C', cons, C)$.

Transition $t = (P, prod, P')$ is the only transition enabled at the initial marking m_0; its firing produces the marking $m_1 = P \oplus C \oplus P'$ (see Figure 3.2(b)). Then, transition t is enabled again, and its firing produces the marking $m_2 = C \oplus 2 \cdot P'$ (Figure 3.2(c)). Also transition $t' = (C \oplus P', \tau, P \oplus C')$ is enabled at m_1, and its firing produces the marking $m_3 = 2 \cdot P \oplus C'$. From marking m_2, the only enabled transition is t', and the result of its firing is marking $m_4 = P \oplus C' \oplus P'$ (Figure 3.2(d)). Transition $t'' = (C', cons, C)$ is enabled at m_3, and its firing produces the initial marking m_0. Also transition t is enabled at m_3, producing marking m_4. From marking m_4, the firing of t'' produces m_1, while the firing of t produces marking $m_5 = C' \oplus 2 \cdot P'$. Finally, from m_5 only transition t'' is enabled, and its firing produces m_2. So we have explored the whole state space for this net; it is composed of six global states (called *reachable markings*): m_0, m_1, m_2, m_3, m_4 and m_5. (See also Example 6.21 for a description of this bounded producer/consumer in FNC.) □

Example 3.4. **(Unbounded producer/consumer** UPC**)** As a second, more interesting example, consider the unbounded producer/consumer in Figure 3.3. It is interesting to note that place P' can hold an unbounded number of tokens, as the producer P can perform the initial transition *prod* repeatedly, depositing each time one token in that place. Therefore, this example shows an interesting feature of Petri nets:

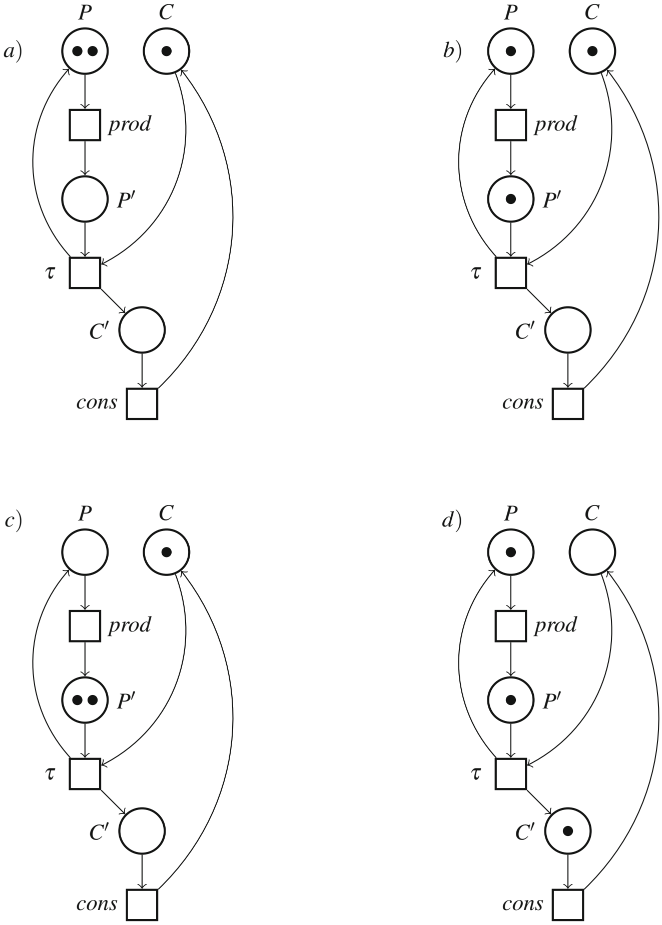


Fig. 3.2 The Petri net for $2PC$ in its initial marking (a) and after some transitions (b)-(d)

in some cases, we can offer a finite Petri net model for a system whose reachable markings are infinitely many. (See also Example 6.12 for its FNC description.) □

Let us now formalize the definition of reachable marking.

Definition 3.5. (Reachable markings and firing sequences) Given a P/T net system $N(m_0) = (S, A, T, m_0)$, the set of markings *reachable from m*, denoted $[m\rangle$, is defined as the least set such that

- $m \in [m\rangle$ and
- if $m_1 \in [m\rangle$ and, for some transition $t \in T$, $m_1[t\rangle m_2$, then $m_2 \in [m\rangle$.

We say that m is *reachable* if m is reachable from the initial marking m_0. A *firing sequence* starting at m is defined inductively as follows:

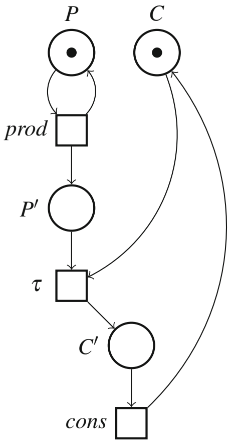


Fig. 3.3 The Petri net for an unbounded producer/consumer UPC

- m is a firing sequence and
- if $m_1[t_1\rangle m_2 \ldots [t_{n-1}\rangle m_n$ (with $m = m_1$ and $n \geq 1$) is a firing sequence and $m_n[t_n\rangle m_{n+1}$, then $m = m_1[t_1\rangle m_2 \ldots [t_{n-1}\rangle m_n[t_n\rangle m_{n+1}$ is a firing sequence.

A firing sequence $m = m_1[t_1\rangle m_2 \ldots [t_n\rangle m_{n+1}$ is usually abbreviated as $m[t_1 \ldots t_n\rangle m_{n+1}$ and $t_1 \ldots t_n$ is called a *transition sequence* starting at m and ending at m_{n+1}. □

Example 3.5. Continuing Example 3.3, the set of the reachable markings for the P/T net in Figure 3.2 is given by the *finite* set $\{m_0, m_1, m_2, m_3, m_4, m_5\}$. A possible firing sequence is

$$m_0[t\rangle m_1[t'\rangle m_3[t\rangle m_4[t''\rangle m_1[t\rangle m_2[t'\rangle m_4[t\rangle m_5[t''\rangle m_2[t'\rangle m_4[t''\rangle m_1.$$

Note that the set of its firing sequences starting at m_0 is *infinite*. Continuing Example 3.4, the set of reachable markings for the net in Figure 3.3 is *infinite*, as transition $t = (P, prod, P \oplus P')$ can be always fired from the initial marking $P \oplus C$, leading to a different marking $P \oplus n \cdot P' \oplus C$, for any $n \in \mathbb{N}$. Of course, its firing sequences are infinitely many as well. □

3.2.1 Some Classes of Petri Nets

We introduce some classes of P/T Petri nets that will be useful in the following.

Definition 3.6. (Classes of P/T Petri nets) A P/T Petri net $N = (S, A, T)$ is

- *statically acyclic* if there exists no sequence $x_1x_2 \ldots x_n$, such that $n \geq 3$, $x_i \in S \cup T$ for $i = 1, \ldots, n$, $x_1 = x_n$, $x_1 \in S$ and $x_i \in {}^\bullet x_{i+1}$ for $i = 1, \ldots, n-1$;
- *distinct* if all the transitions have distinct labels: for all $t_1, t_2 \in T$, if $l(t_1) = l(t_2)$, then $t_1 = t_2$;
- *finite* if both S and T are finite sets;

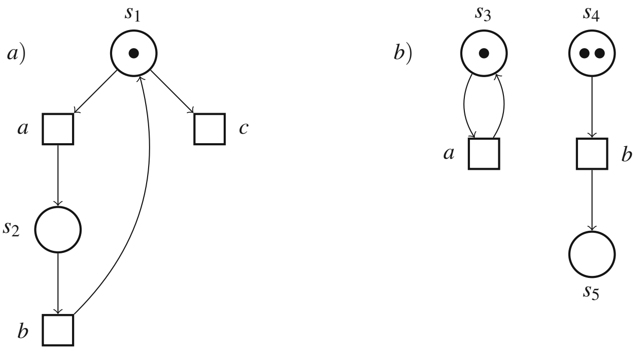


Fig. 3.4 A sequential finite-state machine in (a), and a concurrent finite-state machine in (b)

- a *finite-state machine* (FSM, for short) if N is finite and for all $t \in T$, $|{}^\bullet t| = 1$ and $|t^\bullet| \leq 1$;
- a *BPP net* if N is finite and every transition has exactly one input place, i.e., for all $t \in T$, $|{}^\bullet t| = 1$;
- a *CCS net* if for all $t \in T$, $1 \leq |{}^\bullet t| \leq 2$ and if $|{}^\bullet t| = 2$ then $l(t) = \tau$.

A P/T net system $N(m_0)$ is

- *dynamically acyclic* if there exists no $m_1 \in [m_0\rangle$ with a nonempty (i.e, with $n \geq 2$) firing sequence $m_1[t_1\rangle m_2 \dots [t_{n-1}\rangle m_n$ such that $m_1 \subseteq m_n$;
- a *sequential* FSM if N is an FSM and m_0 is a singleton, i.e., $|m_0| = 1$;
- a *concurrent* FSM if N is an FSM and m_0 is arbitrary;
- *k-bounded* if any place contains at most k tokens in any reachable marking, i.e., $\forall s \in S\ m(s) \leq k$ for all $m \in [m_0\rangle$;
- *safe* if it is 1-bounded;
- *bounded* if $\forall s \in S\ \exists k \in \mathbb{N}$ such that $m(s) \leq k$ for all $m \in [m_0\rangle$. □

The nets in Figure 3.4 are finite-state machines; more precisley, the net in (a) is a sequential FSM, while the net in (b) is a concurrent FSM. Note that FSMs are k-bounded, where $k = |m_0|$, because it is not possible to produce more tokens than those consumed, since $|t^\bullet| \leq |{}^\bullet t| = 1$ for any t of an FSM; therefore, a sequential FSM is safe. Note that the bound $k = |m_0|$ may not be optimal; for instance, the net in Figure 3.4(b) is 2-bounded, while $|m_0| = 3$. Of course, if a net $N(m_0)$ is k-bounded, then it is also $k+1$-bounded, while the reverse is not true; for instance, the net in Figure 3.4(b) is 2-bounded, but it is not safe. The least $k \in \mathbb{N}$ such that the net system $N(m_0)$ is k-bounded is called the *bound limit* of $N(m_0)$.

Of course, any FSM is also a BPP net, and furthermore, any BPP net is a finite CCS net. The net in Figure 3.5(a) is a BPP net; note that this net is not bounded, because there is no upper bound on the number of tokens that can be accumulated in place s_2. The net in Figure 3.5(b) is not BPP because of the τ-labeled transition, while it is a finite CCS net.

The net in Figure 3.6(a) is a safe, acyclic BPP net. The net system in Figure 3.6(b) is a safe, acyclic, infinite net, formally defined as: $S = \{s_1^i \mid i \in \mathbb{N}\} \cup \{s_2^i \mid i \in \mathbb{N}\}$,

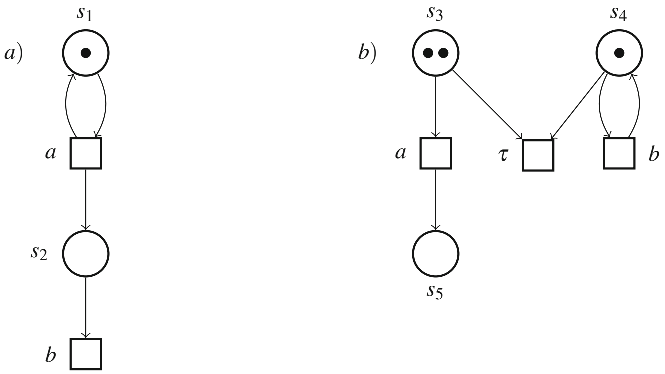


Fig. 3.5 Some further nets: a BPP net in (a), and a CCS net in (b)

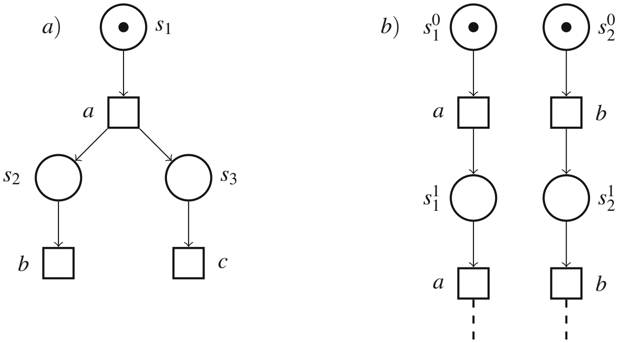


Fig. 3.6 Two nets

$A = \{a,b\}$, $T = \{(s_1^i, a, s_1^{i+1}) \mid i \in \mathbb{N}\} \cup \{(s_2^i, b, s_2^{i+1}) \mid i \in \mathbb{N}\}$ and $m_0 = s_1^0 \oplus s_2^0$. This net is not a (safe, concurrent) FSM, because it is not finite. Note that in a safe net, any reachable marking is a set.

The bounded net in Figure 3.2 is actually 2-bounded. Any finite net that is bounded is also k-bounded for some suitable $k \in \mathbb{N}$; in fact, boundedness implies that for all $s \in S$ there exists an upper bound k_s on the number of tokens that can be accumulated on s; if the net is finite, then it is enough to choose the largest k_s (call it k'), which has the property that for all s, $k_s \leq k'$, so that the net is k'-bounded. It follows that a bounded net that is not k-bounded for any $k \in \mathbb{N}$ is infinite. For instance, consider the net $N(m_0) = (S, A, T, m_0)$ — where $S = \{s_i \mid i \in \mathbb{N}\}$, $A = \{a_i \mid i \in \mathbb{N}\}$, $T = \{(s_i, a_i, 2 \cdot s_{i+1}) \mid i \in \mathbb{N}\}$, $m_0 = \{s_0\}$ — is an infinite net such that place s_i can hold up to 2^i tokens; hence, $N(m_0)$ is bounded, but there is no k such that $2^i \leq k$ for all $i \in \mathbb{N}$.

The interest in these classes is because we will prove in the following chapters that they are strictly related to particular process algebras derived from CCS [Mil89] and Multi-CCS [GV15]. In particular, we will show that SFM process terms originate sequential FSMs (Chapter 4), CFM process terms represent concurrent FSMs (Section 5.1), BPP process terms are mapped to BPP nets (Section 5.2), FNC process terms originate finite CCS P/T nets (Chapter 6), and finally FNM process terms represent all finite P/T nets (Chapter 7).

The following proposition recapitulates some obvious consequences of Definition 3.6 that we have already discussed above.

Proposition 3.1. *Given a P/T system $N(m_0)$, the following hold:*

1. *if N is an FSM net, then N is also a BPP net;*
2. *if N is a BPP net, then N is also a finite CCS net;*
3. *if $N(m_0)$ is a sequential FSM, then $N(m_0)$ is also a concurrent FSM;*
4. *if $N(m_0)$ is a sequential FSM, then $N(m_0)$ is also safe;*
5. *if $N(m_0)$ is a concurrent FSM, then $N(m_0)$ is also $|m_0|$-bounded;*
6. *if $N(m_0)$ is finite and bounded, then $N(m_0)$ is also k-bounded for some suitable $k \in \mathbb{N}$;*
7. *if $N(m_0)$ is finite and bounded, then the set $[m_0\rangle$ of the markings reachable from m_0 is finite;*
8. *if N is statically acyclic, then $N(m_0)$ is dynamically acyclic;*
9. *if $N(m_0)$ is finite and dynamically acyclic, then the set of its firing sequences is finite.*

Proof. *We prove only (7). Assume $N(m_0)$ is finite and bounded, where the cardinality of the set S of places is n and the bound limit on places is k. Then, there cannot be more than $(k+1)^n$ different markings, because each place s can hold any number of tokens in the range $\{0,\dots,k\}$.* □

3.2.2 Dynamically Reachable and Statically Reachable Subnets

Given a P/T net system $N(m_0)$, it may be the case that some of its places are actually never *dynamically reachable* (i.e., reachable by means of the token game) and that some of its transitions can never be performed. Hence, we are interested in singling out the subnet $Net_d(N(m_0))$ of the places and transitions of N dynamically reachable from the initial marking m_0.

Definition 3.7. (Dynamically reachable subnet) Given a P/T net system $N(m_0) = (S, A, T, m_0)$, the *dynamically reachable subnet* $Net_d(N(m_0))$ is (S', A', T', m_0), where

$$\begin{aligned} S' &= \{s \in S \mid \exists m \in [m_0\rangle \text{ such that } m(s) \geq 1\},\\ T' &= \{t \in T \mid \exists m \in [m_0\rangle \text{ such that } m[t\rangle\},\\ A' &= \{\ell \mid \exists t \in T' \text{ such that } l(t) = \ell\}. \end{aligned}$$

□

Consider the net system in Figure 3.7(a): its dynamically reachable subnet is outlined in Figure 3.7(b).

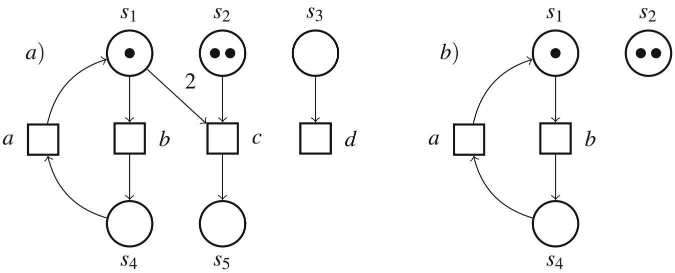


Fig. 3.7 A net system in (a) and its dynamically reachable subnet in (b)

Definition 3.8. (Dynamically reduced net) A P/T net system $N(m_0) = (S, A, T, m_0)$ is *dynamically reduced* if $N(m_0) = Net_d(N(m_0))$, i.e., the net system is equal to its dynamically reachable subnet. □

For instance, the net system in Figure 3.2 is dynamically reduced, and so too is the net in Figure 3.7(b). In Section 3.3.1 we will show that, given a finite P/T net $N(m_0)$, it is algorithmically possible to derive $Net_d(N(m_0))$ by means of the coverability tree.

We are actually interested in a weaker variant of these notions for finite nets: the *statically* reachable subnet $Net_s(N(m_0))$ of a net system $N(m_0)$ and the definition of *statically reduced* net.

Definition 3.9. (Statically reachable subnet and statically reduced net) Given a finite P/T net $N = (S, A, T)$, we say that a transition t is *statically enabled* by a set of places $S' \subseteq S$, denoted by $S'[\![t\rangle$, if $dom({}^\bullet t) \subseteq S'$.

Given two sets of places $S_1, S_2 \subseteq S$, we say that S_2 is *statically reachable in one step* from S_1 if there exists a transition $t \in T$, such that $S_1[\![t\rangle$, $dom(t^\bullet) \not\subseteq S_1$ and $S_2 = S_1 \cup dom(t^\bullet)$; this is denoted by $S_1 \overset{t}{\Longrightarrow} S_2$. The *static reachability relation* $\Longrightarrow^* \subseteq \mathscr{P}_{fin}(S) \times \mathscr{P}_{fin}(S)$ is the least relation such that

- $S_1 \Longrightarrow^* S_1$ and
- if $S_1 \Longrightarrow^* S_2$ and $S_2 \overset{t}{\Longrightarrow} S_3$, then $S_1 \Longrightarrow^* S_3$.

A set of places $S_k \subseteq S$ is the *largest* set statically reachable from S_1 if $S_1 \Longrightarrow^* S_k$ and for all $t \in T$ such that $S_k[\![t\rangle$, we have that $dom(t^\bullet) \subseteq S_k$.

Given a finite P/T net system $N(m_0) = (S, A, T, m_0)$, we denote by $[\![dom(m_0)\rangle$ the largest set of places statically reachable from $dom(m_0)$, i.e., the largest S_k such that $dom(m_0) \Longrightarrow^* S_k$.

The *statically reachable subnet* $Net_s(N(m_0))$ is the net (S', A', T', m_0), where

$$\begin{aligned} S' &= [\![dom(m_0)\rangle, \\ T' &= \{t \in T \mid S'[\![t\rangle\}, \\ A' &= \{\ell \mid \exists t \in T' \text{ such that } l(t) = \ell\}. \end{aligned}$$

```
Let $N(m_0) = (S, A, T, m_0)$ be a finite P/T net system, where $T = \{t_1, \dots, t_k\}$ for $k \geq 0$.

1. Let $S'$ be the set of currently statically reached places, initialized to $dom(m_0)$;
2. Let $P$ be the set of transitions to be checked, initialized to $T$;
3. Let $T'$ be the set of statically reachable transitions, initialized to $\emptyset$;
4. Let $A'$ be the set of labels, initialized to $\emptyset$;
5. while $\exists t_j \in P$ such that $S' [\![ t_j \rangle$ do:
    a) add $dom(t_j^\bullet)$ to $S'$;
    b) remove $t_j$ from $P$;
    c) add $t_j$ to $T'$;
    d) add $l(t_j)$ to $A'$;
6. The statically reachable subnet is $Net_s(N(m_0)) = (S', A', T', m_0)$.
```

Table 3.1 Algorithm for computing the statically reachable subnet

A finite P/T net system $N(m_0) = (S, A, T, m_0)$ is *statically reduced* if $Net_s(N(m_0)) = N(m_0)$, i.e., the net system is equal to its statically reachable subnet. □

There is an obvious algorithm to compute $Net_s(N(m_0))$, as outlined in Table 3.1. This algorithm terminates for finite P/T nets because the iteration at step 5) can be repeated $k = |T|$ times at most, because, once a transition has been selected, it is removed from P. Hence, this algorithm is polynomial in the size of the net.

Note that a finite P/T net system $N(m_0) = (S, A, T, m_0)$ is statically reduced if all the places are statically reachable from the places in the initial marking — $[\![dom(m_0) \rangle = S$ — because all the transitions in T are statically enabled by S.

Note also that a finite P/T net system $N(m_0) = (S, A, T, m_0)$ is statically reduced if there exist two sequence $S_1, S_2, \dots, S_{n+1}$ and $t_1, t_2, \dots, t_n$, for $n \geq 0$, such that $dom(m_0) = S_1$, $S = S_{n+1}$ and

$$S_1 \overset{t_1}{\Longrightarrow} S_2 \overset{t_2}{\Longrightarrow} \dots S_n \overset{t_n}{\Longrightarrow} S_{n+1}.$$

Finally, we outline some useful properties relating statically reduced nets and dynamically reduced ones.

Proposition 3.2. *Given a P/T net system $N(m_0) = (S, A, T, m_0)$, if $N(m_0)$ is dynamically reduced, then it is also statically reduced.*

Proof. If $N(m_0)$ is dynamically reduced, then (i) $\forall s \in S\ \exists m \in [m_0\rangle . m(s) \geq 1$, and (ii) $\forall t \in T\ \exists m, m' \in [m_0\rangle$ such that $m[t\rangle m'$. Note that if m is dynamically reachable, then there is a firing sequence $m_0[t_1\rangle m_1 \dots [t_n\rangle m_n = m$, usually abbreviated as $m_0[t_1 \dots t_n\rangle m$; since if a transition is dynamically enabled, then it is also statically enabled, we have $dom(m_0) \Longrightarrow^* dom(m)$; since this holds for any reachable marking m, it follows that $S = [\![dom(m_0) \rangle$, and so $N(m_0)$ is statically reduced. □

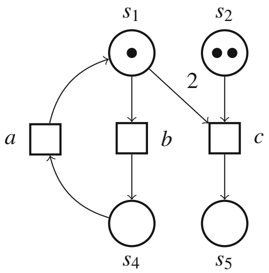


Fig. 3.8 The statically reachable subnet of Figure 3.7(a)

However, the converse implication is not true: there are statically reduced P/T net systems that are not dynamically reduced. For instance, the statically reduced P/T net system $N(s_1) = (\{s_1, s_2, s_3\}, \{a, b\}, \{(s_1, a, s_2), (2 \cdot s_1, b, s_3)\}, s_1)$ cannot dynamically reach place s_3.

Consider the net system in Figure 3.7(a); its statically reachable subnet is outlined in Figure 3.8. If we compare it with its dynamically reachable subnet in Figure 3.7(b), we note that the statically reachable subnet contains the dynamically reachable subnet. This holds in general.

Proposition 3.3. *Given a P/T net system $N(m_0) = (S, A, T, m_0)$, if its dynamically reachable subnet $Net_d(N(m_0))$ is (S', A', T', m_0) and its statically reachable subnet $Net_s(N(m_0))$ is (S'', A'', T'', m_0), then $S' \subseteq S''$, $T' \subseteq T''$ and $A' \subseteq A''$.* □

For some classes of nets, however, the two notions coincide.

Proposition 3.4. *If $N(m_0)$ is a BPP net that is statically reduced, then it is also dynamically reduced.*

Proof. A BPP transition t is such that $|^\bullet t| = 1$; therefore, the notions of dynamically enabled transition and statically enabled transition coincide. □

3.3 Decidable Properties

Finite P/T Petri nets enjoy some interesting properties that are decidable even if the sets of reachable markings and firing sequences are infinite. In particular, many of these properties are decidable by means of the construction of the *coverability tree* [KM69], which is a finite, abstract representation of the firing sequences and of the reachable markings of the net. Some other properties derive from the decidability of the *reachability problem* [Kos82, May84]. These are the topic of the next two subsections.

3.3.1 Coverability Tree

As the set of the reachable markings of a finite P/T net is, in general, infinite, it seems that there is no way to describe, even approximately, the set of reachable markings of the net with a finite structure. However, this is not the case: we can find a finite tree whose nodes are labeled with *extended* markings (i.e., markings that may hold infinitely many tokens in a place), such that each reachable marking is either explicitly represented as a node label of the tree, or *covered* by some node label of the tree. The construction of such a *coverability tree* was proposed by Karp and Miller [KM69].

The naïve construction of the tree proceeds as follows.[2] Initially, the coverability tree contains only one node x_0 (the *root*), labeled with the initial marking m_0 of the net. For each transition t enabled at m_0, we add an arc, labeled with t, from x_0 to a new node x_i, labeled by the marking we get after firing t. Then, repeat this step for all the new nodes. Of course, this naïve construction may easily lead to an infinite tree, even when the number of reachable markings is finite. However, we can omit to process or to add some of the new nodes (called *frontier nodes*) at each step. Two classes of nodes are useful to this aim:

- nodes labeled with *dead* markings (i.e., markings enabling no transition) are not to be processed further, thus they are called *terminal nodes*;
- nodes labeled with markings previously appearing in the tree (i.e., labeling also an ancestor node) need not be processed further: all of its successors have already been produced from the first occurrence of the marking in the tree; these nodes are called *duplicate nodes*.

Even if, following the algorithm sketched above, a bounded net would generate a finite tree (thanks to Proposition 3.1(7)), still this is not enough to get a finite tree when the finite net is unbounded, as there are infinitely many different reachable markings. So, we need a way to abstract from markings without losing information about the firability of transitions enabled at those markings.

Consider a transition sequence $\sigma = t_1 t_2 \ldots t_n$ which starts at marking m and ends at marking m' such that $m \subset m'$. This means that m' is the same as m, except it has some extra tokens in some places: $m' = m \oplus (m' \ominus m)$ with $m' \ominus m \neq \emptyset$. Since transition firings are not affected by extra tokens (see Remark 3.4), the sequence σ can be fired at marking m', leading to a marking $m'' = m' \oplus (m' \ominus m) = m \oplus 2 \cdot (m' \ominus m)$. In general, we can repeat k times the firing of σ to produce a marking $m \oplus k \cdot (m' \ominus m)$. Thus, we can produce an arbitrarily large number of tokens in each place s such that $(m' \ominus m)(s) > 0$ by iterating the firing of σ. We can give a finite representation to this infinite set of markings by using an *extended* marking $\overline{m}$ such that $\overline{m}(s) = m(s)$ if $m(s) = m'(s)$, otherwise $\overline{m}(s) = \omega$, where the special symbol ω represents unboundedness.

Definition 3.10. Let ω be a symbol not in $\mathbb{N}$. For any $n \in \mathbb{N}$, we define $\omega + n = \omega = \omega - n$. Moreover, $n < \omega$ and $\omega \leq \omega$. Given a P/T net $N = (S, A, T)$, an *extended*

[2] The following description is an elaboration of [Pet81, Bus02].

```
Let $N(m_0) = (S, A, T, m_0)$ be a finite P/T net system, where $T = \{t_1, \ldots, t_k\}$ for $k \geq 0$.

• Let $i$ be an integer variable, initialized to 0.
• Let $b$ be a boolean variable.
• Let $F$ be the set of frontier nodes, initialized to $\{x_0\}$, i.e., $F := \{x_0\}$.
• Let $M$ be the labeling function, initialized to $\{(x_0, m_0)\}$, denoted $M[x_0] = m_0$.
• Let $I$ be the set of internal nodes, initialized to $\emptyset$.
• Let $D$ be the set of duplicate nodes, initialized to $\emptyset$.
• Let $B$ be the set of terminal nodes, initialized to $\emptyset$.
• Let $A$ be the set of arcs, initialized to $\emptyset$.
• While $F \neq \emptyset$ do:

    1. let $x_h$ be the element in $F$ with lowest index $h$;
    2. $F := F \setminus \{x_h\}$;
    3. if $\exists x_l \in I \cup D \cup B$ such that $M[x_l] = M[x_h]$, then $D := D \cup \{x_h\}$,
       else
       a. $b := false$;
       b. for $j := 1$ to $k$ do:
          if $t_j$ is enabled at $M[x_h]$ and $m'_j = (M[x_h] \ominus {}^\bullet t_j) \oplus t_j^\bullet$ then
          •  $b := true$;
          •  $i := i + 1$;
          •  $F := F \cup \{x_i\}$;
          •  $M := M \cup \{(x_i, M[x_i])\}$, where for all $s \in S$, $M[x_i](s) = \omega$ if $m'_j(s) = \omega$
             or there exists a node $x_l$ on the path from the root $x_0$ to $x_h$ such that
             $M[x_l] \subset m'_j$ and $M[x_l](s) < m'_j(s)$, otherwise $M[x_i](s) = m'_j(s)$;
          •  $A := A \cup \{(x_h, t_j, x_i)\}$;
       c. if $b$ then $I := I \cup \{x_h\}$ else $B := B \cup \{x_h\}$;

• The coverability tree is $(V, M, A)$, where the set of arcs $A$ and the node labeling
  $M$ are computed as above, while the set $V$ of nodes is $I \cup D \cup B$.
```

Table 3.2 Algorithm for the construction of the coverability tree

marking m is a finite multiset $m : S \to \mathbb{N} \cup \{\omega\}$ (so a marking is just a particular case of an extended marking). We say that an extended marking m' *covers* an extended marking m if $m \subseteq m'$ and, for all $s \in S$, $m(s) \neq m'(s)$ implies $m'(s) = \omega$. □

Assume that an extended marking m, labeling some frontier node x_h, enables t, whose execution produces $m' = (m \ominus {}^\bullet t) \oplus t^\bullet$, and that an ancestor node (i.e., a node in the path from the root to x_h) is labeled with m'', such that $m'' \subset m'$. Then, a new frontier node x_i is added, labeled with the extended marking $\overline{m}$ defined as

$$\forall s \in S, \overline{m}(s) = \begin{cases} \omega & \text{if } m'(s) = \omega \text{ or } m'(s) > m''(s), \\ m'(s) & \text{otherwise.} \end{cases}$$

In this way, each reachable marking either appears explicitly in the tree or there is in the tree an extended marking that covers it.

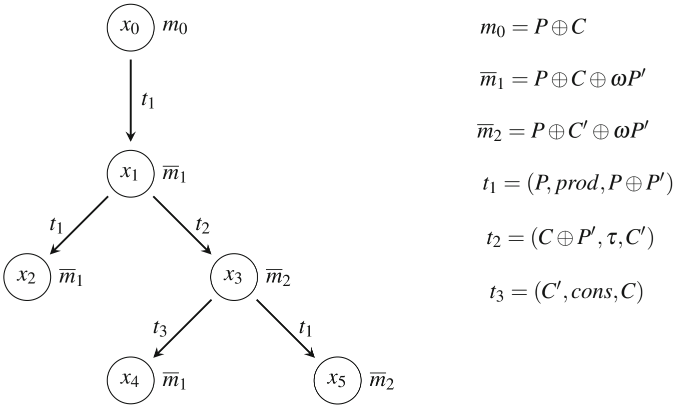


Fig. 3.9 Coverability tree for the unbounded net *UPC* in Figure 3.3

The precise algorithm is described in Table 3.2 and proceeds as follows: each node x_i in the tree is labeled with an extended marking $M[x_i]$ and is classified as a frontier node (in set F), a duplicate node (in set D), a terminal node (in set B) or an internal node (in set I). Frontier nodes are those which have not been processed yet; they are turned by the algorithm into duplicate, terminal or internal nodes. The algorithm begins by defining the initial marking m_0 as the label for the root x_0 of the tree, initially a frontier node. As long as frontier nodes are present, they are processed by the algorithm, which terminates when no frontier nodes are left. The nodes are indexed by increasing natural numbers and, at each iteration step, the node with lowest index, say x_h, is selected and removed from the set F of frontier nodes. If an already processed node x_l has the same labeling as x_h, then x_h is put in the set D of duplicate nodes and will not be processed further; otherwise, for each transition t_j enabled at $M[x_h]$ and reaching the extended marking $m'_j = (M[x_h] \ominus {}^{\bullet}t_j) \oplus t_j^{\bullet}$, we produce a new frontier node x_i and a new arc (x_h, t_j, x_i); the labeling of the node x_i is the extended marking $M[x_i]$ where, for each $s \in S$, $M[x_i](s) = \omega$ if $m'_j(s) = \omega$ or there exists a node x_l on the path from the root x_0 to x_h such that $M[x_l] \subset m'_j$ and $M[x_l](s) < m'_j(s)$, otherwise $M[x_i](s) = m'_j(s)$. If at least one transition is enabled at $M[x_h]$ (i.e., if the flag b is true), then x_h is put in the set I of internal nodes, otherwise in the set B of terminal nodes.

Example 3.6. As an example, consider the P/T net in Figure 3.3 for the unbounded producer/consumer *UPC*. The step-by-step construction of the coverability tree is as follows: the root x_0 of the tree is labeled with the initial marking $M[x_0] = m_0 = P \oplus C$. The only enabled transition is $t_1 = (P, prod, P \oplus P')$ and $m_0[t_1\rangle m_1 = P \oplus C \oplus P'$. Then, we create a new node x_1 and an arc from x_0 to x_1 labeled with transition t_1. As $m_0 \subset m_1$, the extended marking $M[x_1]$ is $\overline{m}_1 = P \oplus C \oplus \omega P'$.

Now, at $M[x_1]$ transition t_1 is enabled again, so that $\overline{m}_1[t_1\rangle \overline{m}_1$. Hence, a new node x_2 is added to the coverability tree, as well as an arc from x_1 to x_2 labeled with t_1. The extended marking $M[x_2]$ is again $\overline{m}_1$ so that the node x_2 is classified as a duplicate one and will not be processed further. At $M[x_1]$ also transition $t_2 =$

$(C \oplus P', \tau, C')$ is enabled and $\overline{m}_1[t_2\rangle\overline{m}_2 = P \oplus C' \oplus \omega P'$. Then, a new node x_3 is added to the coverability tree as well as an arc from x_1 to x_3 labeled with t_2. The extended marking $M[x_3]$ is exactly $\overline{m}_2$, as no ancestor node is covered by $\overline{m}_2$.

Now, at $M[x_3] = \overline{m}_2$ transition $t_3 = (C', cons, C)$ is enabled and $\overline{m}_2[t_3\rangle\overline{m}_1$. Hence, a new node x_4 is added, as well as an arc from x_3 to x_4 labeled with t_3. The extended marking $M[x_4]$ is $\overline{m}_1$, hence also x_4 is classified as a duplicate node. Finally, at $M[x_3] = \overline{m}_2$ also transition t_1 is enabled and $\overline{m}_2[t_1\rangle\overline{m}_2$. Hence, a new node x_5 is added, as well as an arc from x_3 to x_5 labeled with t_1. The extended marking $M[x_5]$ is $\overline{m}_2$, hence also x_5 is classified as a duplicate node.

So, all the nodes we have created are either internal (namely, x_0, x_1 and x_3) or duplicate (namely, x_2, x_4 and x_5), hence the construction of the coverability tree is finished. The resulting tree is depicted in Figure 3.9. □

The coverability tree associated with a finite P/T Petri net by the algorithm in Table 3.2 is finite. The details of the proof are inspired by [Hack75, Pet81], which are based on the following auxiliary lemmata.

Lemma 3.1. *In any infinite, finitely branching tree, there exists an infinite path starting from the root.*

Proof. By König's infinity lemma [Kön36]. □

Lemma 3.2. *Every infinite sequence of elements in $\mathbb{N} \cup \{\omega\}$ contains an infinite subsequence which satisfies one of the following conditions:*

- *either all the elements in the subsequence are equal,*
- *or they appear in strictly increasing order.*

Proof. Either an element n occurs infinitely often in the infinite sequence and the infinite subsequence is $n\,n\,n\ldots$. Or each element occurs only a finite number of times in the infinite sequence; so we can always find a new number greater than those already inserted in the subsequence. □

Lemma 3.3. *Every infinite sequence of extended markings contains an infinite subsequence ordered w.r.t. $\subseteq$.*

Proof. By Dickson's lemma [Dic13]. It can be proved directly by induction on the size of S, the base case $|S| = 1$ guaranteed by Lemma 3.2. □

Theorem 3.1. (Finite coverability tree) *Let $N(m_0)$ be a finite P/T net system. Then its coverability tree is finite.*

Proof. The proof is by contradiction. Assume that the coverability tree is infinite. Since the coverability tree is finitely branching because the set T of transitions is finite, Lemma 3.1 ensures that there is an infinite path $x_0 x_1 x_2 \ldots$ starting from the root x_0. Hence, $M[x_0]M[x_1]M[x_2]\ldots$ is an infinite sequence of extended markings. Note that by construction it is not possible that $M[x_i] = M[x_j]$ for $i \neq j$ because otherwise one would be a duplicate node, hence with no successor (contradicting the *fact that the sequence is infinite). Note also that by Lemma 3.3 there exists an infinite nondecreasing subsequence $M[x_{i_0}]M[x_{i_1}]M[x_{i_2}]\ldots$, which must be strictly increasing because no repetition is possible. By construction, since $M[x_{i_j}] \subset M[x_{i_{j+1}}]$, there is a place s such that $M[x_{i_j}](s) = n$ and $M[x_{i_{j+1}}](s) = \omega$. Hence, $M[x_{i_1}]$ has at least one ω place, $M[x_{i_2}]$ has at least two ω places, and so on. If the cardinality of S is k, then $M[x_{i_k}]$ has all places marked ω, that is $M[x_{i_k}]$ is the maximum extended marking. Hence, $M[x_{i_{k+1}}]$ cannot be strictly greater than $M[x_{i_k}]$. This is a contradiction, proving that our assumption that the coverability tree is infinite was wrong.* □

All reachable markings are covered by some extended marking in the coverability tree and each firing sequence is represented in the coverability tree, as the following proposition explains.

Proposition 3.5. *Given a finite P/T net system $N(m_0) = (S,A,T,m_0)$, for each firing sequence $m_0[t_1\rangle m_1[t_2\rangle \ldots m_{n-1}[t_n\rangle m_n$ there exists a sequence of nodes and arcs $x_0 \xrightarrow{t_1} y_1\ x_1 \xrightarrow{t_2} y_2 \ldots x_{n-1} \xrightarrow{t_n} y_n$ in the coverability tree such that*

- *x_0 is the root of the tree;*
- *$M[y_i] = M[x_i]$ for $i = 1,\ldots n-1$;*
- *m_i is covered by $M[x_i]$ for $i = 0,\ldots n-1$ and m_n is covered by $M[y_n]$.*

Proof. By induction on the length of the firing sequence. □

Moreover, the ω-components in the extended markings appearing in the coverability tree effectively correspond to places that may hold an unlimited number of tokens.

Proposition 3.6. *Let z be a node of the coverability tree associated with a finite P/T net system $N(m_0) = (S,A,T,m_0)$, and let $k > 0$. Then there exists a reachable marking $m \in [m_0\rangle$ such that, for all $s \in S$*

- *if $M[z](s) < \omega$ then $m(s) = M[z](s)$;*
- *if $M[z](s) = \omega$ then $m(s) > k$.*

Proof. Take the path from the root x_0 of the tree to node z. If $M[z](s) \neq \omega$ for all $s \in S$, then the chosen path determines a firing sequence of equal length, that goes from m_0 to m_z, such that, for any $s \in S$, $m_z(s) = M[z](s)$. If $M[z](s) = \omega$ for some $s \in S$, then it is possible to find a firing sequence, possibly longer than the path, that goes from m_0 to m_z, passing through m', such that $m' \subseteq m_z$ and, for any $s \in S$, either $m_z(s) = M[z](s)$ or $m_z(s) = n < \omega = M[z](s)$. Then, if $n > k$ we are done. Otherwise, we can repeat the transition subsequence from m' to m_z as many times as required so that, at the end, the reached m_z^j is such that $m_z^j(s) > k$. □

However, it is not true that any marking covered by an extended marking appearing in the coverability tree is reachable, nor that we can derive all and only the firing sequences from the coverability tree. For instance, consider the P/T net in Figure 3.10(a) and its associated coverability tree in Figure 3.10(b). It is clear that, e.g., marking $m = 2 \cdot s_1$ is not reachable from the initial marking $m_0 = s_1$, even though

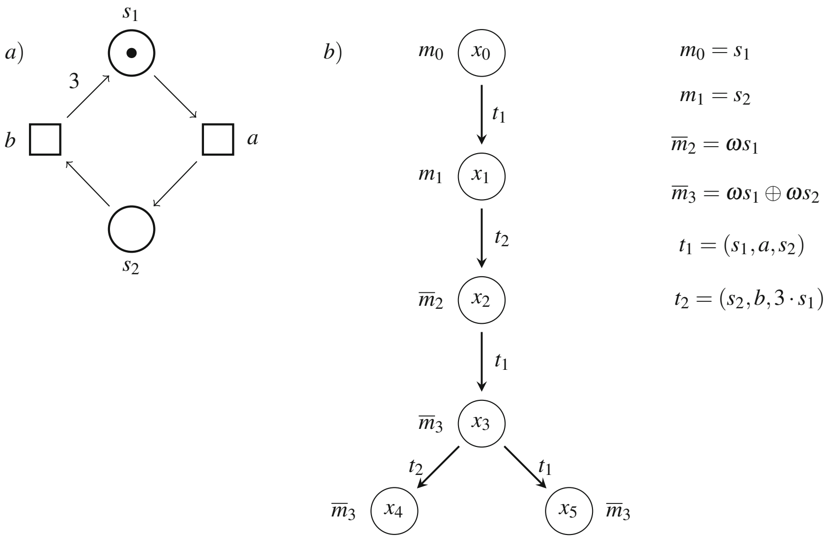


Fig. 3.10 A simple net in (a) and its associated coverability tree in (b)

there is a node in the tree labeled with marking $M[x_2] = \overline{m} = \omega s_1$. Hence, the coverability tree does not provide enough information to solve precisely the reachability problem, even though it may provide enough information to solve some related problems; for instance, if a marking is not covered by some node of the coverability tree, then it is not reachable.

Similar arguments hold for transition sequences. In the light of Proposition 3.5, we may wonder whether any sequence of arcs with the property that adjacent nodes have the same associated extended marking defines a firing sequence. This is not the case. For instance, consider again the P/T net in Figure 3.10(a) and its associated coverability tree in Figure 3.10(b); it is clear that, e.g., from the sequence of arcs

$$x_0 \xrightarrow{t_1} x_1 \; x_1 \xrightarrow{t_2} x_2 \; x_2 \xrightarrow{t_1} x_3 \; x_3 \xrightarrow{t_1} x_5 \; x_3 \xrightarrow{t_1} x_5 \; x_3 \xrightarrow{t_1} x_5$$

which has the property that adjacent nodes have the same associated extended marking, we cannot derive an associated firing sequence, because $t_1 t_2 t_1 t_1 t_1 t_1$ is not a transition sequence of the net. Hence, the coverability tree does not provide enough information to derive the set of firing sequences, even though it may provide enough information to solve some related problems; for instance, if we cannot find a sequence of arcs in the tree with the property that adjacent nodes have the same associated extended marking for a given transition sequence σ, we can conclude that σ is not a transition sequence for the net.

Nonetheless, the coverability tree is a useful tool for the analysis of finite P/T nets. Indeed, some problems about boundedness and coverability can be reduced to properties of the coverability tree.

Theorem 3.2. *A finite P/T net system $N(m_0) = (S, A, T, m_0)$ is bounded if and only if, for all nodes x in its coverability tree and for all $s \in S$, $M[x](s) \neq \omega$.* □

Therefore, boundedness is decidable: it is enough to build the coverability tree and inspect its labeling to check the absence of the symbol ω.

From the coverability tree of a finite P/T net, we can single out its dead transitions, i.e., those transitions that can never fire.

Definition 3.11. Given a finite P/T net system $N(m_0)$, a transition t is *dead* if, for all $m \in [m_0\rangle$, t is not enabled at m. □

Theorem 3.3. *A transition t of a finite P/T net system $N(m_0)$ is dead if no t-labeled arc occurs in the associated coverability tree.* □

Similarly, from the coverability tree, we can single out dead places, i.e., places that can never be marked.

Definition 3.12. Given a finite P/T net system $N(m_0)$, a place s is *dead* if, for all $m \in [m_0\rangle$, $m(s) = 0$. □

Theorem 3.4. *A place s of a finite P/T net system $N(m_0)$ is dead if for all nodes x of the associated coverability tree we have that $M[x](s) = 0$.* □

As a consequence of Theorems 3.3 and 3.4, given a finite P/T net system $N(m_0)$, we can algorithmically compute its associated dynamically reachable subnet $Net_d(N(m_0))$ (Definition 3.7): it is enough to inspect the coverability tree and remove all dead transitions as well as dead places from $N(m_0)$ to get $Net_d(N(m_0))$.

Another interesting problem that can be decided by inspecting the nodes of the coverability tree is the *coverability problem*, which consists of finding a reachable marking m' that is larger than a given marking m, i.e., such that $m \subset m'$.

Theorem 3.5. *Let $N(m_0)$ be a finite P/T net system and let m be a marking. There exists a marking $m' \in [m_0\rangle$ such that $m \subset m'$ if and only if there exists a node x in the coverability tree such that $m \subset M[x]$.* □

From a computational point of view, the very simple algorithm of Karp and Miller for constructing the coverability tree turns out to be surprisingly inefficient: the construction may require non-primitive recursive space! Hence, also proving boundedness by means of the coverability tree is not primitive recursive in general. However, Rackoff in [Rac78] gave a better algorithm, showing that boundedness can be decided for finite P/T net in $2^{O(n \log n)}$ in the size n of the net. This is very close to the theoretical lower bound, as Lipton proved in [Lip76] that deciding boundedness requires at least exponential space (EXPSPACE-hard), more precisely $2^{O(\sqrt{n})}$.

Strictly connected to the boundedness problem is the problem of deciding whether a finite P/T net is safe: such a problem is PSPACE-complete [JLL77].

3.3.2 Reachability, Liveness and Deadlock

We recall that a marking is reachable if there exists a firing sequence (starting from the initial marking) leading to it. The reachability problem for a net consists of deciding whether a given marking is reachable from the initial marking. As we have seen, the coverability tree can be used to solve the reachability problem only partially: if a marking m occurs in the coverability tree, then it is reachable; symmetrically, if m is not covered by some node in the coverability tree, then it is not reachable. However, in general, a different technique is necessary and this was the research goal of many researchers for several years.

The reachability problem is decidable. Sacerdote and Tenney provided in [ST77] a partial proof. Such a proof was completed in 1981 by Mayr [May81, May84] and simplified by Kosaraju [Kos82] on the basis of [ST77, May81]. A whole book [Reu88] is devoted to a step-by-step description of such a long and complex proof. Later, Lambert in [Lam92] provided a simplified proof, based on [Kos82]. Recently, an even simpler proof of decidability of the reachability problem for finite P/T nets has been presented by Leroux in [Ler10, Ler11] which has an associated, more accessible algorithm. About complexity, Lipton proved an exponential space lower bound (EXPSPACE-hard) [Lip76]. None of the algorithms known so far is primitive recursive in the general case of finite P/T nets. Nonetheless, in some restricted cases, the complexity is much better: for safe nets the reachability problem is PSPACE-complete [CEP95], while for BPP nets it is NP-complete [Huy83, Esp95].

Hack showed in [Hack76b] that some variations of the reachability problem are recursively equivalent to it, hence decidable as well. These related problems are

- The *submarking reachability problem*: Given a subset $S' \subseteq S$ of places and a marking m, does there exist a marking $m' \in [m_0\rangle$ such that $m'(s) = m(s)$ for all $s \in S'$?
- The *zero-reachability problem*: Is the empty marking $\emptyset$ reachable from the initial marking m_0?
- The *single-place zero-reachability problem*: For a given place $s \in S$, does there exist a marking $m \in [m_0\rangle$ such that $m(s) = 0$?

A problem P is recursively reducible to a problem Q if a solution for P can be computed as a solution of a specific instance of Q. For instance, the zero-reachability problem is reducible to the reachability problem: simply set $m = \emptyset$ for the reachability problem. Similarly simple is to prove that the reachability problem is reducible to the submarking reachability problem: simply set $S' = S$ for the submarking reachability problem. Also the single-place zero-reachability problem is trivially reducible to the submarking reachability problem: simply set $S' = \{s\}$ and $m' = \emptyset$ for the submarking reachability problem. Following [Hack76b, Pet81], one can provide suitable net constructions proving that the submarking reachability problem is reducible to the zero-reachability problem, and also that the zero-reachability problem is reducible to the single-place zero-reachability problem.

Similarly, Hack also showed in [Hack74] that another, apparently more distant, problem can be reduced to the reachability problem: the *liveness problem*.

Definition 3.13. (Liveness) Given a P/T net system $N(m_0) = (S, A, T, m_0)$, a transition $t \in T$ is *live* if for each marking m reachable from m_0 there exists a marking m' reachable from m such that t is enabled at m': $\forall m \in [m_0\rangle\, \exists m' \in [m\rangle\, .m'[t\rangle$. The net system $N(m_0)$ is *live* if each of its transitions is live. □

In other words, a net is live if any of its transitions can always occur again in the future. The liveness problem for a net consists of deciding whether it is live. Since this problem is recursively equivalent to the reachability problem, it is also decidable. The complexity of the liveness problem for P/T Petri nets is open [EN94], although it is known that this problem is PSPACE-complete for safe nets [CEP95].

An interesting problem is reducible in polynomial time to the liveness problem: the *deadlock problem.*

Definition 3.14. (Deadlock problem) Given a P/T net system $N(m_0) = (S, A, T, m_0)$, a marking m is *dead* if it does not enable any transition, i.e., $\forall t \in T \ \neg(m[t\rangle)$. The net $N(m_0)$ has a *deadlock* if there exists a dead marking m reachable from m_0. The net $N(m_0)$ is *deadlock-free* if it has no deadlock. The deadlock problem for a net consists of deciding whether it has a deadlock. □

Since the liveness problem is decidable, and the deadlock problem is reducible to the liveness problem, it follows that also the deadlock problem is decidable. The proof of this fact, given below, follows [CEP95, Bus02].

Theorem 3.6. *The deadlock problem is reducible to the liveness problem.*

Proof. Given a P/T net system $N(m_0) = (S, A, T, m_0)$*, we construct a net system* $N'(m_0) = (S', A, T', m_0)$*, where* $S' = S \cup \{ok\}$ *and* $T' = T \cup \{t' \mid t \in T\} \cup \{live\}$ *where* $t' = (^\bullet t, l(t), ok)$ *and* $live = (ok, a, ok \oplus \bigoplus_{t \in T} {}^\bullet t)$ *for some* $a \in A$*. Hence,* N' *is just an extension of* N *with additional transitions and one additional place. Observe that any firing sequence of* N *is also a firing sequence of* N'*. Note also that the firing of transition live in* N' *enables all the transitions in* T'*.*

We show that $N(m_0)$ *is deadlock-free if and only if* $N'(m_0)$ *is live.*

$\Leftarrow$) *We actually prove the reverse implication: if* N *has a deadlock, then* N' *is not live. Suppose that* N*, by firing the transition sequence* σ*, reaches a dead marking* m*, i.e.,* $\forall t \in T, \neg(m[t\rangle)$*. Then, also* N' *can reach marking* m *by firing the same transition sequence* σ*. We have that* $m(ok) = 0$ *because no* t' *transition has been fired, hence the transition live is not enabled at* m*. Since each transition* t' *has the same pre-set as the corresponding transition* t*, also all these transitions are not enabled at* m *in* N'*. So, we can conclude that* m *is a dead marking for* N'*, hence* N' *is not live.*

$\Rightarrow$) *Now suppose that* N *is deadlock-free. Let* m *be a reachable marking in* N'*. Two cases may happen:*

- $m(ok) > 0$*. In such a case, transition live is enabled at* m *and, after its firing, every transition in* T' *is enabled. Note that if we produce a token in the place ok it will remain always marked, so that transition live will always be enabled and therefore* N' *is live.*
- $m(ok) = 0$*. In such a case, only transitions* $t \in T$ *have been fired to reach* m*, hence* m *is reachable also in* N*. As* N *has no reachable dead markings, there exists* $t \in T$ *such that* $m[t\rangle$ *in* N*. As* $^\bullet t = {}^\bullet t'$*, also* t' *is enabled at* m*:* $m[t'\rangle m'$ *in* N'*, with* $m'(ok) = 1$*. So, we are now back to the previous case.* □

Although a precise complexity measure of this problem is open, for safe nets it has been proved to be PSPACE-complete [CEP95].

3.4 Behavioral Equivalences

When can two Petri nets be considered equivalent? There is a wide range of possibilities and we are going to examine only a few of them.

Of course, equivalence notions for Petri nets include all those for LTSs we have discussed in Chapter 2. This because we can associate with a Petri net an LTS, called the *interleaving marking graph*, which describes its interleaving behavior: the states are the reachable markings and each LTS transition corresponds to the firing of a net transition. On the interleaving marking graph, we can define trace equivalence and bisimulation equivalence, as done on ordinary LTSs. Additionally, we can associate with a Petri net an STS, called the *step marking graph*, which describes its step behavior: the states are the reachable markings and each STS transition corresponds to the firing of a multiset of concurrently enabled net transitions [NT84]. These equivalences, albeit decidable for finite bounded P/T nets, are undecidable in general for unbounded finite P/T nets [Jan95, Esp98]. This negative undecidability result holds also for most equivalence relations defined in the literature for finite P/T nets, such as *ST*-bisimilarity [GV87, GL95, BG02], *history-preserving* bisimilarity [RT88, DDM89, GG89, BDKP91] and *hereditary* history-preserving bisimilarity [JNW96, FH99]; these non-interleaving equivalences, more discriminating than step bisimilarity, observe to some extent the causality relation among transitions, while abstracting away the structure of the markings; even if they are interesting, their description is outside the scope of this book. A relevant exception is *net isomorphism*, a generalization of LTS isomorphism, which is a decidable equivalence for finite P/T nets, although exponential in the size of the net. Its major drawback is that it is very concrete, so that many nets that no observer can tell apart are considered not equivalent. Nonetheless, net isomorphism turns out to be very useful for the many results we are going to prove in the next chapters, in particular the so-called *representability theorems*, showing that specific classes of finite P/T nets can be represented, up to net isomorphism, by specific process algebras. So, let us start our brief overview of behavioral equivalences with net isomorphism.

3.4.1 Net Isomorphism

Definition 3.15. (Net isomorphism) Given two P/T nets $N_1 = (S_1, A, T_1)$ and $N_2 = (S_2, A, T_2)$, we say that N_1 and N_2 are *isomorphic* – denoted $N_1 \cong N_2$ – if there exists a bijection $f : S_1 \to S_2$, homomorphically extended to markings,[3] such that $(m, \ell, m') \in T_1$ if and only if $(f(m), \ell, f(m')) \in T_2$.

Two P/T net systems $N_1(m_1)$ and $N_2(m_2)$ are *rooted isomorphic* – denoted $N_1(m_1) \cong_r N_2(m_2)$ – if the isomorphism $f : S_1 \to S_2$ ensures, additionally, that $f(m_1) = m_2$. □

[3] This means that f is applied element-wise to each component of the marking, i.e., $f(m_1 \oplus m_2) = f(m_1) \oplus f(m_2)$.

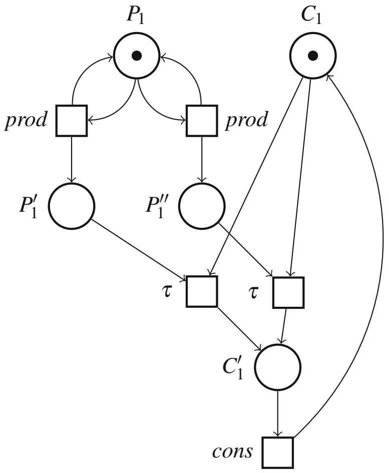


Fig. 3.11 The Petri net for another unbounded producer/consumer UPC_1

It is an easy exercise to prove that net isomorphism $\cong$ and rooted net isomorphism $\cong_r$ are equivalence relations, i.e., they are reflexive, symmetric and transitive.

Example 3.7. Consider the Petri net system $N'(m_0') = (S', A, T', m_0')$, where $S' = \{s_1', s_2', s_4'\}, A = \{a, b\}, T' = \{(s_1', b, s_4'), (s_4', a, s_1')\}$ and $m_0' = s_1' \oplus 2s_2'$. It is easy to check that $N'(m_0')$ is rooted isomorphic to the net system in Figure 3.7(b), because $f(s_i) = s_i'$ for $i = 1, 2, 4$ is the required bijection. □

Note that the nets in Figure 3.3 (producer/consumer UPC) and Figure 3.11 (producer/consumer UPC_1) are not isomorphic, even if it seems that there is no observable reason to tell them apart: UPC_1 is just a nondeterministic variant of UPC. As a matter of fact, net isomorphism is often a too-concrete equivalence relation. Nonetheless, $\cong$ is decidable for finite P/T nets. The problem of determining whether two finite net systems are (rooted) isomorphic is a generalization of the problem of checking whether two finite graphs are isomorphic, as Petri nets can be equivalently represented as bipartite graphs (i.e., graphs with two kinds of nodes – places and transitions – as in the pictorial representation of nets). This problem is in the class NP, which is not known to be in P, and the best known algorithm is exponential in the number of nodes [BKL83].

3.4.2 Interleaving Semantics

It seems natural that a Petri net can be observed as an interleaving device, where only one transition is performed at a time. This is what we already assumed implicitly when defining the firing sequences of a net (Definition 3.5) and even the token game (Definition 3.4). Generalizing this reasoning, we can associate a labeled transition

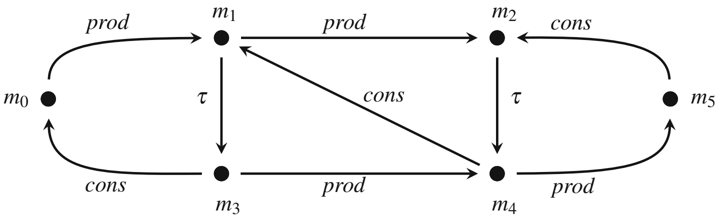


Fig. 3.12 The *IMG* for the 2-bounded producer/consumer $2PC$ whose net is in Figure 3.2(a)

system with a net system, where the states are the reachable markings and each transition is labeled with the label of the corresponding net transition. Such an LTS is called the *interleaving marking graph*, usually abbreviated as *IMG*.

Definition 3.16. (Interleaving marking graph) The *interleaving marking graph* of $N(m_0) = (S, A, T, m_0)$ is the rooted LTS $IMG(N(m_0)) = ([m_0\rangle, A, \rightarrow, m_0)$, where m_0 is the initial state, the set of states is given by the set $[m_0\rangle$ of reachable markings, and the transition relation $\rightarrow \subseteq \mathscr{M}_{fin}(S) \times A \times \mathscr{M}_{fin}(S)$ is defined by $m \xrightarrow{a} m'$ if and only if there exists a transition $t \in T$ such that $m[t\rangle m'$ and $l(t) = a$. □

Continuing Example 3.3, the interleaving marking graph for the net system in Figure 3.2(a) for the 2-bounded producer/consumer $2PC$ is the deadlock-free, finite-state LTS depicted in Figure 3.12. Now we list some obvious facts.

Proposition 3.7. *Given a P/T net system $N(m_0)$, the following hold:*

1. *if $N(m_0)$ is finite and bounded, then $IMG(N(m_0))$ is a finite-state LTS;*
2. *if $N(m_0)$ is finite but not bounded, then $IMG(N(m_0))$ is not finite-state;*
3. *if $N(m_0)$ is bounded and $IMG(N(m_0))$ is not finite-state, then N is infinite;*
4. *if $IMG(N(m_0))$ is finite-state, then the dynamically reachable subnet $Net_d(N(m_0))$ is finite and bounded;*
5. *if $N(m_0)$ is distinct, then $IMG(N(m_0))$ is deterministic.* □

The proof of (1) above is ensured by Proposition 3.1(7), which states that the set of reachable markings of a finite, bounded net is finite, and by the fact that the set of net transitions is finite.

As a consequence, also (2) and (3) hold, trivially. As an example for (2), the net in Figure 3.5(a) is an unbounded BPP net, whose *IMG* has infinitely many states and is isomorphic to the LTS in Figure 2.3(b), up to renaming of action *inc* to a and *dec* to b. As an example for (3), consider the net $N(m_0) = (S, A, T, m_0)$, where $S = \{s_i \mid i \in \mathbb{N}\} \cup \{s\}$, $A = \{a, b\}$, $T = \{(s_i, b, s_{i+1}) \mid i \in \mathbb{N}\} \cup \{(s, a, \emptyset)\}$ and $m_0 = s \oplus s_0$; clearly, $N(m_0)$ is safe, and its associated *IMG* is not finite-state; and in fact N is infinite.

About (4), if $Net_d(N(m_0))$ is not finite or not bounded, of course $IMG(N(m_0))$ cannot be finite-state. As an example of a net which is infinite, but such that its *IMG* is finite-state, consider the net N above, but with initial marking $m_1 = s$; even if N

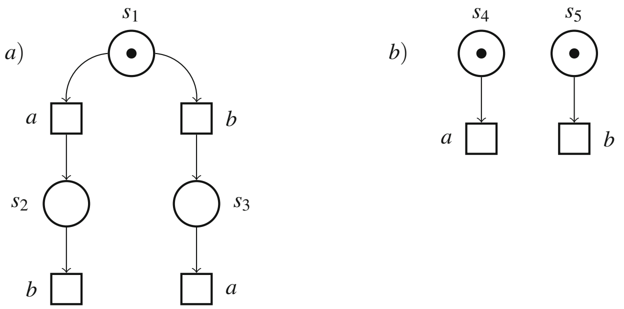


Fig. 3.13 Two interleaving trace equivalent net systems

is infinite, its *IMG* is finite-state and can be generated by its dynamically reachable subnet $Net_d(N(m_1)) = (\{s\}, \{a\}, \{(s,a,\emptyset)\}, s)$, which is finite and bounded.

About (5), if $IMG(N(m_0))$ is not deterministic, then there is a reachable marking m and two transitions t_1 and t_2 such that $l(t_1) = l(t_2)$, $m[t_1\rangle m_1$, $m[t_2\rangle m_2$ and $m_1 \neq m_2$; therefore, $t_1 \neq t_2$ and so $N(m_0)$ is not distinct.

Since $IMG(N(m_0))$ is a labeled transition system, we can define over it the whole spectrum of equivalences and preorders we have defined in Chapter 2. Here we recall only a few of them.

Definition 3.17. (Interleaving trace equivalence) Given a net system $N(m_0) = (S, A, T, m_0)$, we write the set of traces of $N(m_0)$ as the set

$$Tr(IMG(N(m_0))) = \{\sigma \in A^* \mid \exists m \in [m_0\rangle .\ m_0 \xrightarrow{\sigma}{}^* m\}.$$

With abuse of notation, we often write such a set as $Tr(N(m_0))$ or simply $Tr(m_0)$ when the net N is clear from the context. Two P/T net systems $N_1(m_1)$ and $N_2(m_2)$ are trace equivalent if $Tr(N_1(m_1)) = Tr(N_2(m_2))$.

This definition can be extended to any state of $IMG(N(m_0))$ as follows: given a marking $m \in [m_0\rangle$, we write $Tr(m) = \{\sigma \in A^* \mid \exists m' \in [m\rangle .\ m \xrightarrow{\sigma}{}^* m'\}$. Two markings $m_1, m_2 \in [m_0\rangle$ are trace equivalent if $Tr(m_1) = Tr(m_2)$. □

Example 3.8. The two net systems in Figure 3.13 are clearly trace equivalent: the set of traces is $\{\varepsilon, a, b, ab, ba\}$ for either of them; the associated *IMG*s, composed of four states, are isomorphic. Similarly, the three net systems in Figure 3.14 are trace equivalent, and also trace equivalent to those in Figure 3.13; however, the associated *IMG*s are not isomorphic, in general. For instance, the *IMG* for the net system in Figure 3.14(c) has six states, while the *IMG* for the net system in Figure 3.14(a) has five states. Finally, the two nets in Figure 3.7(a) and (b) are trace equivalent: the infinite set of traces is $\{\varepsilon, b, ba, bab, baba, \ldots\} = (ba)^*(b|\varepsilon)$. Note that, in this case, the two *IMG*s are isomorphic. □

Trace equivalence is decidable for *distinct*, finite P/T nets (i.e., finite nets whose transitions have distinct labels), because Hack proved in [Hack76b] that this equivalence problem is reducible to the marking reachability problem, which is decidable.

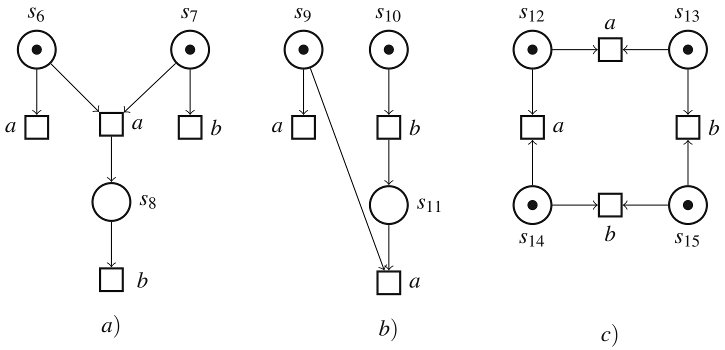


Fig. 3.14 Some other, rather different, interleaving trace equivalent net systems [GV87]

However, Hack also proved that, in general, trace equivalence is undecidable for finite (not distinct) P/T nets. Jančar [Jan95] strengthened this negative result, proving that undecidability holds for finite P/T nets with at least two unbounded places. The trace equivalence problem for finite P/T nets with only one unbounded place is open [EN94]. Trace equivalence is undecidable even for (not distinct) BPP nets [Hir93]. Of course, it is decidable for finite bounded nets, as the resulting *IMG* is finite-state. In particular, trace equivalence is EXPSPACE-complete for safe nets [JM96].

Definition 3.18. (Petri net language) Given a P/T net system $N(m_0) = (S, A, T, m_0)$, such that $B = A \setminus \{\tau\}$, we write the set of its weak completed traces as

$$WCTr(IMG(N(m_0))) = \{\sigma \in B^* \mid \exists m \in [m_0\rangle.\ m_0 \overset{\sigma}{\Longrightarrow} m \overset{\ell}{\not\Rightarrow} \text{ for all } \ell \in B\}.$$

The Petri net language $L[N(m_0)]$ recognized by the net system $N(m_0)$ is exactly the set $WCTr(IMG(N(m_0)))$ of its weak completed traces. □

Peterson in [Pet81] studies a variety of possible definitions of Petri net language; our chosen definition corresponds to his T-type language, where a trace belongs to the language only if the reached marking is a terminal one (either a deadlock or a livelock). There, it is shown that if, no transition in $N(m_0)$ is labeled with the invisible action τ, then its recognized language $L[N(m_0)]$ is context-dependent; instead, if τ is allowed, then [Pet81] states that determining whether $L[N(m_0)]$ is context-dependent is an open problem.[4] The problem of checking whether two finite P/T net systems recognize the same Petri net language is undecidable.

Example 3.9. The language recognized by the net in Figure 3.5(b) is the regular language b^*ab^*. In fact, the only terminal marking is s_5 and the token in s_4 can disappear only by performing the τ-labeled transition. Hence, in order to reach the terminal marking, first the b-labeled transition can be performed any number of times;

[4] In the classification of Figure 1.9, the class of Petri net languages is a subset of the class of context-dependent languages; to be precise, this has been proved in [Pet81] only for finite P/T nets without τ-labeled transitions.

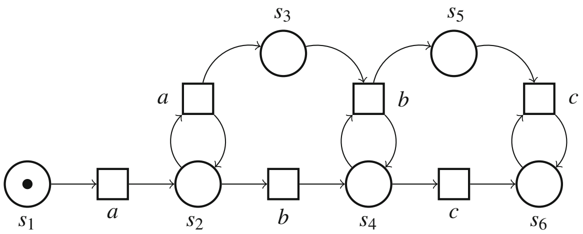


Fig. 3.15 A net system recognizing a context-dependent language

then, either the τ-labeled transition is performed (so that b cannot be performed anymore) and, finally, one instance of the a-labeled transition must be executed; or one instance of the a-labeled transition is performed, then the b-labeled transition is performed again any number of times, and finally the τ-labeled transition must be performed. □

Example 3.10. Figure 3.15 shows a net whose recognized language is the context-dependent language $\{a^n b^m c^m \mid n \geq m \geq 1\}$. As a matter of fact, after firing the first a, one token is present in place s_2; now an arbitrary number of the other a-labeled transitions can be fired, depositing one token each time into s_3; then, the token in s_2 can fire the first b, reaching place s_4; now, a number of b-labeled transitions can be fired by consuming the tokens in s_3 and producing tokens in s_5; the token in s_4 can, at any time, perform the first c, even if some tokens are still present in s_3, reaching place s_6; finally, the tokens in s_5 can be used to perform a number of c-labeled transitions equal to the number of tokens deposited into s_5. Summing up, the number of c's and b's is always the same, while the number of a's may be larger, as the terminal configurations are of the form $s_6 \oplus (n-m) \cdot s_3$. (See also Examples 6.5 and 8.5.) □

Definition 3.19. (Interleaving bisimulation equivalence) The P/T systems $N_1(m_1)$ and $N_2(m_2)$ are *interleaving bisimilar* – denoted $N_1(m_1) \sim N_2(m_2)$, or $m_1 \sim m_2$ when the nets are clear from the context – if and only if there exists a strong bisimulation $R \subseteq [m_1\rangle \times [m_2\rangle$ over the LTSs $IMG(N_1(m_1))$ and $IMG(N_2(m_2))$ such that $(m_1, m_2) \in R$.

Given a P/T net system $N(m_0)$ and a rooted LTS $TS(q_0) = (Q, A, \rightarrow, q_0)$, we say that N and TS are interleaving bisimilar, denoted $N(m_0) \sim TS(q_0)$ or even $m_0 \sim q_0$, if and only if there exists a strong bisimulation $R \subseteq [m_0\rangle \times Q$ over the LTSs $IMG(N(m_0))$ and $TS(q_0)$ such that $(m_0, q_0) \in R$. □

Example 3.11. The two net systems in Figure 3.13 are clearly interleaving bisimilar, as their associated *IMG*s are actually even isomorphic.

Now, consider the nets in Figure 3.14(a) and (b). It is an easy exercise to check that relation $R = \{(s_6 \oplus s_7, s_9 \oplus s_{10}), (s_7, s_{10}), (s_6, s_9 \oplus s_{11}), (s_8, s_{10}), (\emptyset, s_{11}), (\emptyset, \emptyset)\}$ is an interleaving bisimulation. It is not difficult to realize that all the five nets in Figures 3.13 and 3.14 are pairwise interleaving bisimilar. □

Example 3.12. Consider the two versions of the unbounded producer/consumer of Figures 3.3 and 3.11. The relation

$$\begin{aligned} R = \ & \{(P \oplus C \oplus (k_1 + k_2) \cdot P', P_1 \oplus C_1 \oplus k_1 \cdot P_1' \oplus k_2 \cdot P_1'') \ \big| \ k_1, k_2 \geq 0\} \cup \\ & \{(P \oplus C' \oplus (k_1 + k_2) \cdot P', P_1 \oplus C_1' \oplus k_1 \cdot P_1' \oplus k_2 \cdot P_1'') \ \big| \ k_1, k_2 \geq 0\} \end{aligned}$$

is a bisimulation relating the initial markings: $(P \oplus C, P_1 \oplus C_1) \in R$ when $k_1 = 0 = k_2$. Let us check that R is a bisimulation. If $P \oplus C \oplus (k_1 + k_2) \cdot P'$ performs *prod*, reaching the marking $P \oplus C \oplus (k_1 + k_2 + 1) \cdot P'$, then $P_1 \oplus C_1 \oplus k_1 \cdot P_1' \oplus k_2 \cdot P_1''$ can reply, by reaching the marking $P_1 \oplus C_1 \oplus (k_1 + 1) \cdot P_1' \oplus k_2 \cdot P_1''$ and the two reached markings are related. If $P \oplus C \oplus (k_1 + k_2) \cdot P'$ performs τ, because $k_1 + k_2 > 0$, reaching the marking $P \oplus C' \oplus (k_1 + k_2 - 1) \cdot P'$, then $P_1 \oplus C_1 \oplus k_1 \cdot P_1' \oplus k_2 \cdot P_1''$ can reply, by reaching the marking $P_1 \oplus C_1' \oplus (k_1 - 1) \cdot P_1' \oplus k_2 \cdot P_1''$ or $P_1 \oplus C_1' \oplus k_1 \cdot P_1' \oplus (k_2 - 1) \cdot P_1''$, and the two reached markings are related. If $P \oplus C' \oplus (k_1 + k_2) \cdot P'$ performs *cons*, reaching $P \oplus C \oplus (k_1 + k_2) \cdot P'$, then $P_1 \oplus C_1' \oplus k_1 \cdot P_1' \oplus k_2 \cdot P_1''$ can reply, reaching $P_1 \oplus C_1 \oplus k_1 \cdot P_1' \oplus k_2 \cdot P_1''$, and the two reached markings are related. The reader can complete the check that R is indeed a bisimulation. □

Interleaving bisimulation equivalence coincides with trace equivalence over *distinct* P/T nets, because, by Proposition 3.7(5), a distinct P/T net generates a deterministic *IMG*, and we know from Chapter 2 that over deterministic LTSs the two equivalences coincide. Since trace equivalence is decidable over distinct finite P/T nets, then also interleaving bisimulation equivalence is decidable over such a restricted class of finite P/T nets. However, bisimulation equivalence is undecidable in general for finite (not distinct) P/T nets, even with only two unbounded places [Jan95]; this negative results holds also for finite CCS nets, as proved in [GV15] (Section 3.5.4) directly over the FNC calculus (see Chapter 6), which represents precisely that subclass of finite P/T nets. In [BGJ10] it has been proved that interleaving bisimulation equivalence is decidable if the finite P/T net has at most one unbounded place. Interleaving bisimulation equivalence is decidable also for BPP nets [CHM93], which may have finitely many unbounded places but transitions with singleton pre-set only; more recently, this problem has been proved PSPACE-complete [Jan03]. Of course, interleaving bisimulation equivalence is decidable for bounded finite P/T nets, as the associated *IMG*s are finite-state; in particular, for safe nets this equivalence problem is DEXPTIME-complete [JM96].

Although interleaving bisimulation equivalence is undecidable in general over finite P/T nets, some related problems have a positive solution:

- *Strong bisimilarity with a finite-state system*: given a finite-state LTS, rooted in q, and a finite P/T net $N(m_0)$, we can decide whether q and m_0 are strongly bisimilar [JM95]. This problem is interesting because it enables us to perform equivalence checking between the complex behavior of an infinite-state implementation (the finite P/T net) and its finite-state specification.
- *Strong regularity*: given a finite P/T net $N(m_0)$, we can decide whether there exists a finite-state LTS, rooted in q, such that m_0 and q are bisimilar [JE96].

All these properties are difficult to check in practice, as they are at least exponential. However, for BPP nets the complexity is better: strong bisimilarity with a finite-state system can be checked in polynomial time [KS05], while strong regularity is PSPACE-complete [Kot05].

The definition of weak bisimilarity over two P/T net systems $N_1(m_1)$ and $N_2(m_2)$ is as expected: we check whether there exists a weak bisimulation over $IMG(N_1(m_1))$ and $IMG(N_2(m_2))$ relating the initial markings m_1 and m_2. Of course, also weak bisimilarity is undecidable for finite P/T nets. In the case of BPP nets, weak bisimilarity has been proved decidable in some restricted cases, e.g., when one of the two processes is finite-state [JKM01, KM02], but the problem in the general case is still open, even if a conjecture about its decidability has been proposed in [CHL11].

3.4.3 Step Semantics

Given a P/T net $N = (S, A, T)$ and a marking m, we say that two transitions $t_1, t_2 \in T$ are *concurrently enabled* at m if ${}^\bullet t_1 \oplus {}^\bullet t_2 \subseteq m$. The *concurrent firing* of these two transitions produces the marking $m' = (m \ominus ({}^\bullet t_1 \oplus {}^\bullet t_2)) \oplus (t_1^\bullet \oplus t_2^\bullet)$. We denote this fact by $m[\{t_1, t_2\}\rangle m'$. For instance, if we consider the net in Figure 3.13(b), we can easily realize that its two transitions are concurrently enabled: $s_4 \oplus s_5[\{t_1, t_2\}\rangle \emptyset$, where $t_1 = (s_4, a, \emptyset)$ and $t_2 = (s_5, b, \emptyset)$. Note that in sequential FSMs, such as those in Figure 3.13(a) and Figure 3.4(a), there is no possibility to have two concurrently enabled transitions at any reachable marking. It is also possible that the same transition is *self-concurrent* at some marking m, meaning that two or more occurrences of it are concurrently enabled at m. For instance, if we consider the net in Figure 3.4(b) with initial marking $m_0 = s_3 \oplus 2 \cdot s_4$, then $m_0[\{t_2, t_2\}\rangle s_3$, where $t_2 = (s_4, b, \emptyset)$.

We can generalize the definition of concurrently enabled transitions to a finite, nonempty multiset G over the set T, called a *step*. A step $G : T \to \mathbb{N}$ is enabled at marking m if ${}^\bullet G \subseteq m$, where ${}^\bullet G = \bigoplus_{t \in T} G(t) \cdot {}^\bullet t$ and $G(t)$ denotes the number of occurrences of transition t in the step G. The execution of a step G enabled at m produces the marking $m' = (m \ominus {}^\bullet G) \oplus G^\bullet$, where $G^\bullet = \bigoplus_{t \in T} G(t) \cdot t^\bullet$. This is written $m[G\rangle m'$. We sometimes refer to this as the *concurrent token game*, in opposition to the *sequential* token game of Definition 3.4.

Proposition 3.8. *Given a P/T net N, a marking m and a step $G = G_1 \oplus G_2$, where $G_i \neq \emptyset$ for $i = 1, 2$, if $m[G_1 \oplus G_2\rangle m'$, then there exist two markings m_1 and m_2 such that $m[G_1\rangle m_1[G_2\rangle m'$ and $m[G_2\rangle m_2[G_1\rangle m'$.*

Proof. *If $G_1 \oplus G_2$ is enabled at m, then ${}^\bullet G_1 \oplus {}^\bullet G_2 \subseteq m$. Therefore, also G_1 and G_2 are enabled at m. The firing of G_1 at m produces the marking $m_1 = (m \ominus {}^\bullet G_1) \oplus G_1^\bullet$, which still enables G_2, because ${}^\bullet G_1 \oplus {}^\bullet G_2 \subseteq m$ ensures that ${}^\bullet G_2 \subseteq m \ominus {}^\bullet G_1$ and $m \ominus {}^\bullet G_1 \subseteq m_1$. The firing of G_2 at m_1 produces the marking $m' = (m_1 \ominus {}^\bullet G_2) \oplus G_2^\bullet = m \ominus ({}^\bullet G_1 \oplus {}^\bullet G_2) \oplus (G_1^\bullet \oplus G_2^\bullet)$. Similarly, the firing of G_2 at m produces the marking $m_2 = (m \ominus {}^\bullet G_2) \oplus G_2^\bullet$, which still enables G_1, because ${}^\bullet G_1 \oplus {}^\bullet G_2 \subseteq m$ ensures that ${}^\bullet G_1 \subseteq m \ominus {}^\bullet G_2$ and $m \ominus {}^\bullet G_2 \subseteq m_2$. And the firing of G_1 at m_2 produces the marking $(m_2 \ominus {}^\bullet G_1) \oplus G_1^\bullet = m'$, as required.* □

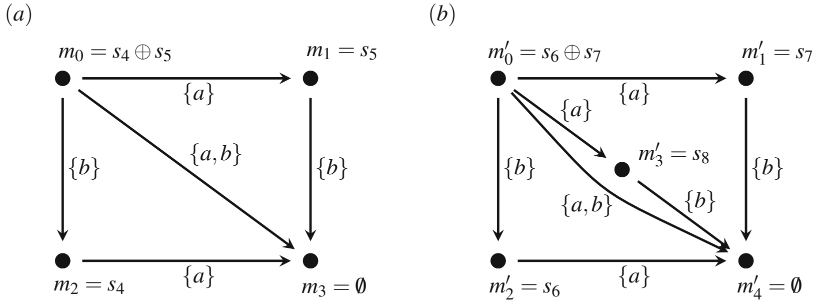


Fig. 3.16 In (a) the *SMG* for the net in Figure 3.13(b); in (b) the *SMG* for the net in Figure 3.14(a)

A consequence of the above proposition is that a step can be simulated *sequentially*. If a step G can fire, $m[G\rangle m'$, then for any linearization of its transitions $t_1, t_2, \dots t_n$ (i.e., such that $G(t) = |\{i \mid 1 \leq i \leq n \wedge t_i = t\}|$ for all $t \in T$), there exist markings $m_1, m_2, \dots, m_{n-1}$ such that $m[t_1\rangle m_1[t_2\rangle m_2 \dots m_{n-1}[t_n\rangle m'$. The obvious consequence is that the set of markings we can reach by the concurrent token game is exactly the same set we can compute by the sequential token game.

Starting from the definition of the concurrent token game, we can associate with a P/T net system $N(m_0)$ a step transition system $SMG(N(m_0))$, called the *step marking graph*, which shows also the concurrency of transitions in the net system.

Definition 3.20. (Step marking graph) The *step marking graph* associated with a P/T net system $N(m_0) = (S, A, T, m_0)$ is the rooted STS

$SMG(N(m_0)) = ([m_0\rangle, \mathscr{M}_{fin}(A), \longrightarrow_s, m_0)$

where $\longrightarrow_s \subseteq \mathscr{M}_{fin}(S) \times \mathscr{M}_{fin}(A) \times \mathscr{M}_{fin}(S)$ is defined by $m \xrightarrow{B}_s m'$ iff there exists a step G such that $m[G\rangle m'$ and $B = l(G)$, and the labeling function l is extended to multisets of transitions in the obvious way: $l(G)(a) = \sum_{t_i \in dom(G).l(t_i)=a} G(t_i)$. □

Proposition 3.9. (Fully concurrent) *For any P/T net system $N(m_0)$, its associated step marking graph $SMG(N(m_0))$ is fully concurrent.*

Proof. A direct consequence of Proposition 3.8. □

Definition 3.21. (Step bisimilarity) Two P/T systems $N_1(m_1)$ and $N_2(m_2)$ are *step bisimilar* (denoted by $N_1(m_1) \sim_{step} N_2(m_2)$ or simply $m_1 \sim_{step} m_2$) if and only if there exists a strong bisimulation $R \subseteq [m_1\rangle \times [m_2\rangle$ over the STSs $SMG(N_1(m_1))$ and $SMG(N_2(m_2))$ such that $(m_1, m_2) \in R$. □

Example 3.13. The step marking graphs associated with the nets in Figure 3.13(b) and Figure 3.14(a) are outlined in Figure 3.16. It is easy to see that, albeit not isomorphic, these two *SMG*s are strongly bisimilar, hence these two nets are step bisimilar. The reader may check that all the nets in Figure 3.14 are pairwise step bisimilar, first by constructing their associated *SMG*s and then by exhibiting suitable strong bisimulations. On the contrary, the *SMG*s for the two nets in Figure 3.13 are not

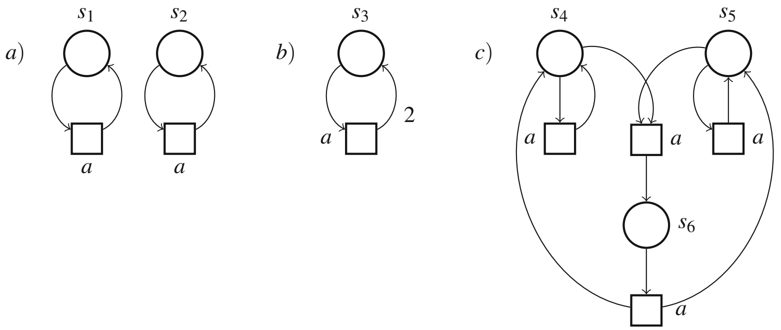


Fig. 3.17 Three unmarked nets

bisimilar. Note that the step $s_4 \oplus s_5 \xrightarrow{\{a,b\}}_s \emptyset$ cannot be matched by s_1. However, we noticed in Example 3.11 that these two nets are interleaving bisimilar. □

Example 3.14. Consider the nets in Figure 3.17. It is an easy exercise to check that, although $s_1 \sim s_1 \oplus s_2$, we have that $s_1 \not\sim_{step} s_1 \oplus s_2$. Moreover, although $s_1 \sim s_3$, we have that $s_1 \not\sim_{step} s_3$ and $s_1 \oplus s_2 \not\sim_{step} s_3$. Finally, although $s_1 \sim_{step} s_4$ and $s_2 \sim_{step} s_5$, we have that $s_1 \oplus s_2 \not\sim_{step} s_4 \oplus s_5$. □

The examples above show the expected fact that step semantics is finer than interleaving semantics, in particular that step bisimulation equivalence is finer than interleaving bisimulation equivalence (but the same holds for any other equivalence, e.g., step trace equivalence is finer than interleaving trace equivalence).

Lemma 3.4. *Given a net system $N(m_0)$, $m \xrightarrow{a} m'$ is a transition in the $IMG(N(m_0))$ if and only if $m \xrightarrow{\{a\}}_s m'$ is a transition in the $SMG(N(m_0))$.*

Proof. *Any transition $m \xrightarrow{a} m'$ in $IMG(N(m_0))$ is due to a transition t such that $m[t\rangle m'$ and $l(t) = a$. Therefore, also $m[\{t\}\rangle m'$ and so $m \xrightarrow{\{a\}}_s m'$ is a transition in $SMG(N(m_0))$. This means that all the interleaving transitions are represented in the SMG. Conversely, with a similar argument, one can show that all the singleton-labeled step transitions are represented in the IMG.* □

Proposition 3.10. *Given two net systems $N_1(m_1)$ and $N_2(m_2)$, if $N_1(m_1) \sim_{step} N_2(m_2)$, then $N_1(m_1) \sim N_2(m_2)$.*

Proof. *By Lemma 3.4, for any net system $N(m_0)$, the LTS $IMG(N(m_0))$ is isomorphic (up to renaming of a to $\{a\}$) to the singleton-labeled subpart of the STS $SMG(M(m_0))$. Therefore, if $N_1(m_1) \sim_{step} N_2(m_2)$, they are necessarily bisimilar over their singleton-labeled transitions, i.e., over their IMGs.* □

Step bisimilarity is also undecidable for finite P/T nets with at least two unbounded places: a simple observation, described below and due to [Esp98], suffices to show that Jančar's undecidability proof of interleaving bisimilarity [Jan95] applies to, besides step bisimilarity, essentially all the non-interleaving equivalences proposed in the literature, such as ST-bisimilarity [GV87, GL95, BG02], *history-preserving* bisimilarity [RT88, GG89, DDM89, BDKP91] and *hereditary* history-preserving bisimilarity [JNW96, FH99] (see also [PRS92]). Let us call *sequential* a Petri net which does not offer any concurrent behavior, i.e., whose *SMG* is labeled only with singletons. The proof of undecidability of bisimulation equivalence in [Jan95] is based on the comparison of two sequential finite Petri nets; hence, bisimulation equivalence is undecidable even for the subclass of *sequential finite* Petri nets. All the non-interleaving bisimulation equivalences collapse to interleaving bisimilarity over sequential Petri nets. Hence, the proof in [Jan95] applies to all non-interleaving bisimulation equivalences as well. For BPP, the problem of checking step bisimilarity is decidable, because bisimulation equivalence is decidable for BPP [CHM93] and step bisimilarity is just ordinary bisimulation equivalence over the *SMG*s of the BPP nets under scrutiny, whose state space is the same as that of their associated *IMG*s. On bounded finite P/T nets, step bisimilarity is decidable, because the associated *SMG*s are finite-state. About other non-interleaving equivalences, such as history-preserving bisimilarity, we have that they are decidable (DEXPTIME-complete) for finite safe nets [JM96, Vog91], and also can be decided in polynomial time for BPP [FJLS10].

3.5 Nonpermissive Petri Nets

In this section, we present a Petri net model more general than P/T nets, that constitutes also a generalization of P/T nets with *inhibitor arcs* [FA73, Hack76a, Pet81, JK95, Bus02]. The distinguishing feature of this model is that it is not permissive (see Remark 3.4): a transition t enabled at m may be disabled at a larger marking m' (i.e., such that $m \subseteq m'$) because m' can contain additional tokens inhibiting the firability of t.

Definition 3.22. (Nonpermissive net) A *Nonpermissive P/T Petri net* (or NP/T net, for short) is a tuple $N = (S, D, A, T)$ where

- S is the countable set of *places*, ranged over by s (possibly indexed),
- $D \subseteq S$ is the set of *testable* places,
- $A \subseteq Lab$ is the countable set of *labels*, ranged over by ℓ (possibly indexed), and
- $T \subseteq (\mathscr{M}_{fin}(S) \setminus \{\emptyset\}) \times \mathscr{P}_{fin}(D) \times A \times \mathscr{M}_{fin}(S)$ is the countable set of *transitions*, ranged over by t (possibly indexed), such that for each $\ell \in A$ there exists a transition $t \in T$ of the form (m, I, ℓ, m').

Given a transition $t = (m, I, \ell, m')$, we use the notation

- $^\bullet t$ to denote its *pre-set* m (which cannot be an empty multiset) of tokens to be consumed;

- $^\circ t$ to denote its *neg-set* I (which can be an empty set) of testable places where no further tokens can be present besides those required by $^\bullet t$;
- $l(t)$ for its *label* ℓ, and
- $t^\bullet$ to denote its *post-set* m' of tokens to be produced.

Hence, a transition t can be also represented as $(^\bullet t, {}^\circ t) \xrightarrow{l(t)} t^\bullet$. The *neg-set* $^\circ t$ of a transition t represents the set of places to be "tested for absence" of *additional* tokens. If $s \in {}^\circ t \cap dom(^\bullet t)$, then this means that t can be executed only if the tokens in s are precisely $^\bullet t(s)$; if $s \in {}^\circ t$ but $s \not\in dom(^\bullet t)$, then t can be executed only if s contain no tokens at all. This changes the definition of enabling: a transition t is enabled at m, written $m[t\rangle$, if $^\bullet t(s) \leq m(s)$ for all $s \in dom(^\bullet t)$ (the tokens to be consumed are available, as for P/T nets) and, additionally, $^\bullet t(s) = m(s)$ for all $s \in {}^\circ t$. As a matter of fact, this additional constraint means that, for all $s \in {}^\circ t$, the tokens in s for m are exactly as many as are required by t; note that if $^\bullet t(s) = 0$, then the condition $m(s) = {}^\bullet t(s) = 0$ corresponds to the classic inhibiting condition (test for absence of tokens in place s). The firing of a transition t enabled at m produces the marking $m' = (m \ominus {}^\bullet t) \oplus t^\bullet$, as for P/T nets; this is denoted $m[t\rangle m'$, as usual.

An NP/T system is a tuple $N(m_0) = (S, A, T, m_0)$, where (S, A, T) is an NP/T net and m_0 is the *initial marking*. □

Note that P/T nets, as in Definition 3.2, can be seen as the subclass of NP/T nets with the condition that $D = \emptyset$ and so $^\circ t = \emptyset$ for all $t \in T$. Note also that P/T nets *with inhibitor arcs* can be seen as the subclass of NP/T nets with the condition that $^\circ t \cap dom(^\bullet t) = \emptyset$ for all $t \in T$.

We adopt the following drawing convention for NP/T nets: a testable place is represented by a grey-colored circle. Moreover, an arc connecting a testable place to a transition may be labeled not only by a number (1 is the default value, in case the number is omitted), but also by a number prefixed by the special symbol !. If $s \in {}^\bullet t$, $s \not\in {}^\circ t$, then the arc from s to t is normally labeled with $^\bullet t(s)$, even if $s \in D$. If $s \in {}^\circ t \cap dom(^\bullet t)$, then the arc from s to t is labeled by $!^\bullet t(s)$, i.e., the number $^\bullet t(s)$ is prefixed by the symbol ! to express that *exactly* that number of tokens are to be present in the current marking m to enable t. If $s \in {}^\circ t$ but $s \not\in dom(^\bullet t)$, then a !0-labeled arc is included from s to t, meaning that no tokens can be present in s in the current marking m to enable t.

Figure 3.18 shows two NP/T nets, where s_1 and s_5 are the only two testable places. The net on the left can fire the a-labeled transition only if in the current marking m there is exactly one token in place s_1, which will then be consumed; the b-labeled transition can be performed if in m two or more tokens are present in s_1, and two of them will be consumed. Note that these two transitions are mutually exclusive: if one token is present in s_1, then only the a-labeled transition can fire, while if two or more tokens are present, then only the b-labeled transition can fire. The system in Figure 3.18(b) shows a net such that if the initial marking is s_4, then the a-labeled transition can be executed because no token is present in s_5; this transition can be performed any number of times, until the b-labeled transition is performed; this has the effect of producing one token in s_5, so that, from then on, only the b-labeled transition can be performed.

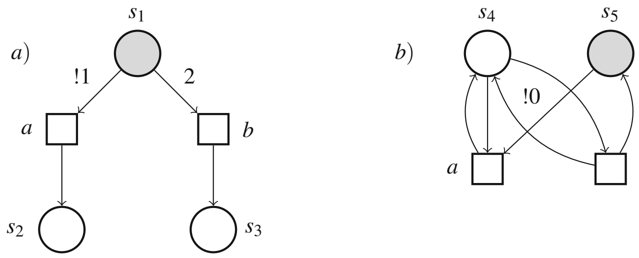


Fig. 3.18 Two simple NP/T systems

Definition 3.23. (Dynamically reachable subnet) Given an NP/T system $N(m_0) = (S, D, A, T, m_0)$, the *dynamically reachable subnet* $Net_d(N(m_0))$ is (S', D', A', T', m_0), where

$$\begin{aligned} T' &= \{t \in T \mid \exists m \in [m_0\rangle \text{ such that } m[t\rangle\},\\ S' &= \{s \in S \mid \exists m \in [m_0\rangle \text{ such that } m(s) \geq 1 \text{ or } \exists t \in T' \text{ such that } s \in {}^{\circ}t\},\\ D' &= D \cap S',\\ A' &= \{\ell \mid \exists t \in T' \text{ such that } l(t) = \ell\}. \end{aligned}$$

An NP/T net system $N(m_0) = (S, D, A, T, m_0)$ is *dynamically reduced* if $Net_d(N(m_0)) = N(m_0)$, i.e., the net system is equal to its dynamically reachable subnet. □

Definition 3.24. (Statically reachable subnet) Given a finite NP/T net $N = (S, D, A, T)$, we say that a transition t is *statically enabled* by a set of places $S' \subseteq S$, denoted by $S'[\![t\rangle$, if $dom({}^{\bullet}t) \subseteq S'$. Given two sets $S_1, S_2 \subseteq S$, we say that S_2 is *statically reachable in one step* from S_1 if there exists a transition $t \in T$ such that $S_1[\![t\rangle$, $dom(t^{\bullet}) \cup {}^{\circ}t \not\subseteq S_1$ and $S_2 = S_1 \cup dom(t^{\bullet}) \cup {}^{\circ}t$; this is denoted by $S_1 \overset{t}{\Longrightarrow} S_2$. The *static reachability relation* $\Longrightarrow^* \subseteq \mathscr{P}_{fin}(S) \times \mathscr{P}_{fin}(S)$ is the least relation such that

- $S_1 \Longrightarrow^* S_1$ and
- if $S_1 \Longrightarrow^* S_2$ and $S_2 \overset{t}{\Longrightarrow} S_3$, then $S_1 \Longrightarrow^* S_3$.

A set of places $S_k \subseteq S$ is the *largest* set statically reachable from S_1 if $S_1 \Longrightarrow^* S_k$ and for all $t \in T$ such that $S_k[\![t\rangle$, we have that $dom(t^{\bullet}) \cup {}^{\circ}t \subseteq S_k$.

Given a finite NP/T net system $N(m_0) = (S, D, A, T, m_0)$, we denote by $[\![dom(m_0)\rangle$ the largest set of places statically reachable from $dom(m_0)$, i.e., the largest S_k such that $dom(m_0) \Longrightarrow^* S_k$. The *statically reachable subnet* $Net_s(N(m_0))$ is the net (S', D', A', T', m_0), where

$$\begin{aligned} S' &= [\![dom(m_0)\rangle, & D' &= S' \cap D,\\ T' &= \{t \in T \mid S'[\![t\rangle\}, & A' &= \{\ell \mid \exists t \in T' \text{ such that } l(t) = \ell\}. \end{aligned}$$

A finite NP/T system $N(m_0)$ is *statically reduced* if $Net_s(N(m_0)) = N(m_0)$. □

The algorithm of Table 3.1 can be adapted to compute the statically reachable subnet $Net_s(N(m_0))$ of any finite NP/T system $N(m_0) = (S, D, A, T, m_0)$. On the contrary, it is not always possible to compute the dynamically reachable subnet for a finite NP/T net, as this class of nets is Turing-complete (as we will see in the following) and so the reachability problem is undecidable, in general.

3.5.1 Behavioral Equivalences

Now we extend the behavioral equivalences defined for P/T nets in Section 3.4 to the richer case of NP/T nets.

Definition 3.25. (Net isomorphism) Two NP/T nets $N_1 = (S_1, D_1, A, T_1)$ and $N_2 = (S_2, D_2, A, T_2)$ are *isomorphic* if there exists a bijection $f : S_1 \to S_2$, homomorphically extended to markings, such that f restricted to D_1 is also a bijection from D_1 to D_2, and it preserves transitions, i.e., $(m, I, \ell, m') \in T_1$ if and only if $(f(m), f(I), \ell, f(m')) \in T_2$. This definition is extended to NP/T systems by requiring that the initial markings are related by f. □

Note that net isomorphism is decidable for finite NP/T nets, even though they are a Turing-complete model of computation.

Definition 3.26. (Interleaving marking graphs) The *abstract* interleaving marking graph of an NP/T system $N(m_0) = (S, D, A, T, m_0)$ is the rooted LTS

$aIMG(N(m_0)) = ([m_0\rangle, A, \longmapsto, m_0)$,

where m_0 is the initial state and the transition relation $\longmapsto \subseteq \mathscr{M}_{fin}(S) \times A \times \mathscr{M}_{fin}(S)$ is defined by $m \overset{a}{\longmapsto} m'$ if and only if there exists a transition $t \in T$ such that $m[t\rangle m'$ and $l(t) = a$.

The NP/T systems $N_1(m_1)$ and $N_2(m_2)$ are *abstract interleaving bisimilar* (denoted $N_1(m_1) \sim^a_{int} N_2(m_2)$) iff there exists a strong bisimulation $R \subseteq [m_1\rangle \times [m_2\rangle$ over the LTSs $aIMG(N_1(m_1))$ and $aIMG(N_2(m_2))$ such that $(m_1, m_2) \in R$.

The *concrete* interleaving marking graph of an NP/T net system $N(m_0) = (S, D, A, T, m_0)$ is the rooted LTS

$cIMG(N(m_0)) = ([m_0\rangle, Lab, \longrightarrow, m_0)$,

where m_0 is the initial state, the set of labels is $Lab = (A \cup D) \times \mathscr{P}_{fin}(D)$, and the transition relation $\longrightarrow \subseteq \mathscr{M}_{fin}(S) \times Lab \times \mathscr{M}_{fin}(S)$ is defined by $m \xrightarrow{\ell} m'$ if and only if either there exists a transition $t \in T$ such that $m[t\rangle m'$, $l(t) = a \in A$, ${}^\circ t = I$ and $\ell = (a, I)$; or there exists $s \in D$ such that $m(s) > 0$, $\ell = (s, \emptyset)$ and $m' = m$.

We say that two NP/T systems $N_1(m_1)$ and $N_2(m_2)$ are *concrete interleaving bisimilar* (denoted $N_1(m_1) \sim^c_{int} N_2(m_2)$) if and only if there exists a strong bisimulation $R \subseteq [m_1\rangle \times [m_2\rangle$ over the LTSs $cIMG(N_1(m_1))$ and $cIMG(N_2(m_2))$ such that $(m_1, m_2) \in R$. □

For instance, the (initial fragment of the) abstract interleaving marking graph associated with the NP/T net in Figure 3.18(b), with initial marking s_1, is outlined in Figure 3.19(a), while Figure 3.19(b) shows the (initial fragment of the) concrete

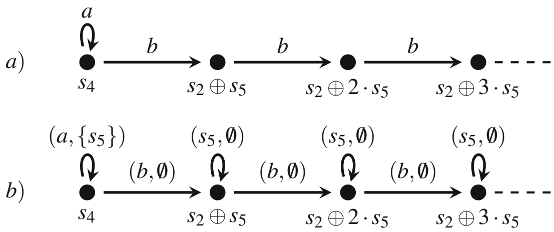


Fig. 3.19 Abstract and concrete interleaving marking graphs: an example

interleaving marking graph for the same net system. The concrete interleaving marking graph shows visibly the neg-set of the executed net transition and, additionally, introduces self-loop transitions labeled by the name of a currently active place that can be tested for absence. This definition deviates from the standard definition of interleaving marking graph for Petri nets with inhibitor arcs. We will see in Chapter 8 that this additional information is important for compositionality.

From the abstract interleaving marking graph $aIMG(N(m_0))$ of a finite NP/T net system $N(m_0)$, we can compute its associated Petri net language $L[N(m_0)]$ as the set of its weak completed traces (see Definition 3.18). Since finite NP/T nets are Turing-complete, the class of its net languages coincides with the class of recursively enumerable languages, i.e., the languages generated by general grammars in the Chomsky hierarchy [HMU01].

A possible step semantics for NP/T systems can be based on ideas developed for P/T nets with inhibitor arcs; among the various proposals (e.g., [JK95, Vog97, BP99, BP00]), we will follow the intuition of [BP99, BP00]. The distinguishing feature of this proposal is the following property: if two transitions can happen in the same step, then they can happen in either order. Formally, a step G is enabled at m iff

- $^\bullet G \subseteq m$, where $^\bullet G = \bigoplus_t G(t) \cdot {}^\bullet t$ (all the needed tokens are available);
- for all $t \in dom(G)$, for all $s \in {}^\circ t$, $m(s) = {}^\bullet t(s)$ (all the constraints on the absence of additional tokens are satisfied); additionally,
- for all t_1, t_2 such that $\{t_1, t_2\} \subseteq G$, we have that $dom(t_1^\bullet) \cap {}^\circ t_2 = \emptyset$ and $dom(t_2^\bullet) \cap {}^\circ t_1 = \emptyset$ (no transition produces tokens on places tested for absence of additional tokens by another transition); this can also be written as: for all G_1, G_2 such that $G = G_1 \oplus G_2$, $dom(G_1^\bullet) \cap {}^\circ G_2 = \emptyset$ and $dom(G_2^\bullet) \cap {}^\circ G_1 = \emptyset$, where $G^\bullet = \bigoplus_t G(t) \cdot t^\bullet$.

The third condition ensures that, for each pair of occurrences of transitions in the step, it never happens that one occurrence puts a token in a place inhibiting the other one. Note also that the second condition ensures that if $s \in {}^\circ t$ and $^\bullet t(s) > 0$, then $G(t) = 1$, or, equivalently, if $s \in {}^\circ t$ and $G(t) > 1$, then $^\bullet t(s) = 0$.

The execution of a step G enabled at m produces the marking $m' = (m \ominus {}^\bullet G) \oplus G^\bullet$, as for P/T nets. This is written as $m[G\rangle m'$.

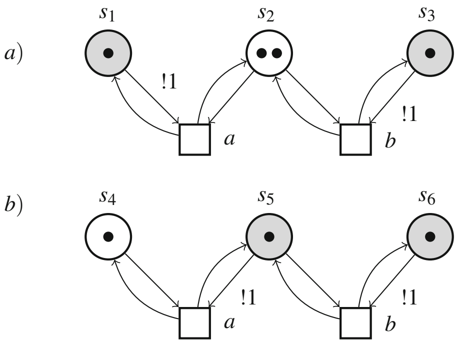


Fig. 3.20 Two further NP/T systems

Let us consider the NP/T system in Figure 3.20(a). The two transitions are $t_1 = (\{s_1, s_2\}, \{s_1\}, a, \{s_1, s_2\})$ and $t_2 = (\{s_2, s_3\}, \{s_3\}, b, \{s_2, s_3\})$. Step $G = \{t_1, t_2\}$ is enabled at the initial marking $m_0 = s_1 \oplus 2 \cdot s_2 \oplus s_3$ and it holds that $m_0[G\rangle m_0$. It is clear that the two transitions can be executed in either order: $m_0[t_1\rangle m_0[t_2\rangle m_0$ as well as $m_0[t_2\rangle m_0[t_1\rangle m_0$. However, it is not true that if two transitions can be executed in either order, then they can be performed in a step. Consider the NP/T system in Figure 3.20(b); its two transitions are $t_1 = (\{s_4, s_5\}, \{s_5\}, a, \{s_4, s_5\})$ and $t_2 = (\{s_5, s_6\}, \{s_6\}, b, \{s_5, s_6\})$ and the initial marking $m_1 = s_4 \oplus s_5 \oplus s_6$. Note that these two transitions can be performed in either order — $m_1[t_1\rangle m_1[t_2\rangle m_1$ and $m_1[t_2\rangle m_1[t_1\rangle m_1$ — however the step $G = \{t_1, t_2\}$ is not enabled at m_1 because ${}^\bullet t_1 \oplus {}^\bullet t_2 \not\subseteq m_1$.

Proposition 3.11. *Given an NP/T system $N(m_0) = (S, D, A, T, m_0)$, a marking m and a step $G_1 \oplus G_2$, where $G_i \neq \emptyset$ for $i = 1, 2$, if $m[G_1 \oplus G_2\rangle m'$, then there exist two markings m_1 and m_2 such that $m[G_1\rangle m_1[G_2\rangle m'$ and $m[G_2\rangle m_2[G_1\rangle m'$.*

Proof. If $G_1 \oplus G_2$ is enabled at m, then ${}^\bullet G_1 \oplus {}^\bullet G_2 \subseteq m$; for all $t \in dom(G_1 \oplus G_2)$, for all $s \in {}^\circ t$, $m(s) = {}^\bullet t(s)$; and, additionally, for all $\overline{G}_1, \overline{G}_2$ such that $G_1 \oplus G_2 = \overline{G}_1 \oplus \overline{G}_2$, $dom(\overline{G}_1^\bullet) \cap {}^\circ\overline{G}_2 = \emptyset$ and $dom(\overline{G}_2^\bullet) \cap {}^\circ\overline{G}_1 = \emptyset$; this last condition holds also when $\overline{G}_1 = G_1$ and $\overline{G}_2 = G_2$.

Therefore, also G_1 and G_2 are enabled at m. In fact, G_1 is enabled because ${}^\bullet G_1 \subseteq m$; for all $t \in dom(G_1)$, for all $s \in {}^\circ t$, $m(s) = {}^\bullet t(s)$; and, additionally, for all $\overline{G}_1, \overline{G}_2$ such that $G_1 = \overline{G}_1 \oplus \overline{G}_2$, $dom(\overline{G}_1^\bullet) \cap {}^\circ\overline{G}_2 = \emptyset$ and $dom(\overline{G}_2^\bullet) \cap {}^\circ\overline{G}_1 = \emptyset$.

The firing of G_1 at m produces the marking $m_1 = (m \ominus {}^\bullet G_1) \oplus G_1^\bullet$; marking m_1 enables G_2, because ${}^\bullet G_2 \subseteq m \ominus {}^\bullet G_1$ (given ${}^\bullet G_1 \oplus {}^\bullet G_2 \subseteq m$ and $m \ominus {}^\bullet G_1 \subseteq m_1$); for all $t \in dom(G_2)$, for all $s \in {}^\circ t$, $m_1(s) = {}^\bullet t(s)$ (given this condition was true for m and that $dom(G_1^\bullet) \cap {}^\circ G_2 = \emptyset$); and, additionally, also the third condition holds since it holds also for $G_1 \oplus G_2$. The firing of G_2 at m_1 produces the marking $m' = (m_1 \ominus {}^\bullet G_2) \oplus G_2^\bullet = m \ominus ({}^\bullet G_1 \oplus {}^\bullet G_2) \oplus (G_1^\bullet \oplus G_2^\bullet)$.

Similarly, G_2 is enabled at m, and its firing produces the marking $m_2 = (m \ominus {}^\bullet G_2) \oplus G_2^\bullet$; with an argument similar to the above, we can show that marking m_2

enables G_1, and the firing of G_1 at m_2 produces the marking $(m_2 \ominus {}^\bullet G_1) \oplus G_1^\bullet = m'$, as required. □

As a consequence, it is possible to prove that if a step G can fire, $m[G\rangle m'$, then for any linearization $t_1, t_2, \ldots, t_n$ of its transitions (i.e., such that for all $t \in T$, $G(t) = |\{i \mid 1 \leq i \leq n \wedge t_i = t\}|$), there exist markings $m_1, m_2, \ldots m_{n-1}$ such that $m[t_1\rangle m_1 [t_2\rangle m_2 \ldots m_{n-1} [t_n\rangle m'$. The obvious consequence is that, also for NP/T nets, the set of markings we can reach by the concurrent (or step) token game is exactly the same set we can compute by the sequential token game.

Definition 3.27. (Step marking graphs) The *abstract* step marking graph of $N(m_0) = (S, D, A, T, m_0)$ is the rooted STS

$aSMG(N(m_0)) = ([m_0\rangle, \mathscr{M}_{fin}(A), \longmapsto_s, m_0)$,

where m_0 is the initial state and the transition relation $\longmapsto_s \subseteq \mathscr{M}_{fin}(S) \times \mathscr{M}_{fin}(A) \times \mathscr{M}_{fin}(S)$ is defined by $m \overset{M}{\longmapsto}_s m'$ if and only if there exists a step G such that $m[G\rangle m'$ and $M = l(G)$, where the labeling function l is extended to multisets of transitions as $l(G)(a) = \sum_{t_i \in dom(G).l(t_i)=a} G(t_i)$.

The NP/T systems $N_1(m_1)$ and $N_2(m_2)$ are *abstract step bisimilar* (denoted $N_1(m_1) \sim^a_{step} N_2(m_2)$ or simply $m_1 \sim^a_{step} m_2$) if and only if there exists a strong bisimulation $R \subseteq [m_1\rangle \times [m_2\rangle$ over the STSs $aSMG(N_1(m_1))$ and $aSMG(N_2(m_2))$ such that $(m_1, m_2) \in R$.

The *concrete* step marking graph of an NP/T net system $N(m_0) = (S, D, A, T, m_0)$ is the rooted STS

$cSMG(N(m_0)) = ([m_0\rangle, \mathscr{M}_{fin}(Lab), \longrightarrow_s, m_0)$,

where m_0 is the initial state, $Lab = (A \cup D) \times \mathscr{P}_{fin}(D)$, and the transition relation $\longrightarrow_s \subseteq \mathscr{M}_{fin}(S) \times \mathscr{M}_{fin}(Lab) \times \mathscr{M}_{fin}(S)$ is defined by $m \overset{M}{\longrightarrow} m'$ if and only if either $m' = m$, $M \subseteq \mathscr{M}_{fin}(D \times \{\emptyset\})$ and $M((s, \emptyset)) \leq m(s)$; or there exists a step G such that $m[G\rangle m'$, $M = M_1 \oplus M_2$, $M_1 = {}^\circ l(G)$, where ${}^\circ l(t) = (l(t), {}^\circ t)$ and ${}^\circ l(G' \oplus t) = {}^\circ l(G') \oplus {}^\circ l(t)$, $M_2 \subseteq \mathscr{M}_{fin}(D \times \{\emptyset\})$, $m_2 = m \ominus {}^\bullet G$ and $M_2((s, \emptyset)) \leq m_2(s)$.

We say that two NP/T systems $N_1(m_1)$ and $N_2(m_2)$ are *concrete step bisimilar* (denoted $N_1(m_1) \sim^c_{step} N_2(m_2)$ or simply $m_1 \sim^c_{step} m_2$) iff there exists a strong bisimulation $R \subseteq [m_1\rangle \times [m_2\rangle$ over the STSs $cSMG(N_1(m_1))$ and $cSMG(N_2(m_2))$ such that $(m_1, m_2) \in R$. □

While the definition of the abstract step marking graph for NP/T nets is analogous to the definitions given in the literature for P/T nets with inhibitor arcs [JK95, Vog97, BP99], the definition of the concrete step marking graph is rather different, as it shows visibly the currently active testable places in D. This piece of information may be useful for compositionality.

Proposition 3.12. (Fully concurrent) *Given an NP/T system $N(m_0) = (S, D, A, T, m_0)$, its associated $aSMG(N(m_0))$ and $cSMG(N(m_0))$ are fully concurrent.*

Proof. The abstract/concrete step marking graphs are fully concurrent, as a direct consequence of Proposition 3.11. In particular, for $cSMG(N(m_0))$, note that, in any step label M, the presence of elements of the form $(s, \emptyset)$ can be always separated, as these can be freely performed whenever s is a currently marked testable place. □

3.5.2 Turing-Completeness

Finite NP/T nets are a Turing-complete formalism, as it is possible to represent any counter machine (CM, for short) as a finite NP/T net. We assume the reader is familiar with CMs [ER64, Min67], and so we give a short presentation of this computational model; a didactical presentation can be found in [GV15]. A CM M is a pair (I,n), where

$$I = \{(1 : I_1), \ldots, (m : I_m)\}$$

is the set of indexed instructions of M, with $|I| = m$, n is the number of registers r_j of M, and each instruction I_i is of two possible kinds:

- $I_i = Inc(r_j)$ $\quad 1 \leq j \leq n$ (increment register r_j and go to the next instruction of index $i+1$);
- $I_i = DecJump(r_j, s))$ $\quad 1 \leq j \leq n$ (if r_j holds value 0, then jump to the instruction of index s; otherwise, decrement register r_j and go to the next instruction of index $i+1$).

The finite NP/T net system $N(m_0) = (S,D,A,T,m_0)$ modeling the CM M is given as follows. The set of places $S = \{P_1, P_2, \ldots P_m, P_{last}, R_1, R_2, \ldots R_n\}$ is composed of a place P_i for each program-counter/instruction I_i, a place P_{last} to model the case in which the next instruction has index greater than m, and a place R_j for each register/counter r_j. The set D of testable places is $\{R_1, R_2, \ldots R_n\}$. The set A is $\{inc_j, dec_j, zero_j \;\big|\; 1 \leq j \leq n\}$. The set T of transitions models the operations of the CM M. In Figure 3.21(a) we show pictorially the transition for instruction $(i : Inc(r_j))$. The program counter flows from P_i to P_{i+1}, while one additional token is deposited into place R_j, to represent the increment of the register; if $i = m$, then P_{i+1} is actually P_{last}. In Figure 3.21(b), we show the net transitions for $(i : DecJump(r_j, s))$. If R_j contains at least one token, then the net transition labeled dec_j may be performed and instruction P_{i+1} is activated; if $i = m$, then P_{i+1} is actually P_{last}. Moreover, if R_j contains no tokens, the net transition labeled $zero_j$ can move the token from P_i to P_s; if $s > m$, then P_s is actually P_{last}. The initial marking m_0 puts one token into place P_1 and v_j tokens into place R_j (for $j = 1, \ldots n$), according to the initial values of the registers.

Summing up, we have showed that M can be faithfully represented by a finite NP/T net. As CMs constitute a Turing-complete formalism if they have at least two registers [Min67], the net modeling above ensures that finite NP/T nets with at least two unbounded places are a Turing-complete formalism.

However, differently from other Turing-complete models of concurrency such as CCS [Mil89], Multi-CCS [GV15] and the π-calculus [MPW92, SW01], finite NP/T nets can solve problems that are unsolvable in these process calculi. One such problem is the *Last Man Standing* (LMS) problem, introduced in [VBG09], which can be solved in a process calculus if there exists a process p able to detect the presence or absence of other copies of itself without generating deadlocks or introducing divergence. We would need to identify a process p such that p is able to execute an action a only when there is exactly one copy of p in the current system, while p is

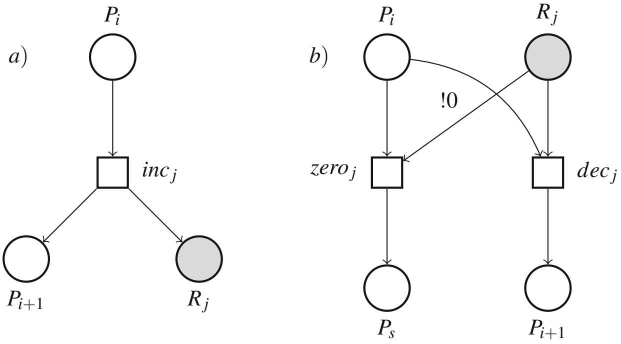


Fig. 3.21 Net representation of CM instructions

able to perform an action b only when there are at least two copies of p in the current system. To be precise, if q_i is the system where i copies of p are enabled, we require that all of its computations will end eventually (i.e., no divergence is allowed) and that the observable content of each of these computations is a if $i = 1$ or b if $i > 1$, where $a \neq b$. In other words,

$$\begin{array}{llll} & q_1 \;= p & q_1 \overset{a}{\Longrightarrow} q_1' \not\rightarrow & q_1 \overset{b}{\not\Longrightarrow} \\ \text{while} & & & \\ & q_2 \;= p\,|\,p & q_2 \overset{a}{\not\Longrightarrow} & q_2 \overset{b}{\Longrightarrow} q_2' \not\rightarrow \\ & \dots & & \\ & q_n \;= \underbrace{p\,|\,p\,|\cdots|\,p}_{n} & q_n \overset{a}{\not\Longrightarrow} & q_n \overset{b}{\Longrightarrow} q_n' \not\rightarrow \end{array}$$

where $-|-$ is the parallel composition operator of the calculus under scrutiny. The operational rules for parallel composition of CCS, as well as of the other related calculi, state that any process p, able to execute some action a, can perform the same action in the presence of other processes as well, so that if $p \overset{a}{\Longrightarrow} p'$, then also $p\,|\,p \overset{a}{\Longrightarrow} p\,|\,p'$, which contradicts the requirement that $q_2 \overset{a}{\not\Longrightarrow}$. As a matter of fact, CCS is *permissive*: no parallel process can prevent the execution of an action of another process.

However, the finite, dynamically acyclic, NP/T net in Figure 3.22 is a solution to the LMS problem! As a matter of fact, assuming that no token is in place s_2 at the beginning of the computation, if only one token is present in s_1 (i.e., there is only one copy of that process), then only a can be performed and the reached marking is empty; on the contrary, if two or more tokens are present in s_1 (i.e., we have multiple copies of the same process), then only b can be performed, at the end of a protocol involving the two τ-labeled transitions: first, two tokens from s_1 are removed and one token is put into place s_2, so that the a-labeled transition is inhibited from now on; then one token at a time from s_1 is consumed, together with the token in s_2, which is immediately regenerated; when place s_1 is empty, the token

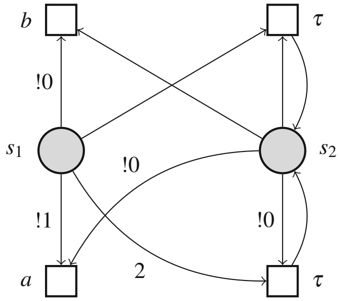


Fig. 3.22 Net solution to the LMS problem

in s_2 can perform b, reaching the empty marking. Note that this net is *dynamically acyclic*: at each step of the computation, the marking becomes smaller in size (one token less, at each step). Note also that finite, dynamically acyclic, NP/T nets are not Turing-complete; still they can solve the LMS problem.

Hence, this means there exists a problem in concurrency (i.e., the LMS problem) that a Turing-complete language (i.e., CCS) cannot solve, while it can be solved by a non-Turing-complete model (i.e., finite, dynamically acyclic, NP/T nets). This observation spurs an obvious question: when is a formalism for concurrency *complete*? And with respect to what? A possible answer is proposed in this book: a formalism is complete w.r.t. a given model of concurrency when it can model all the elements of that model, up to isomorphism. Following this intuition, the next chapters offer a list of increasingly expressive process calculi, each one *complete* w.r.t. a given class of nets, where the last calculus, called NPL (see Chapter 8), is able to represent all and only the finite NP/T nets. NPL is the only Turing-complete calculus we study in this book.

Chapter 4
The Basic Calculus: SFM

Abstract The sequential calculus SFM (acronym of *Sequential Finite-state Machines*) is equipped with an operational LTS semantics and an operational Petri net semantics; we observe that the two semantics are essentially isomorphic. We prove that all and only the sequential finite-state machines are the models of the Petri net semantics for SFM. Moreover, a denotational net semantics is presented and proved to be the same as the operational net semantics.

The basic language we start with is a simple *sequential* calculus, equipped with two operators only — *action prefixing* and *choice* — and a mechanism for describing recursive behavior, by means of so-called *process constants*. This calculus, called SFM (acronym of *Sequential Finite-state Machines*), is essentially finite-state CCS [Mil89, GV15].

4.1 Syntax

Let $\mathscr{L}$ be a finite set of *observable* actions, ranged over by $a, b, c, \ldots$. Let τ be an *invisible* action, not in $\mathscr{L}$. Let $Act = \mathscr{L} \cup \{\tau\}$, ranged over by μ. Let $\mathscr{C}ons$ be a countable set of process *constants* (i.e., names given to processes), disjoint from Act, ranged over by $A, B, C, \ldots$, possibly indexed.

SFM$_{\mathscr{L}}$ is the calculus whose process terms are generated from actions and constants as described by the following abstract syntax:

$$
\begin{aligned}
s &::= \mathbf{0} \mid \mu.p \mid s+s \\
p &::= s \mid \ C
\end{aligned}
$$

where $\mathbf{0}$ is the empty process, $\mu.p$ is a process where action $\mu \in Act$ (which can be an observable action a or the invisible action τ) prefixes the residual p ($\mu.-$ is called the *action prefixing* operator), $s_1 + s_2$ denotes the alternative composition of

R. Gorrieri, *Process Algebras for Petri Nets*, Monographs in Theoretical Computer Science. An EATCS Series, DOI 10.1007/978-3-319-55559-1_4

s_1 and s_2 ($- + -$ is called the *choice* operator), and C is a constant. We are actually defining a family of SFM process algebras, each one for the chosen, finite set $\mathscr{L}$ of observable actions; however, for simplicity, we simply use SFM do denote the language when the chosen $\mathscr{L}$ is not important, or clear from the context. Moreover, we can avoid this parametrization altogether, by choosing a comfortably large set $\mathscr{L}$, so that SFM is *the* process algebra we study in this chapter. (Similarly, for all the process algebras in the next chapters, we will omit the explicit parametrization.)

We assume that each constant C is equipped with a definition of the form $C \doteq p$. This allows for the definition of recursive behavior; for instance, with $C \doteq a.C$ we represent a simple recursive process that performs a forever, as will be clear in the next section. A constant $C \doteq p$ is defined by a process p in syntactic category s, thus ensuring that C is *guarded*, i.e., any constant in its body p occurs inside a subterm of p of the form $\mu.p'$. Moreover, we also assume that the set of constants $Const(p)$ used by p is finite. Let us formalize these concepts.

Definition 4.1. (*Const*(p): Set of constants used by p**)** By $Const(p)$ we denote the least set of process constants such that the following equations are satisfied:

$$Const(\mathbf{0}) = \emptyset, \qquad Const(p_1 + p_2) = Const(p_1) \cup Const(p_2),$$

$$Const(\mu.p) = Const(p), \qquad Const(C) = \begin{cases} \{C\} & \text{if } C \text{ undef.,} \\ \{C\} \cup Const(p) & \text{if } C \doteq p. \end{cases}$$

A term p such that $Const(p)$ is finite is called *finitary*, as it uses finitely many process constants only. □

When p is finitary, there is an obvious algorithm to compute $Const(p)$: it is enough to remember all the constants that have been already found while scanning p, in order to avoid applying again function $Const$ over their bodies. This can be achieved by the following auxiliary function δ, which has, as an additional parameter, a set I of already known constants:

$$\delta(\mathbf{0},I) = \emptyset, \qquad \delta(p_1 + p_2,I) = \delta(p_1,I) \cup \delta(p_2,I),$$

$$\delta(\mu.p,I) = \delta(p,I), \qquad \delta(C,I) = \begin{cases} \emptyset & C \in I, \\ \{C\} & C \notin I \wedge C \text{ undef.,} \\ \{C\} \cup \delta(p, I \cup \{C\}) & C \notin I \wedge C \doteq p. \end{cases}$$

Then, for any finitary term p, *Const(p)* $= \delta(p,\emptyset)$, where the additional parameter is the empty set because so is the set of process constants we know about when we begin scanning p. In this book, we always restrict ourselves to finitary calculi. So also in the next chapters, we will require that the set of constants used by any process is finite.

Remark 4.1. **(Finitary calculi vs finite calculi)** The six calculi we use in this book are finitary, meaning that any process term must use only finitely many constants. As argued in [Gor17], it could be preferable to assume that the set $\mathscr{C}ons$ of constants is finite. Hence, each calculus would be parametrized not only by a finite set $\mathscr{L}$ of actions, but also by a finite set $\mathscr{C}ons$ of constants, hence yielding a *finite* calculus,

$$\begin{array}{rclrcl}
sub(\mathbf{0},I) & = & \{\mathbf{0}\} & sub(p_1+p_2,I) & = & \{p_1+p_2\} \cup sub(p_1,I) \cup sub(p_2,I) \\
sub(\mu.p,I) & = & \{\mu.p\} \cup sub(p,I) & sub(C,I) & = & \begin{cases} \emptyset & C \in I, \\ \{C\} \cup sub(p, I \cup \{C\}) & C \notin I \wedge C \doteq p \end{cases}
\end{array}$$

Table 4.1 Subterms of a process

which is a finite formal system, indeed. However, in order to describe some simple examples exploiting infinitely many constants, we prefer to allow for a countable set $\mathscr{C}ons$ of constants, thus avoiding the explicit finite parametrization. □

Definition 4.2. (Defined constant, fully defined term and SFM process) A constant C is *defined* if it possesses a defining equation of the form $C \doteq p$, with p in syntactic category s. A term p is *fully defined* if all the constants in $Const(p)$ are defined. The set $\mathscr{P}_{SFM}$ of SFM *processes* contains the fully defined terms p such that $Const(p)$ is finite. $\mathscr{P}_{SFM}$ will be ranged over by p,q,r, possibly indexed. □

As the choice operator is associative and commutative w.r.t. most behavioral equivalences we have introduced in Chapter 2, we sometimes use the n-ary version of this operator: we shorten $p_1 + p_2 + \ldots + p_n$ as $\Sigma_{i=1}^{n} p_i$. As $\mathbf{0}$ is the neutral element for $+$ w.r.t. most behavioral equivalences, we often omit occurrences of $\mathbf{0}$ summands; e.g., $(\mathbf{0}+p)+\mathbf{0}$ is simply written as p. Using this alternative n-ary choice operator and the assumption of absorption of useless $\mathbf{0}$ summands, the syntax of SFM processes is more succinctly given as

$$p \ ::= \ \Sigma_{j \in J} \mu_j . p_j \ \mid \ C$$

where J is finite, $\Sigma_{j \in J} \mu_j . p_j = \mathbf{0}$ when $J = \emptyset$, and any constant C is guarded, i.e., its body is of the form $\Sigma_{j \in J} \mu_j . p_j$.

Definition 4.3. (Subterm) For any SFM process p, the set of its subterms $sub(p)$ is defined by means of the auxiliary function (with the same name, with abuse of notation) $sub(p,\emptyset)$, whose second parameter is a set of already known constants, initially empty, described in Table 4.1. □

Theorem 4.1. *For any SFM process p, the set of its subterms $sub(p)$ is finite.*

Proof. *By induction on the definition of $sub(p,\emptyset)$. The base cases are $sub(\mathbf{0},I)$ and $sub(C,I)$ when $C \in I$. Note that induction will end eventually because an SFM process uses finitely many constants.* □

Proposition 4.1. *For any SFM processes p and q, the following hold:*

(i) *$p \in sub(p)$, and*
(ii) *if $p \in sub(q)$, then $sub(p) \subseteq sub(q)$.*

Proof. *The proof of (i) follows directly from the definition of $sub(-)$ in Table 4.1. The proof of (ii) follows by the observation that the definition of $sub(q)$ recursively calls itself on all of its subterms.* □

$$(\text{Pref})\ \frac{}{\mu.p \xrightarrow{\mu} p} \qquad\qquad (\text{Cons})\ \frac{p \xrightarrow{\mu} p'}{C \xrightarrow{\mu} p'}\ \ C \doteq p$$

$$(\text{Sum}_1)\ \frac{p \xrightarrow{\mu} p'}{p+q \xrightarrow{\mu} p'} \qquad\qquad (\text{Sum}_2)\ \frac{q \xrightarrow{\mu} q'}{p+q \xrightarrow{\mu} q'}$$

Table 4.2 Structural operational LTS semantics

4.2 Operational LTS Semantics

The next step is to define the labeled transition system for SFM. This is, of course, an infinite LTS, for the number of states and the number of transitions. The states are actually the SFM processes. A finite, implicit representation of this infinite, countable set is given by means of the abstract syntax outlined in the previous section. More difficult is finding a finite, implicit representation of the infinite, countable set of transitions. For this, we resort to Plotkin's technique called *Structural Operational Semantics* (SOS for short) [Plo04a, Plo04b], according to which the transitions are defined by means of an inference system composed of axioms and rules.

A typical SOS operational rule has the form

$$\frac{\mathit{premises}}{\mathit{conclusion}}\ \ \mathit{side\ condition}$$

where *premises* is the conjunction of zero (and in such a case the rule is called an *axiom*) or more transitions, *conclusion* is one transition and *side condition* is a predicate that must be true for rule applicability.

Definition 4.4. The SFM labeled transition system $\mathscr{C}_{SFM}$ is the triple $(\mathscr{P}_{SFM}, Act, \rightarrow)$ where $\rightarrow \subseteq \mathscr{P}_{SFM} \times Act \times \mathscr{P}_{SFM}$ is the least transition relation generated by the axiom and rules in Table 4.2. □

A brief comment on the rules follows. Axiom (Pref) states that $\mu.p$ can perform its prefix μ and then may continue with the residual p. Rule (Cons) states that, when $C \doteq p$, if p can perform μ reaching p', then C can also perform μ, reaching p' as well. Rule (Sum$_1$) states that any move executable by p is also a possible move for $p+q$; symmetrically for rule (Sum$_2$).

Proposition 4.2. *For any $p \in \mathscr{P}_{SFM}$, if $p \xrightarrow{\mu} p'$, then $p' \in \mathscr{P}_{SFM}$.*

Proof. By induction on the proof of $p \xrightarrow{\mu} p'$. □

Hence, by iterating this result, we have the expected result that any term reachable from a process is also a process.

Remark 4.2. **(Notation)** For any $p \in \mathscr{P}_{SFM}$, the LTS reachable from p (Definition 2.5) is $\mathscr{C}_p = (\mathscr{P}_p, sort(p), \rightarrow_p, p)$, where $\mathscr{P}_p$ is the set of processes reachable from

p, $sort(p)$ is the set of actions that can be performed by p (formally, $sort(p) = \{\mu \in Act \mid \exists p'.p \longrightarrow^* p' \xrightarrow{\mu} \}$) and $\rightarrow_p$ is the restriction of the transition relation to $\mathscr{P}_p \times sort(p) \times \mathscr{P}_p$. □

The following obvious fact follows.

Proposition 4.3. *For any $p \in \mathscr{P}_{SFM}$, the LTS $\mathscr{C}_p = (\mathscr{P}_p, sort(p), \rightarrow_p, p)$ reachable from p is a rooted, reduced LTS.* □

4.2.1 Expressiveness

We prove that any SFM process originates, via the SOS semantics, a finite-state LTS, i.e., an LTS with finitely many states and labels; then, we provide a *representability theorem*, stating that any finite-state LTS can be represented by an SFM process, up to isomorphism.

Proposition 4.4. *For any $q \in \mathscr{P}_{SFM}$, the set $T_q = \{(q, \mu, q') \mid \mu \in Act, q' \in \mathscr{P}_{SFM}$ and $q \xrightarrow{\mu} q'\}$ of its outgoing transitions is finite.*

Proof. *We can define the number $\gamma(q)$ of transitions leaving a given state/process q. Function $\gamma : \mathscr{P}_{SFM} \rightarrow \mathbb{N}$ is defined as follows:*

$$\gamma(\mathbf{0}) = 0, \qquad \gamma(\mu.p) = 1,$$
$$\gamma(p_1 + p_2) = \gamma(p_1) + \gamma(p_2), \qquad \gamma(C) = \gamma(p) \;\; \mathit{if}\, C \doteq p.$$

By guardedness, we are sure that $\gamma(C)$ will not call itself recursively, and so it is guaranteed that $\gamma(C)$ is always a finite number. It is not difficult then to check — by reasoning on the shape of the SOS inference rules — that indeed $\gamma(q)$ is the number of transitions leaving q, i.e., $\gamma(q) = |T_q|$. □

Corollary 4.1. (Finitely branching LTS) *For any $p \in \mathscr{P}_{SFM}$, the rooted LTS $\mathscr{C}_p = (\mathscr{P}_p, sort(p), \rightarrow_p, p)$ reachable from p is finitely branching.*

Proof. *All the states reachable from $p \in \mathscr{P}_{SFM}$ are SFM processes by Proposition 4.2. Hence $\mathscr{C}_p$ is a finitely branching LTS by Proposition 4.4.* □

Lemma 4.1. *For any $p \in \mathscr{P}_{SFM}$, if $p \xrightarrow{\mu} p'$, then*

(i) $p' \in sub(p)$, and
(ii) $sub(p') \subseteq sub(p)$.

Proof. *(i) By induction on the proof of $p \xrightarrow{\mu} p'$. (ii) By (i) and Proposition 4.1.* □

Proposition 4.5. *For any $p \in \mathscr{P}_{SFM}$, if $p \longrightarrow^* p'$, then*

(i) $p' \in sub(p)$, and
(ii) $sub(p') \subseteq sub(p)$.

Proof. *By induction on the length of the computation $p \longrightarrow^* p'$. The base case is $p \longrightarrow^* p$ and the thesis follows by Proposition 4.1. Otherwise, let $p \longrightarrow^* q \xrightarrow{\mu} p'$. By induction we have that $q \in sub(p)$ and $sub(q) \subseteq sub(p)$. By Lemma 4.1, we also have $p' \in sub(q)$ and $sub(p') \subseteq sub(q)$. Hence, the thesis follows trivially.* □

Theorem 4.2. (Finite number of reachable states) *For any SFM process p, the set $\mathscr{P}_p$ of its reachable states is finite.*

Proof. *By Proposition 4.5, any reachable state p' is a subterm of p. By Theorem 4.1, the set $sub(p)$ is finite. Hence, the set $\mathscr{P}_p$ of its reachable states is finite.* □

Corollary 4.2. (Finite-state LTS) *For any SFM process p, the LTS reachable from p, $\mathscr{C}_p = (\mathscr{P}_p, sort(p), \rightarrow_p, p)$, is finite-state.*

Proof. *By Theorem 4.2, the set $\mathscr{P}_p$ of the states reachable from p is finite. By Corollary 4.1, $\mathscr{C}_p$ is finitely branching. Hence, also sort(p) must be finite, as required by Definition 2.7.* □

Theorem 4.3. (Representability, up to LTS isomorphism) *For any finite-state, rooted, reduced LTS TS, there exists an SFM process p such that the LTS $\mathscr{C}_p = (\mathscr{P}_p, sort(p), \rightarrow_p, p)$ reachable from p is isomorphic to TS.*

Proof. *Let $TS = (Q, A, \rightarrow_1, q_0)$, with $Q = \{q_0, q_1, \dots, q_n\}$. We define a process constant C_i in correspondence with state q_i, for $i = 0, 1, \dots, n$, defined as follows: if q_i is a deadlock, then $C_i \doteq \mathbf{0}$; if $T(q_i) = \{(q_i, \mu_k, q_k) \mid \mu_k \in A, q_k \in Q \text{ and } q_i \xrightarrow{\mu_k}_1 q_k\}$, then $C_i \doteq \Sigma_{(q_i, \mu_k, q_k) \in T(q_i)} \mu_k.C_k$. Let us consider $\mathscr{C}_{C_0} = (\mathscr{P}_{C_0}, sort(C_0), \rightarrow_2, C_0)$. It is not difficult to see that $\mathscr{P}_{C_0} = \{C_0, C_1, \dots, C_n\}$ because TS is reduced. Hence, the bijection we are looking for is $f : Q \rightarrow \mathscr{P}_{C_0}$, defined as $f(q_i) = C_i$. It is also easy to observe that the two conditions of isomorphism are satisfied, namely*

- $C_0 = f(q_0)$, *and*
- $q_i \xrightarrow{\mu_k}_1 q_k$ *iff* $f(q_i) \xrightarrow{\mu_k}_2 f(q_k)$.

Hence, f is indeed an LTS isomorphism. □

Example 4.1. As a simple application of this theorem, the finite-state LTS in Figure 2.1 can be represented by the four mutually recursive constants

$$C_1 \doteq coin.C_2, \qquad C_2 \doteq ask\text{-}esp.C_3 + ask\text{-}am.C_4,$$
$$C_3 \doteq esp\text{-}coffee.C_1, \qquad C_4 \doteq am\text{-}coffee.C_1.$$ □

Remark 4.3. **(SFM languages = regular languages)** A set $L \subseteq \mathscr{L}^*$ is an SFM language if there exists an SFM process p such that $L = WCTr(p)$. By Corollary 4.2, for any SFM process p, the LTS $\mathscr{C}_p$, reachable from p, is finite-state. As discussed in Remark 2.2, for any finite-state LTS TS_1, the set $WCTr(TS_1)$ of its weak completed traces is a regular language. Hence, *all* the SFM languages are regular. Conversely, given a regular language L, Remark 2.2 shows that there exists a finite-state LTS TS_2 such that $L = WCTr(TS_2)$. By Theorem 4.3, there exists an SFM process q such that $\mathscr{C}_q$ is isomorphic to TS_2; hence, $L = WCTr(q)$ and so, *all* regular languages are SFM languages. Summing up, we can conclude that SFM can represent exactly this class of formal languages. □

4.2.2 Congruence

When we compare the behavior of different systems, we expect to use a behavioral relation that is not only an equivalence, but rather a *congruence*. An equivalence relation R is a congruence if it is preserved by the operators of the process calculus. In our setting, we expect to be able to prove that if $(p,q) \in R$, then $(\mu.p, \mu.q) \in R$ for any $\mu \in Act$ as well as $(p+r, q+r) \in R$ for any process r. The reason why we require that our chosen behavioral equivalence be a congruence is that it will support *substitution of equals for equals in any context*: if a subcomponent p in a complex system $\mathscr{C}[p]$ becomes faulty, we can safely replace it with a congruent process q, so that the whole system $\mathscr{C}[q]$ behaves as $\mathscr{C}[p]$ (where a *context* $\mathscr{C}[-]$ is a process expression with a single occurrence of a hole $[-]$ in it, as a subexpression).

Not all the equivalences we have discussed in Chapter 2 are congruences. For instance, it is not difficult to see that LTS isomorphism $\cong$ (see Definition 2.8) is not a congruence for $+$. As an instance, consider $a.\mathbf{0}$ and $a.(\mathbf{0}+\mathbf{0})$, which generate isomorphic LTSs; however, when put in a context $\mathscr{C}[-] = - + a.\mathbf{0}$, we get the two processes $\mathscr{C}[a.\mathbf{0}] = a.\mathbf{0} + a.\mathbf{0}$ and $\mathscr{C}[a.(\mathbf{0}+\mathbf{0})] = a.(\mathbf{0}+\mathbf{0}) + a.\mathbf{0}$ which generate non-isomorphic LTSs: the former is composed of two states only, and the latter of three. Nonetheless, the most fundamental behavioral equivalences for our aims are actually congruences.

Theorem 4.4. (Bisimulation equivalence is a congruence)

i) For any $p, q \in \mathscr{P}_{SFM}$, if $p \sim q$, then $\mu.p \sim \mu.q$ for all $\mu \in Act$;
ii) For any processes p, q in syntactic category s, if $p \sim q$, then $p + r \sim q + r$ for all r in syntactic category s.

Proof. *Assume R is a bisimulation such that $(p,q) \in R$.*

For case 1), consider relation $R_1 = \{(\mu.p, \mu.q) \mid \mu \in Act\} \cup R$. It is very easy to check that R_1 is a bisimulation.

For case 2), we show that $R_2 = \{(p+r, q+r) \mid r$ in syntactic category $s\} \cup R \cup \mathscr{I}$ is a bisimulation, where $\mathscr{I} = \{(r,r) \mid r \in \mathscr{P}_{SFM}\}$. If $p + r \xrightarrow{\mu} p'$, this must be due to either $p \xrightarrow{\mu} p'$ (due to rule (Sum$_1$)) or $r \xrightarrow{\mu} p'$ (due to rule (Sum$_2$)). In the former case, as $(p,q) \in R$, there exists a transition $q \xrightarrow{\mu} q'$ with $(p', q') \in R$; by rule (Sum$_1$) also transition $q + r \xrightarrow{\mu} q'$ is derivable with $(p', q') \in R_2$. In the latter case, $q + r$ can do exactly the same transition from r: $q + r \xrightarrow{\mu} p'$ with $(p', p') \in R_2$. The symmetric case when $q + r$ moves first is analogous, hence omitted. Note that the symmetric case $r + p \sim r + q$ is implied by the fact that the choice operator is commutative w.r.t. $\sim$. □

It can be proved that also trace equivalence is a congruence for the operators of SFM. A discussion about weak congruences is outside the scope of this book (see, e.g., [GV15]).

$$
\begin{array}{ll}
dec(\mathbf{0}) = \emptyset & dec(\mu.p) = \{\mu.p\} \\[2ex]
dec(p+p') = \{p+p'\} & dec(C) = \{C\}
\end{array}
$$

Table 4.3 Decomposition function

$$
\begin{array}{llll}
\text{(pref)} & \dfrac{}{\{\mu.p\} \xrightarrow{\mu} dec(p)} & \text{(cons)} & \dfrac{dec(p) \xrightarrow{\mu} m}{\{C\} \xrightarrow{\mu} m} \; C \doteq p \\[3ex]
(\text{sum}_1) & \dfrac{dec(p) \xrightarrow{\mu} m}{\{p+q\} \xrightarrow{\mu} m} & (\text{sum}_2) & \dfrac{dec(q) \xrightarrow{\mu} m}{\{p+q\} \xrightarrow{\mu} m}
\end{array}
$$

Table 4.4 Structural operational net semantics

4.3 Operational Net Semantics

For SFM, the operational net semantics mimics closely the operational LTS semantics, so that the two semantics originate structures which are essentially isomorphic.

Similarly to what we did when defining the LTS SOS semantics, we first describe a technique for building a P/T net for the whole calculus SFM, starting from a description of its places and its net transitions. The resulting net $N_{SFM} = (S_{SFM}, Act, T_{SFM})$ is such that, for any $p \in \mathscr{P}_{SFM}$, the net system $N_{SFM}(dec(p))$ *statically* reachable from the initial marking $dec(p)$ is a *statically* reduced,[1] sequential FSM net; such a net system is denoted by $Net(p)$.

The infinite set S_{SFM} of places is exactly $\mathscr{P}_{SFM} \setminus \{\mathbf{0}\}$, i.e., the set of all SFM processes except $\mathbf{0}$. With abuse of notation, the set S_{SFM} is ranged over by s, possibly indexed.

Function $dec : \mathscr{P}_{SFM} \to \mathscr{M}_{fin}(S_{SFM})$ defines the decomposition of processes into markings (see Table 4.3). It is essentially an identity function, associating with each process p the singleton multiset $\{p\}$, the only exception being $\mathbf{0}$, with which the empty multiset is associated.

Let $T_{SFM} \subseteq (\mathscr{M}_{fin}(S_{SFM}) \setminus \{\emptyset\}) \times Act \times \mathscr{M}_{fin}(S_{SFM})$ be the least set of transitions generated by the axiom and rules in Table 4.4, where in a transition $m_1 \xrightarrow{\mu} m_2$, m_1 is the pre-set, μ is the label and m_2 is the post-set. Axiom (pref) states that if one token is present in the place $\mu.p$ then a μ-labeled transition is derivable from marking $\{\mu.p\}$, producing the marking $dec(p)$. Rule (sum_1) and its counterpart (sum_2) are as expected: the transitions from the place $p+q$ are those from the places p and q; in fact, $dec(p)$ is either $\emptyset$, when $p = \mathbf{0}$, or the singleton $\{p\}$; but $\emptyset$ cannot gener-

[1] Since, for FSM nets, the notion of statically reachable subnet and dynamically reachable subnet coincide, we could also write that $N_{SFM}(dec(p))$ is the subnet *dynamically* reachable from the initial marking $dec(p)$; similarly, since the notion of statically reduced net and dynamically reduced net coincide, we could also write *dynamically* reduced.

ate any transition, so the only possibility to derive the conclusion $\{p+q\} \xrightarrow{\mu} m$ is when the premise transition is of the form $dec(p) = \{p\} \xrightarrow{\mu} m$. Finally, rule (cons) explains that if a net transition from $dec(p)$ is derivable, reaching m, then also the place C can do the same, reaching m as well; similarly to the discussion above about rule (sum$_1$), the premise transition is derivable only if $dec(p) = \{p\}$, i.e., when $p \neq \mathbf{0}$.

We now prove that any net transition is such that its pre-set is a singleton and its post-set is a singleton or the empty set.

Proposition 4.6. *For any $t \in T_{SFM}$, we have that $|^\bullet t| = 1$ and $|t^\bullet| \leq 1$.*

Proof. *By induction on the proof of t. The base case is when t is derived by axiom (pref). In such a case, $^\bullet t = \{\mu.p\}$ and $t^\bullet = dec(p)$, which can be either $\{p\}$ or $\emptyset$, so that $|t^\bullet| \leq 1$ as required. An easy induction on the other three rules is enough to conclude that $|t^\bullet| \leq |^\bullet t| = 1$. For instance, for rule (sum$_1$), we know that p is sequential, hence $dec(p)$ is either the empty multiset or $\{p\}$; the former case is vacuous, because no premise transition is derivable from $\emptyset$ and so no conclusion is possible; in the latter case, we can assume by induction that the target marking m of the premise is such that $|m| \leq 1$ and so the thesis follows trivially.* □

Hence, the net transitions are all of the form $\{p\} \xrightarrow{\mu} dec(p')$. Now we want to prove that the LTS transitions are essentially the same as the net transitions.

Proposition 4.7. *For any $p \in \mathscr{P}_{SFM}$, $p \xrightarrow{\mu} p'$ if and only if $\{p\} \xrightarrow{\mu} dec(p')$.*

Proof. *The LTS rules in Table 4.2 and net rules in Table 4.4 are clearly in a one-to-one correspondence. Note that, as $dec(p) = \{p\}$ (when $p \neq \mathbf{0}$), the conclusion of axiom (pref) is actually $\{\mu.p\} \xrightarrow{\mu} \{p\}$, which mimics exactly axiom (Pref) of Table 4.2; in case $p = \mathbf{0}$, the conclusion of rule (pref) is actually $\{\mu.\mathbf{0}\} \xrightarrow{\mu} \emptyset$, which parallels $\mu.\mathbf{0} \xrightarrow{\mu} \mathbf{0}$. A trivial induction on the three rules suffices to prove the result. Let us consider only one case: rules (sum$_1$) and (Sum$_1$). On the one hand, $\{p+q\} \xrightarrow{\mu} dec(p')$ is possible only if the premise of rule (sum$_1$) is of the form $\{p\} \xrightarrow{\mu} dec(p')$; by induction, also $p \xrightarrow{\mu} p'$ is derivable and, by rule (Sum$_1$), also transition $p+q \xrightarrow{\mu} p'$ is derivable, as required. Symmetrically, if we start from $p+q \xrightarrow{\mu} p'$, by (Sum$_1$), its premise is $p \xrightarrow{\mu} p'$; by induction, we have that $\{p\} \xrightarrow{\mu} dec(p')$, and so, by (sum$_1$), $\{p+q\} \xrightarrow{\mu} dec(p')$, as required.* □

Remark 4.4. **(Isomorphic structures)** It is clear that the LTS $\mathscr{C}_{SFM} = (\mathscr{P}_{SFM}, Act, \rightarrow)$ of Definition 4.4 and the P/T net $N_{SFM} = (S_{SFM}, Act, T_{SFM})$ are essentially isomorphic structures: function $dec : \mathscr{P}_{SFM} \rightarrow \mathscr{M}_{fin}(S_{SFM})$ defines a bijection between $\mathscr{P}_{SFM}$ and $\{\{p\} \mid p \in S_{SFM}\} \cup \{\emptyset\}$, and such a bijection, by Proposition 4.7, preserves the transitions: $p \xrightarrow{\mu} p'$ iff $dec(p) \xrightarrow{\mu} dec(p')$. Hence, the net semantic is sound w.r.t. the LTS semantics: the process p in $\mathscr{C}_{SFM}$ and marking $dec(p)$ in N_{SFM} are interleaving bisimilar, as $R = \{(p, dec(p)) \mid p \in \mathscr{P}_{SFM}\}$ is trivially a strong bisimulation. □

Given a process $p \in \mathscr{P}_{SFM}$, the P/T system associated with p is the subnet of N_{SFM} statically reachable from the initial marking $dec(p)$. We indicate by $Net(p)$ such a subnet.

Definition 4.5. Let p be an SFM process. The P/T system statically associated with p is $Net(p) = (S_p, A_p, T_p, \{p\})$, where

$$\begin{aligned} S_p &= [\![\{p\}\rangle \quad \text{computed in } N_{SFM}, \\ T_p &= \{t \in T_{SFM} \mid S_p[\![t\rangle\}, \\ A_p &= \{\mu \in Act \mid \exists t \in T_p, \mu = l(t))\}. \end{aligned}$$ □

The following proposition presents a fact that is obviously true by construction of the net $Net(p)$ associated with an SFM process p.

Proposition 4.8. *For any $p \in \mathscr{P}_{SFM}$, $Net(p)$ is a statically reduced P/T net.* □

It is an obvious observation that all the nonempty reachable markings from the initial marking $\{p\}$ are singletons. Hence, $Net(p)$ is a safe net.

Proposition 4.9. *Given an SFM process p, if $m \in [dec(p)\rangle$, then either $m = \emptyset$ or $m = \{q\}$ for some $q \in sub(p)$.*

Proof. *If $p = \mathbf{0}$, then $dec(p) = \emptyset$ and no other marking is reachable; hence, the thesis trivially holds. If $p \neq \mathbf{0}$ (i.e., $dec(p) = \{p\}$), the proof proceeds by induction on the definition of reachable marking (Definition 3.5). The base case is $\{p\} \in [\{p\}\rangle$ and the thesis holds, as $p \in sub(p)$ by Proposition 4.1. Now, assume that $m_1 \in [\{p\}\rangle$ and that, for some transition $t \in T_{SFM}$, $m_1[t\rangle m_2$; we want to prove that m_2 is either the empty multiset or a singleton containing a subterm of p. By induction, we can assume that $m_1 = \{r\}$ for some $r \in sub(p)$; hence, $sub(r) \subseteq sub(p)$ by Proposition 4.1. Since t is enabled at $m_1 = \{r\}$, necessarily ${}^\bullet t = \{r\}$ as the pre-set of a transition cannot be empty; by Proposition 4.6, either $t^\bullet = \emptyset$ or $t^\bullet = \{q\}$ for some q; in the former case, the reached marking is empty and so the thesis holds; in the latter case, $m_2 = m_1 \ominus {}^\bullet t \oplus t^\bullet = \{q\}$ is a singleton as well. Moreover, by Proposition 4.7, $t = (\{r\}, \mu, \{q\})$ is such that $r \xrightarrow{\mu} q$ is a transition in the LTS. By Lemma 4.1, $q \in sub(r)$; therefore, $q \in sub(p)$, as required.* □

As the notions of dynamically enabled transition and statically enabled transition coincide for FSM nets, the proposition above ensures that the dynamically reachable places are the same as the statically reachable places.

Theorem 4.5. (Finite number of places) *For any SFM process p, the set S_p of its places statically reachable from $dom(dec(p))$ is finite.*

Proof. *By Proposition 4.9 any reachable place q is a subterm of p. By Theorem 4.1, the set $sub(p)$ is finite. Hence, the set S_p of its reachable places is finite.* □

As a corollary to the above, we have that for any SFM process p, the set $[dec(p)\rangle$ of its reachable markings is finite.

Corollary 4.3. (Finite number of reachable markings) *For any SFM process p, the set $[dec(p)\rangle$ of its (dynamically) reachable markings is finite.*

Proof. *By Theorem 4.5, the set S_p of places is finite. By Proposition 4.9, any reachable marking is a singleton or the empty multiset. Therefore, the reachable markings cannot be more than $|S_p|+1$.* □

Lemma 4.2. (Finite number of transitions) *For any SFM process p, the set T^p of transitions with pre-set $\{p\}$ is finite.*

Proof. *A direct consequence of Proposition 4.7 and Proposition 4.4.* □

Theorem 4.6. (Finite P/T Petri net) *For any SFM process p, the P/T net reachable from p, $Net(p) = (S_p, A_p, T_p, \{p\})$, is finite.*

Proof. *By Theorem 4.5, the set S_p of the places reachable from p is finite. By Proposition 4.6 the pre-set of any derivable transition is a singleton. By Lemma 4.2, the set T^q of transitions with pre-set $\{q\}$ is finite, for any $q \in S_p$. Hence, also $T_p = \bigcup_{q \in S_p} T^q$ must be finite, being a finite union of finite sets.* □

Corollary 4.4. (Sequential finite-state machine) *For any SFM process $p \neq \mathbf{0}$, $Net(p)$ is a sequential finite-state machine.*

Proof. *By Theorem 4.6, $Net(p)$ is a finite P/T net. By Proposition 4.6, for any $t \in T_p$, we have $|t^\bullet| \leq |{}^\bullet t| = 1$, i.e., $Net(p)$ is a finite-state machine. Finally, it is sequential as $dec(p) = \{p\}$.* □

For $p = \mathbf{0}$, $Net(p)$ is the empty net $(\emptyset, \emptyset, \emptyset, \emptyset)$, which is not a sequential finite-state machine, because the initial marking is not a singleton. However, except for this singularity, the corollary above ensures that *only* sequential finite-state machines can be represented by SFM processes.

4.4 Representing All Sequential Finite-State Machines

It is not a surprise that *all* sequential finite-state machines can be represented by SFM processes. As a matter of fact, this can be justified by Theorem 4.3, stating that all finite-state LTSs can be represented by SFM processes, and by Remark 4.4, stating that the LTS for SFM and the net for SFM are essentially isomorphic structures.

Here, we give a construction that, albeit a bit redundant, will be useful in proving the result and will be naturally extended to the more complex cases studied in the following chapters. The translation from nets to process terms we are going to present defines a constant C_i corresponding to each place s_i, which has a summand c_i^j for each transition t_j of the net. Such a summand c_i^j defines whether and how constant C_i is involved in the execution of the transition of the net of C_i, corresponding to the original transition t_j. It turns out that, if the original net $N(\{s_1\})$ is (statically) reduced, then the term $\mathscr{T}_{SFM}(N(\{s_1\}))$ associated with such a net generates a net isomorphic to the original net $N(\{s_1\})$.

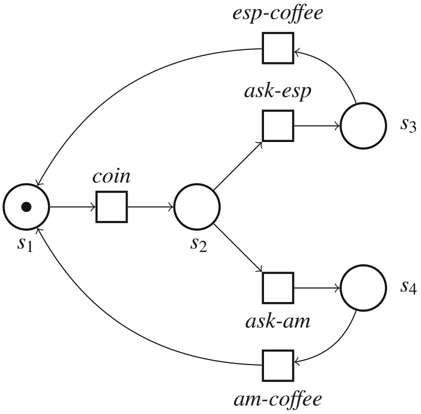


Fig. 4.1 A vending machine as a sequential finite-state machine

Definition 4.6. (Translating sequential FSMs into SFM process terms) Let $N(m_0) = (S,A,T,m_0)$ — with $S = \{s_1,\dots,s_n\}$, $A \subseteq Act$, $T = \{t_1,\dots,t_k\}$, $l(t_j) = \mu_j$ and $m_0 = \{s_1\}$ — be a sequential finite-state machine. Function $\mathscr{T}_{SFM}(-)$, from sequential finite-state machines to SFM processes, is defined as $\mathscr{T}_{SFM}(N(\{s_1\})) = C_1$, where each C_i, for $i = 1,\dots,n$, has a defining equation of the form

$$C_i \doteq c_i^1 + \cdots + c_i^k$$

where $C_i \doteq \mathbf{0}$ if $k = 0$, and each summand c_i^j, for $j = 1,\dots,k$, is equal to

- $\mathbf{0}$, if $s_i \not\in {}^\bullet t_j$;
- $\mu_j.\mathbf{0}$, if ${}^\bullet t_j = \{s_i\}$ and $t_j^\bullet = \emptyset$;
- $\mu_j.C_h$, if ${}^\bullet t_j = \{s_i\}$ and $t_j^\bullet = \{s_h\}$. □

Note that $\mathscr{T}_{SFM}(N(m_0))$ is an SFM process: in fact, the only operators we use are action prefixing and choice, tthere are finitely many constants involved (one for each place) and each constant is guarded. Therefore, the following proposition holds by Proposition 4.8 and Corollary 4.4.

Proposition 4.10. *Given a sequential finite-state machine $N(m_0) = (S,A,T,m_0)$, the P/T net $Net(\mathscr{T}_{SFM}(N(m_0)))$ is a (statically) reduced, sequential FSM.* □

Example 4.2. Let us consider the vending machine described by the net in Figure 4.1. The SFM process associated with such an FSM net is C_1, where the four constants C_i, corresponding to the four places s_i, for $i = 1,\dots,4$, are defined as follows:

$$\begin{aligned}
C_1 &\doteq coin.C_2 + \mathbf{0} + \mathbf{0} + \mathbf{0} + \mathbf{0},\\
C_2 &\doteq \mathbf{0} + ask\text{-}esp.C_3 + ask\text{-}am.C_4 + \mathbf{0} + \mathbf{0},\\
C_3 &\doteq \mathbf{0} + \mathbf{0} + \mathbf{0} + esp\text{-}coffee.C_1 + \mathbf{0},\\
C_4 &\doteq \mathbf{0} + \mathbf{0} + \mathbf{0} + \mathbf{0} + am\text{-}coffee.C_1.
\end{aligned}$$

It is obvious that $Net(C_1)$ is a net isomorphic to the original net in Figure 4.1, where the isomorphism f is the bijection $f(s_i) = C_i$ for $i = 1,\dots,4$. These constant definitions are very similar to those of Example 4.1, where (apparently useless) $\mathbf{0}$ summands are inserted to denote that a constant is not involved in the execution of the transition under scrutiny. □

Now we are ready to state our main result, the so-called *representability theorem*.

Theorem 4.7. (Representability theorem 1) *Let $N(m_0) = (S,A,T,m_0)$ be a (statically) reduced, sequential finite-state machine such that $A \subseteq Act$, and let $p = \mathscr{T}_{SFM}(N(m_0))$. Then, $Net(p)$ is isomorphic to $N(m_0)$.*

Proof. *Let $N(m_0) = (S,A,T,m_0)$ be a reduced FSM net, with $S = \{s_1,\dots,s_n\}$, $m_0 = \{s_1\}$, $A \subseteq Act$, $T = \{t_1,\dots,t_k\}$ and $l(t_j) = \mu_j$ for $j = 1,\dots,k$. The associated SFM process is $\mathscr{T}_{SFM}(N(m_0)) = C_1$, where for each place s_i we have a corresponding constant $C_i \doteq \sum_{j=1}^{k} c_i^j$, defined as in Definition 4.6. For notational convenience, $\sum_{j=1}^{k} c_i^j$ is denoted by p_i, i.e., $C_i \doteq p_i$.*

Let $Net(p)$ be the tuple (S',A',T',m'_0). Then, $m'_0 = dec(p)$ is the multiset $dec(C_1)$ $= \{C_1\}$. Note that, by Definition 4.6, any transition $t' \in T'$ with ${}^\bullet t' = \{C_1\}$ is such that $t'^\bullet = \emptyset$ or $t'^\bullet = \{C_h\}$ for some suitable h; by iterating this observation, each transition in T' has a pre-set $\{C_i\}$ and a post-set which can be empty or of the form $\{C_h\}$, for some suitable C_i and C_h. As, by Proposition 4.10, $Net(p)$ is reduced, it follows that the places in S' are exactly all the constants C_i for $i = 1,\dots,n$. Note also that since $N(m_0)$ is reduced, all the places in S are reachable from the initial marking $\{s_1\}$. Hence, there is a bijection $f : S \to S'$ defined by $f(s_i) = C_i$, which is the natural candidate isomorphism function. To prove that f is an isomorphism, we have to prove that

1. *$f(m_0) = m'_0$,*
2. *$t = (m,\mu,m') \in T$ implies $f(t) = (f(m),\mu,f(m')) \in T'$, and*
3. *$t' = (m'_1,\mu,m'_2) \in T'$ implies there exists $t = (m_1,\mu,m_2) \in T$ such that $f(t) = t'$, i.e., $f(m_1) = m'_1$ and $f(m_2) = m'_2$.*

From items 2) and 3) above, it follows that $A = A'$.

Proof of 1*: The mapping via f of the initial marking m_0 is $f(m_0) = f(\{s_1\}) = \{C_1\} = dec(C_1) = m'_0$, as required.*

Proof of 2*: we prove that, for $j = 1,\dots,k$, if $t_j = (m,\mu_j,m') \in T$, then $t'_j = (f(m),\mu_j,f(m')) \in T'$. By Proposition 4.6, t_j must be of the form $(\{s_i\},\mu_j,\{s_h\})$ (or $(\{s_i\},\mu_j,\emptyset)$) for suitable indexes i,h. Hence, $f({}^\bullet t_j) = \{C_i\}$ and $f(t_j^\bullet) = \{C_h\}$ (or $f(t_j^\bullet) = \emptyset$). According to Definition 4.6, for $C_i = p_i$, we have in p_i a summand $c_i^j = \mu_j.C_h$ (or $c_i^j = \mu_j.\mathbf{0}$). Therefore, not only is $\{c_i^j\} \xrightarrow{\mu_j} \{C_h\}$ (or $\{c_i^j\} \xrightarrow{\mu_j} \emptyset$) derivable by axiom (pref), but also (by rules (sum$_1$) and (sum $_2$)) $\{p_i\} \xrightarrow{\mu_j} \{C_h\}$ (or $\{p_i\} \xrightarrow{\mu_j} \emptyset$) and so, by rule (cons), $\{C_i\} \xrightarrow{\mu_j} \{C_h\}$ (or $\{C_i\} \xrightarrow{\mu_j} \emptyset$). In conclusion, starting from transition $t_j = (\{s_i\},\mu_j,\{s_h\})$ (or $t_j = (\{s_i\},\mu_j,\emptyset)$), we have shown that transition $t'_j = (\{C_i\},\mu_j,\{C_h\})$ (or $t'_j = (\{C_i\},\mu_j,\emptyset)$) belongs to T', as required.*

Proof of 3*: We prove that if $t'_j = (m'_1, \mu_j, m'_2) \in T'$, then there exists a transition $t_j = (m_1, \mu_j, m_2) \in T$ such that $f(m_1) = m'_1$ and $f(m_2) = m'_2$. We already observed that any transition t'_j (statically) reachable from the initial marking $\{C_1\}$ is such that ${}^\bullet t'_j = \{C_i\}$ and $t'^\bullet_j = \{C_h\}$ (or $t'^\bullet_j = \emptyset$) for suitable indexes i, h; hence, $t'_j = (\{C_i\}, \mu_j, \{C_h\})$ (or $t'_j = (\{C_i\}, \mu_j, \emptyset)$). Such a transition is derivable, by rule (cons), only if also $\{p_i\} \xrightarrow{\mu_j} \{C_h\}$ (or $\{p_i\} \xrightarrow{\mu_j} \emptyset$) and so, by rules (sum$_1$) and (sum$_2$), only if $\{c^j_i\} \xrightarrow{\mu_j} \{C_h\}$ (or $\{c^j_i\} \xrightarrow{\mu_j} \emptyset$). Summand c^j_i must be of the form $\mu_j.C_h$ (or $\mu_j.\mathbf{0}$), and this is possible only if transition t_j is of the form $(\{s_i\}, \mu_j, \{s_h\})$ (or $(\{s_i\}, \mu_j, \emptyset)$), as required.* □

Example 4.3. Function $\mathscr{T}_{SFM}(-)$ can be applied to any sequential FSM net $N(m_0)$. However, if $N(m_0)$ is not (statically) *reduced*, the representability theorem does not hold. Let us consider the net

$N(\{s_1\}) = (\{s_1, s_2, s_3, s_4\}, \{a\}, \{(\{s_1\}, a, \{s_2\}), (\{s_3\}, a, \{s_4\})\}, \{s_1\})$.

Clearly such a net is not reduced because places s_3 and s_4 are not reachable from the initial marking $\{s_1\}$. The SFM term $\mathscr{T}_{SFM}(N(\{s_1\}))$ would be C_1, where

$C_1 \doteq a.C_2 + \mathbf{0}, \qquad C_2 \doteq \mathbf{0} + \mathbf{0},$
$C_3 \doteq \mathbf{0} + a.C_4, \qquad C_4 \doteq \mathbf{0} + \mathbf{0},$

but now $Net(\mathscr{T}_{SFM}(N(\{s_1\})))$ is the net $(\{C_1, C_2\}, \{a\}, \{(\{C_1\}, a, \{C_2\})\}, \{C_1\})$, which has two places and one transition only, i.e., it is isomorphic to the subnet of $N(\{s_1\})$ reachable from the initial marking $\{s_1\}$. □

4.5 Denotational Net Semantics

The operational net semantics for SFM is very simple, but has one obvious drawback: the construction of the net system $Net(p)$ associated with an SFM process p is rather indirect (see Definition 4.5), as it has to be computed inductively starting from the initial marking $dec(p)$, by using the operational rules in Table 4.4.

In this section we provide a completely different construction of the net system $[\![p]\!]_\emptyset$ associated with process p, which is compositional and denotational in style. The details of the construction are outlined in Table 4.5. The encoding is parametrized by a set of constants that has already been found while scanning p; such a set is initially empty and it is used to avoid looping on recursive constants. The definition is syntax driven and also the places of the constructed net are syntactic objects, i.e., SFM process terms. For instance, the net system $[\![a.\mathbf{0}]\!]_\emptyset$ is a net composed of one single marked place, namely process $a.\mathbf{0}$, and one single transition $(\{a.\mathbf{0}\}, a, \emptyset)$. A bit of care is needed in the rule for choice: in order to include only strictly necessary places and transitions, the initial place p_1 (or p_2) of the subnet $[\![p_1]\!]_I$ (or $[\![p_2]\!]_I$) is to be kept in the net for $p_1 + p_2$ only if there exists a transition reaching place p_1 (or p_2) in $[\![p_1]\!]_I$ (or $[\![p_2]\!]_I$), otherwise p_1 (or p_2) can be safely removed in the new net. Similarly, for the rule for constants. We will prove that the net system $Net(p)$ associated with p by the operational semantics is exactly the same net system defined by $[\![p]\!]_\emptyset$.

$$
\begin{array}{rcll}
[\![\mathbf{0}]\!]_I & = & (\emptyset,\emptyset,\emptyset,\emptyset) & \\[1ex]
[\![\mu.p]\!]_I & = & (S,A,T,\{\mu.p\}) & \text{given } [\![p]\!]_I = (S',A',T',dec(p)) \text{ and where} \\
 & & & S=\{\mu.p\}\cup S',\ A=\{\mu\}\cup A',\ T=\{(\{\mu.p\},\mu,dec(p))\}\cup T' \\[1ex]
[\![p_1+p_2]\!]_I & = & (S,A,T,\{p_1+p_2\}) & \text{given } [\![p_i]\!]_I = (S_i,A_i,T_i,dec(p_i)) \text{ for } i=1,2, \text{ and where} \\
 & & & S=\{p_1+p_2\}\cup S_1'\cup S_2', \text{ with, for } i=1,2, \\
 & & & S_i' = \begin{cases} S_i & \exists t\in T_i \text{ such that } t^\bullet(p_i)>0 \ \vee\ p_i=\mathbf{0}, \\ S_i\setminus\{p_i\} & \text{otherwise} \end{cases} \\
 & & & A=A_1\cup A_2,\ T=T'\cup T_1'\cup T_2', \text{ with, for } i=1,2, \\
 & & & T_i' = \begin{cases} T_i & \exists t\in T_i\,.t^\bullet(p_i)>0 \ \vee\ p_i=\mathbf{0}, \\ T_i\setminus\{t\in T_i \mid {}^\bullet t(p_i)>0\} & \text{otherwise} \end{cases} \\
 & & & T'=\{(\{p_1+p_2\},\mu,m) \mid (\{p_i\},\mu,m)\in T_i, i=1,2\} \\[1ex]
[\![C]\!]_I & = & (\{C\},\emptyset,\emptyset,\{C\}) & \text{if } C\in I \\[1ex]
[\![C]\!]_I & = & (S,A,T,\{C\}) & \text{if } C\notin I, \text{ given } C\doteq p \text{ and } [\![p]\!]_{I\cup\{C\}} = (S',A',T',dec(p)) \\
 & & & A=A', S=\{C\}\cup S'', \text{ where} \\
 & & & S'' = \begin{cases} S' & \exists t\in T'\,.t^\bullet(p)>0 \ \vee\ p=\mathbf{0}, \\ S'\setminus\{p\} & \text{otherwise} \end{cases} \\
 & & & T=\{(\{C\},\mu,m) \mid (\{p\},\mu,m)\in T'\}\cup T'' \text{ where} \\
 & & & T'' = \begin{cases} T' & \exists t\in T'\,.t^\bullet(p)>0 \ \vee\ p=\mathbf{0}, \\ T'\setminus\{t\in T' \mid {}^\bullet t(p)>0\} & \text{otherwise} \end{cases}
\end{array}
$$

Table 4.5 Denotational net semantics

Example 4.4. Consider the two processes $b.\mathbf{0}$ and $b.A$, where $A\doteq\mathbf{0}$. By using the definitions in Table 4.5, we have that $[\![\mathbf{0}]\!]_\emptyset = (\emptyset,\emptyset,\emptyset,\emptyset)$. By the rule for action prefixing, $[\![b.\mathbf{0}]\!]_\emptyset = (\{b.\mathbf{0}\},\{b\},\{(\{b.\mathbf{0}\},b,\emptyset)\},\{b.\mathbf{0}\})$. Similarly, $[\![\mathbf{0}]\!]_{\{A\}} = (\emptyset,\emptyset,\emptyset,\emptyset)$. Then, by the rule for constants, $[\![A]\!]_\emptyset = (\{A\},\emptyset,\emptyset,\{A\})$; finally, by the rule for action prefixing, $[\![b.A]\!]_\emptyset = (\{b.A,A\},\{b\},\{(\{b.A\},b,\{A\})\},\{b.A\})$. □

Example 4.5. Consider the process $b.A+c.A$, where $A\doteq\mathbf{0}$. By Example 4.4, we have $[\![b.A]\!]_\emptyset = (\{b.A,A\},\{b\},\{(\{b.A\},b,\{A\})\},\{b.A\})$. Similarly,
$[\![c.A]\!]_\emptyset = (\{c.A,A\},\{c\},\{(\{c.A\},c,\{A\})\},\{c.A\})$.
Hence, $[\![b.A+c.A]\!]_\emptyset =$
$(\{b.A+c.A,A\},\{b,c\},\{(\{b.A+c.A\},b,\{A\}),(\{b.A+c.A\},c,\{A\})\},\{b.A+c.A\})$.
Note that places $b.A$ and $c.A$ have not been included in the resulting net. Now consider process $b.\mathbf{0}+a.b.\mathbf{0}$. We have that
$[\![b.\mathbf{0}]\!]_\emptyset = (\{b.\mathbf{0}\},\{b\},\{(\{b.\mathbf{0}\},b,\emptyset)\},\{b.\mathbf{0}\})$ and
$[\![a.b.\mathbf{0}]\!]_\emptyset = (\{a.b.\mathbf{0},b.\mathbf{0}\},\{a,b\},\{(\{a.b.\mathbf{0}\},a,\{b.\mathbf{0}\}),(\{b.\mathbf{0}\},b,\emptyset)\},\{a.b.\mathbf{0}\})$.
Therefore, by summation,

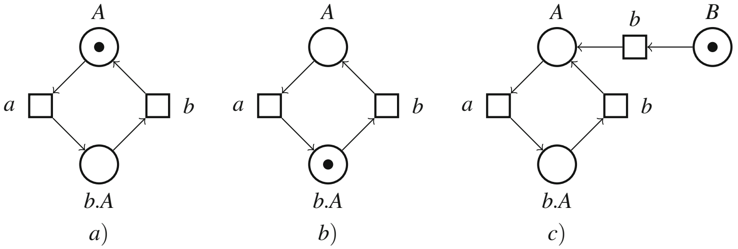


Fig. 4.2 The nets for the processes A in (a), for $b.A$ in (b) and for B in (c) of Example 4.6

$[\![b.\mathbf{0}+a.b.\mathbf{0}]\!]_\emptyset = (\{b.\mathbf{0}+a.b.\mathbf{0}, b.\mathbf{0}\}, \{a,b\}, \{(\{b.\mathbf{0}+a.b.\mathbf{0}\}, b, \emptyset), (\{b.\mathbf{0}+a.b.\mathbf{0}\}, a, \{b.\mathbf{0}\}), (\{b.\mathbf{0}\}, b, \emptyset)\}, \{b.\mathbf{0}+a.b.\mathbf{0}\})$.

Note that place $a.b.\mathbf{0}$ has not been included in the resulting net, while $b.\mathbf{0}$ is in, because it is reachable from $a.b.\mathbf{0}$.

Finally, consider the process $b.\mathbf{0}+\mathbf{0}$. By Example 4.4, we have that $[\![b.\mathbf{0}]\!]_\emptyset = (\{b.\mathbf{0}\}, \{b\}, \{(\{b.\mathbf{0}\}, b, \emptyset)\}, \{b.\mathbf{0}\})$ and $[\![\mathbf{0}]\!]_\emptyset = (\emptyset,\emptyset,\emptyset,\emptyset)$. Therefore, by summation, $[\![b.\mathbf{0}+\mathbf{0}]\!]_\emptyset = (\{b.\mathbf{0}+\mathbf{0}\}, \{b\}, \{(\{b.\mathbf{0}+\mathbf{0}\}, b, \emptyset)\}, \{b.\mathbf{0}+\mathbf{0}\})$. □

Example 4.6. Consider constant $B \doteq b.A$, where $A \doteq a.b.A$. By using the definitions in Table 4.5, $[\![A]\!]_{\{A,B\}} = (\{A\}, \emptyset, \emptyset, \{A\})$. Then, by action prefixing, $[\![b.A]\!]_{\{A,B\}} = (\{b.A, A\}, \{b\}, \{(\{b.A\}, b, \{A\})\}, \{b.A\})$. Again, by action prefixing, $[\![a.b.A]\!]_{\{A,B\}} = (\{a.b.A, b.A, A\}, \{a,b\}, \{(\{a.b.A\}, a, \{b.A\}), (\{b.A\}, b, \{A\})\}, \{a.b.A\})$. Now, the rule for constants ensures that

$[\![A]\!]_{\{B\}} = (\{b.A, A\}, \{a,b\}, \{(\{A\}, a, \{b.A\}), (\{b.A\}, b, \{A\})\}, \{A\})$.

Note that place $a.b.A$ has been removed, as no transition in $[\![a.b.A]\!]_{\{A,B\}}$ reaches that place. This net is depicted in Figure 4.2(a). By action prefixing,

$[\![b.A]\!]_{\{B\}} = (\{b.A, A\}, \{a,b\}, \{(\{A\}, a, \{b.A\}), (\{b.A\}, b, \{A\})\}, \{b.A\})$,

i.e., this operation changes only the initial marking, but does not affect the underlying net! This net in outlined in Figure 4.2(b). Finally,

$[\![B]\!]_\emptyset = (\{B, b.A, A\}, \{a,b\}, \{(\{B\}, b, \{A\}), (\{A\}, a, \{b.A\}), (\{b.A\}, b, \{A\})\}, \{B\})$.

Note that place $b.A$ has been kept, because there is a transition in the net $[\![b.A]\!]_{\{B\}}$ that reaches that place. Figure 4.2(c) shows this net. □

Theorem 4.8. (Operational and denotational semantics coincide) *For any SFM process p, $Net(p) = [\![p]\!]_\emptyset$.*

Proof. By induction on the definition of $[\![p]\!]_I$ and of $Net(p,I)$, where the latter is the operational net associated with p, assuming that the constants in I are undefined. Then, the thesis follows for $I = \emptyset$.

The first base case is for $\mathbf{0}$ and the thesis is obvious, as, for any $I \subseteq \mathscr{C}ons$, $[\![\mathbf{0}]\!]_I = (\emptyset,\emptyset,\emptyset,\emptyset) = Net(\mathbf{0}, I)$, because no place/token is available and so no transition is derivable from $\emptyset$ by the operational rules in Table 4.4. The second base case is for C when $C \in I$, which corresponds to the case when C is not defined. In such a case, for any $I \subseteq \mathscr{C}ons$ with $C \in I$, $[\![C]\!]_I = (\{C\}, \emptyset, \emptyset, \{C\}) = Net(C, I)$, because, *being undefined, no transition is derivable from C (rule (cons) is not applicable). The inductive cases are as follows.*

Action prefixing*: By induction, we assume that* $Net(p,I) = (S',A',T',dec(p)) = [\![p]\!]_I$. *As* $dec(\mu.p) = \{\mu.p\}$, *the only applicable operational rule is (pref), which produces exactly one transition,* $(\{\mu.p\},\mu,dec(p))$, *which reaches the initial marking of* $Net(p,I)$. *Hence,* $Net(\mu.p,I)$ *is the net* $(S' \cup \{\mu.p\}, A' \cup \{\mu\}, T' \cup \{(\{\mu.p\}, \mu, dec(p))\}, \{\mu.p\})$, *which is exactly* $[\![\mu.p]\!]_I$.

Choice*: By induction, we can assume that* $Net(p_i,I) = (S_i,A_i,T_i,dec(p_i)) = [\![p_i]\!]_I$ *for* $i = 1,2$. *As* $dec(p_1+p_2) = \{p_1+p_2\}$, *the only applicable rules are (*sum_1*) and (*sum_2*). Any transition* $(\{p_1\},\mu,dec(p)) \in T_1$ *generates, by rule (*sum_1*), a transition* $(\{p_1+p_2\},\mu,dec(p))$ *in the net for* p_1+p_2. *Symmetrically, for the net transitions from* $\{p_2\}$. *This means that all the places, such as p, reachable in one step from* p_1 *or* p_2, *are also reachable in one step from* p_1+p_2. *Therefore, the set S of places of* $Net(p_1+p_2,I)$ *is* $\{p_1+p_2\} \cup S_1' \cup S_2'$, *where, for* $i = 1,2$,

$$S_i' = \begin{cases} S_i & \exists t \in T_i \text{ such that } t^\bullet(p_i) > 0, \\ S_i \setminus \{p_i\} & \text{otherwise} \end{cases}$$

because p_i *is to be included only if it is reachable from some state different from* p_i *itself. Set S is the same set computed by* $[\![p_1+p_2]\!]_I$. *Similarly, the set T of transitions of* $Net(p_1+p_2,I)$ *is* $T' \cup T_1' \cup T_2'$, *where*

$$T_i' = \begin{cases} T_i & \exists t \in T_i \text{ such that } t^\bullet(p_i) > 0, \\ T_i \setminus \{t \in T_i \mid {}^\bullet t(p_i) > 0\} & \text{otherwise} \end{cases}$$

because we have to remove all the transitions starting from p_i *if such a place has been removed; moreover,*

$$T' = \{(\{p_1+p_2\},\mu,m) \mid (\{p_i\},\mu,m) \in T_i, i = 1,2\}$$

computes the set of the new initial transitions, as obtained by application of the rules (sum_1*) and (*sum_2*). T is exactly the same set of transitions of* $[\![p_1+p_2]\!]_I$.

Constant*: In this case, we assume that* $C \doteq p$ *and that* $C \notin I$. *By induction, we can assume that* $Net(p,I\cup\{C\}) = (S',A',T',dec(p)) = [\![p]\!]_{I\cup\{C\}}$, *so that in p constant C is undefined. As* $dec(C) = \{C\}$, *the only applicable operational rule is (cons), so that any transition* $(\{p\},\mu,dec(q))$ *in* $Net(p,I\cup\{C\})$ *produces a transition* $(\{C\},\mu,dec(q))$ *in* $Net(C,I)$. *This means that all the places, such as q, reachable in one step from p, are also reachable in one step from C. Therefore, the set of places S of* $Net(C,I)$ *is* $\{C\} \cup S''$, *where*

$$S'' = \begin{cases} S' & \exists t \in T' \text{ such that } t^\bullet(p) > 0, \\ S' \setminus \{p\} & \text{otherwise} \end{cases}$$

because p is to be included only if it is reachable from some state different from p itself. Set S is the same set computed by $[\![C]\!]_I$. *Similarly, the set T of transitions of* $Net(C,I)$ *is* $\{(\{C\},\mu,m) \mid (\{p\},\mu,m) \in T'\} \cup T''$, *where*

$$T'' = \begin{cases} T' & \exists t \in T' \text{ such that } t^\bullet(p) > 0, \\ T' \setminus \{t \in T' \mid {}^\bullet t(p) > 0\} & \text{otherwise} \end{cases}$$

because we have to remove all the transitions starting from p if such a place has been removed; moreover,

$\{(\{C\},\mu,m) \mid (\{p\},\mu,m) \in T'\}$

computes the set of new initial transitions, as obtained by application of rule (cons). Note that this has the effect of defining the constant C, by giving a semantics to it in terms of the initial transitions of p. Note also that T is exactly the same set of transitions of $[\![C]\!]_I$. □

Of course, this denotational semantics can be easily adapted to define an LTS denotational semantics for SFM.

Chapter 5
Adding Asynchronous Parallel Composition: CFM and BPP

Abstract SFM is extended with the operator of asynchronous (i.e., without synchronization capabilities) parallel composition in two steps: first, by adding parallel composition at the top level only, thus obtaining CFM (acronym of *Concurrent Finite-state Machines*); then, by additionally extending the definition of action prefixing to parallel processes, thus yielding BPP (acronym of *Basic Parallel Processes*).

5.1 CFM

Also in this chapter, let $Act = \mathscr{L} \cup \{\tau\}$ be the finite set of actions, ranged over by μ, and let $\mathscr{C}ons$ be the countable set of constants, disjoint from Act, ranged over by $A, B, C, \ldots$. The CFM *terms* (where CFM is the acronym of *Concurrent Finite-state Machines*) are generated from actions and constants by the following abstract syntax

$$\begin{array}{lll} s ::= \mathbf{0} \mid \mu.q \mid s+s & & \\ q ::= s \mid \ C & & \textit{sequential processes} \\ p ::= q \mid p \,|\, p & & \textit{parallel processes} \end{array}$$

where $p_1 \,|\, p_2$ denotes the asynchronous parallel composition of p_1 and p_2; as for SFM, C is defined over the syntactic category s, i.e., $C \doteq s$. A term p is a CFM *process* if it is fully defined and the set $Const(p)$ of constants used by p is finite, where function $Const(-)$ of Definition 4.1 is extended to parallel composition as

$Const(p_1 \,|\, p_2) = Const(p_1) \cup Const(p_2)$.

The set of CFM processes is denoted by $\mathscr{P}_{CFM}$, and the set of its sequential processes, i.e., of the processes in syntactic category q, by $\mathscr{P}_{CFM}^{seq}$. Note that $\mathscr{P}_{CFM}^{seq} = \mathscr{P}_{SFM}$. Alternatively, the syntax can be more succinctly given as

$$\begin{array}{l} s ::= \Sigma_{j \in J} \mu_j.s_j \mid C \\ p ::= s \mid p \,|\, p \end{array}$$

R. Gorrieri, *Process Algebras for Petri Nets*, Monographs in Theoretical
Computer Science. An EATCS Series, DOI 10.1007/978-3-319-55559-1_5

$$(\mathrm{Par}_1)\ \frac{p \xrightarrow{\mu} p'}{p \,|\, q \xrightarrow{\mu} p' \,|\, q} \qquad (\mathrm{Par}_2)\ \frac{q \xrightarrow{\mu} q'}{p \,|\, q \xrightarrow{\mu} p \,|\, q'}$$

Table 5.1 Structural operational semantics for asynchronous parallel composition

where J is finite, $\Sigma_{j \in J} \mu_j.s_j = \mathbf{0}$ when $J = \emptyset$, and any constant C is guarded, i.e., its body is of the form $\Sigma_{j \in J} \mu_j.s_j$.

The definition of the subterms $sub(p)$ of a process p (see Definition 4.3) is to be extended to the parallel operator as follows:

$sub(p_1 \,|\, p_2, I) = sub(p_1, I) \cup sub(p_2, I)$.

This means that $sub(p)$ computes the set of *sequential* subterms of p, i.e., for any $p \in \mathscr{P}_{CFM}$, the set $sub(p)$ of its sequential subterms is a subset of $\mathscr{P}_{CFM}^{seq}$.

Theorem 5.1. *For any CFM process p, the set $sub(p)$ of its sequential subterms is finite.*

Proof. By induction on the definition of $sub(p, \emptyset)$. □

Proposition 5.1. *For any CFM processes p and q, the following hold:*

(i) *if p is sequential, then $p \in sub(p)$, and*
(ii) *if $p \in sub(q)$, then $sub(p) \subseteq sub(q)$.*

Proof. The proof of (i) follows directly from the definition of $sub(-)$ in Table 4.1. The proof of (ii) follows by the observation that the definition of $sub(q)$ recursively calls itself on all of its sequential subterms. □

5.1.1 Interleaving LTS Semantics

The CFM labeled transition system $\mathscr{C}_{CFM}$ is the triple $(\mathscr{P}_{CFM}, Act, \to)$ where the transition relation $\to \subseteq \mathscr{P}_{CFM} \times Act \times \mathscr{P}_{CFM}$ is the least relation generated by the axiom and rules in Tables 4.2 and 5.1.

(Par$_1$) and (Par$_2$) are the rules describing the *asynchronous* execution of an action by one of the two subcomponents of a parallel process. Specifically, (Par$_1$) states that if $p \xrightarrow{\mu} p'$, then $p \,|\, q \xrightarrow{\mu} p' \,|\, q$. Note that q is not discarded by the transition, and for this reason this operator is called *static*. (Par$_2$) is symmetric. These two rules together state that $p \,|\, q$ can do whatever p and q can do, possibly *interleaving* their executions.

Example 5.1. Consider the following SFM specification of a bounded counter that can count up to 2:

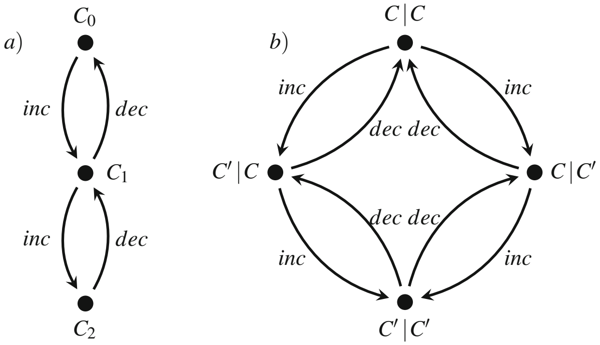


Fig. 5.1 LTSs for two 2-bounded counters

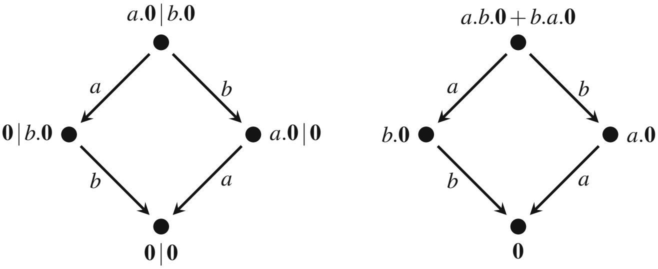

$a.\mathbf{0}\,|\,b.\mathbf{0}$
a
b
$\mathbf{0}\,|\,b.\mathbf{0}$
$a.\mathbf{0}\,|\,\mathbf{0}$
b
a
$\mathbf{0}\,|\,\mathbf{0}$
$a.b.\mathbf{0}+b.a.\mathbf{0}$
a
b
$b.\mathbf{0}$
$a.\mathbf{0}$
b
a
$\mathbf{0}$

Fig. 5.2 Interleaving law: two isomorphic LTSs

$$C_0 \doteq inc.C_1,$$
$$C_1 \doteq inc.C_2 + dec.C_0,$$
$$C_2 \doteq dec.C_1.$$

Consider then the SFM 1-bounded counter:

$$C \doteq inc.C', \qquad C' \doteq dec.C$$

and the CFM process $C\,|\,C$. Figure 5.1 shows the LTSs for C_0 and $C\,|\,C$. These two processes are clearly strongly bisimilar because
$R = \{(C_0, C\,|\,C), (C_1, C'\,|\,C), (C_1, C\,|\,C'), (C_2, C'\,|\,C')\}$
is a bisimulation. $C\,|\,C$ may be considered the *parallel* implementation of C_0. □

Remark 5.1. **(Sequentiality vs concurrency)** Even if LTS isomorphism is the most discriminating (i.e., most concrete) equivalence one can define over LTSs, it is unfortunately already abstract enough to be unable to distinguish *parallelism* (or *concurrency*) from *sequentiality*! Indeed, the parallel execution of two actions a and b, denoted in CFM by the parallel process term $a.\mathbf{0}\,|\,b.\mathbf{0}$, generates an LTS isomorphic to the one for the choice between their two possible sequential orderings, in SFM denoted by the sequential process term $a.b.\mathbf{0}+b.a.\mathbf{0}$ (see Figure 5.2). This example is an instance of a more general fact we are going to prove: given a parallel CFM process p, we can find a sequential SFM process q such that the LTSs for p and q are isomorphic. For instance, for the CFM process $C\,|\,C$ of Example 5.1, we can define the SFM process D_0 as

$$\begin{aligned}
D_0 &\doteq inc.D_1 + inc.D_2,\\
D_1 &\doteq inc.D_3 + dec.D_0,\\
D_2 &\doteq inc.D_3 + dec.D_0,\\
D_3 &\doteq dec.D_1 + dec.D_2.
\end{aligned}$$

which has an associated LTS isomorphic to the LTS of $C\,|\,C$. Hence, SFM and CFM are equally expressive, up to LTS isomorphism. This surprising observation is a consequence of the fact that LTSs are intrinsically a sequential model of computation and cannot truly represent parallelism; models of this sort are called *interleaving models*. □

Now we provide the proof that CFM is as expressive as SFM, up to LTS isomorphism. First, we note that, by inductive syntactic definition, a CFM process is either sequential, hence an SFM process, or the parallel composition of two CFM processes. Then, we prove by structural induction that for any CFM process p, the LTS reachable from p is finite-state. The base case of the induction is guaranteed by Corollary 4.2, as it ensures that the LTS reachable from p, $\mathscr{C}_p = (\mathscr{P}_p, sort(p), \rightarrow_p, p)$, is finite-state for any SFM process p. Now, by induction, we can assume that the cardinality of the set of states reachable from a CFM process p_i is k_i, and moreover that $sort(p_i)$ is finite, for $i = 1,2$; by inspecting the SOS rules (Par$_1$) and (Par$_2$), it is an easy exercise to observe that the cardinality of the set of states reachable from the CFM process $p_1\,|\,p_2$ is $k_1 \times k_2$[1] and that $sort(p_1\,|\,p_2) = sort(p_1) \cup sort(p_2)$. Therefore, we have the following fact.

Proposition 5.2. (Finite-state LTS) *For any CFM process p, the LTS reachable from p, $\mathscr{C}_p = (\mathscr{P}_p, sort(p), \rightarrow_p, p)$, is finite-state.* □

Since the LTS $\mathscr{C}_p$ reachable from any CFM process p is reduced and finite-state, Theorem 4.3 ensures that there exists an SFM process q such that the reachable LTS $\mathscr{C}_q = (\mathscr{P}_q, sort(q), \rightarrow_q, q)$ is isomorphic to $\mathscr{C}_p$. Summing up, if the intended semantics is LTS isomorphism (or any weaker equivalence, such as trace equivalence), CFM is as expressive as SFM, so that the addition of the parallel operator is inessential.

Remark 5.2. **(CFM languages = regular languages)** A set $L \subseteq \mathscr{L}^*$ is a CFM language if there exists a CFM process p such that $L = WCTr(p)$. By the observation above, it follows that the class of CFM languages coincides with the class of SFM languages, which, in turn, coincides with the class of regular languages, as discussed in Remark 4.3. □

[1] In general, if we have a compound process $p_1\,|\,p_2\,|\ldots|\,p_n$, where each p_i generates an LTS with ten states, then the LTS for $p_1\,|\,p_2\,|\ldots|\,p_n$ has 10^n states, i.e., the state space of a compound system grows exponentially w.r.t. the number of components. This phenomenon is sometimes called the *state space explosion problem*.

$$(\mathrm{Pref}^s)\ \frac{}{\mu.p \xrightarrow{\{\mu\}}_s p} \qquad (\mathrm{Cons}^s)\ \frac{p \xrightarrow{\{\mu\}}_s p'}{C \xrightarrow{\{\mu\}}_s p'} \quad C \doteq p$$

$$(\mathrm{Sum}_1^s)\ \frac{p \xrightarrow{\{\mu\}}_s p'}{p+q \xrightarrow{\{\mu\}}_s p'} \qquad (\mathrm{Sum}_2^s)\ \frac{q \xrightarrow{\{\mu\}}_s q'}{p+q \xrightarrow{\{\mu\}}_s q'}$$

$$(\mathrm{Par}_1^s)\ \frac{p \xrightarrow{M}_s p'}{p \,|\, q \xrightarrow{M}_s p' \,|\, q} \qquad (\mathrm{Par}_2^s)\ \frac{q \xrightarrow{M}_s q'}{p \,|\, q \xrightarrow{M}_s p \,|\, q'}$$

$$(\mathrm{Par}_3^s)\ \frac{p \xrightarrow{M_1}_s p' \qquad q \xrightarrow{M_2}_s q'}{p \,|\, q \xrightarrow{M_1 \oplus M_2}_s p' \,|\, q'}$$

Table 5.2 Step operational semantics

Bisimulation equivalence over the interleaving LTS is sometimes called *interleaving* bisimilarity, and denoted by $\sim$ as usual. Interleaving bisimilarity is a congruence for the action prefixing operator and for the choice operator, as proved in Theorem 4.4. Now we show that it is a congruence also for parallel composition.

Theorem 5.2. (Congruence) *For any $p,q \in \mathscr{P}_{CFM}$, if $p \sim q$, then $p \,|\, r \sim q \,|\, r$ for all $r \in \mathscr{P}_{CFM}$.*

Proof. *Assume R is a bisimulation with $(p,q) \in R$. We show that $R' = \{(p' \,|\, r', q' \,|\, r') \mid (p',q') \in R, r' \in \mathscr{P}_{CFM}\}$ is a bisimulation. As $(p,q) \in R$, the thesis follows. Consider $(p' \,|\, r', q' \,|\, r') \in R'$, so that $(p',q') \in R$. If $p' \,|\, r' \xrightarrow{\mu} s$, then this is due to $p' \xrightarrow{\mu} p''$ and $s = p'' \,|\, r'$ (rule (Par$_1$)), or to $r' \xrightarrow{\mu} r''$ and $s = p' \,|\, r''$ (rule (Par$_2$)). In the former case, since $(p',q') \in R$, we have $q' \xrightarrow{\mu} q''$ with $(p'',q'') \in R$, and by rule (Par$_1$) also $q' \,|\, r' \xrightarrow{\mu} q'' \,|\, r'$ with $(p'' \,|\, r', q'' \,|\, r') \in R'$. In the latter case, by rule (Par$_2$) we have $q' \,|\, r' \xrightarrow{\mu} q' \,|\, r''$ with $(p' \,|\, r'', q' \,|\, r'') \in R'$. The symmetric case when $q' \,|\, r'$ moves first is analogous, and so omitted. The symmetric thesis $r \,|\, p \sim r \,|\, q$ is implied by the fact that the parallel operator is commutative w.r.t. $\sim$.* □

5.1.2 Step Semantics

As CFM contains the parallel operator, it can be equipped with a semantics in terms of step transition systems (STSs for short). The CFM step transition system $\mathscr{C}_{CFM}^{step}$ is the triple $(\mathscr{P}_{CFM}, \mathscr{M}_{fin}(Act), \rightarrow_s)$ where $\rightarrow_s \subseteq \mathscr{P}_{CFM} \times \mathscr{M}_{fin}(Act) \times \mathscr{P}_{CFM}$ is the least transition relation generated by the axiom and rules in Table 5.2.

A sequential process can perform step transitions labeled with a singleton, only. This can be proven, for any $p \in \mathscr{P}_{CFM}^{seq}$, by induction on the proof of $p \xrightarrow{M}_s p'$. The base case is axiom (Pref^s) and the label is a singleton. By induction, we can assume that the premise of rules (Sum$_1^s$), (Sum$_2^s$) or (Cons^s) is labeled with a singleton, as it originates from a sequential process, and so also the conclusion of these rules is labeled with a singleton. Instead, for a parallel term, three rules can be applied: (Par$_1^s$) and its symmetric (Par$_2^s$) state that a multiset M performed by one of the two subcomponents is also a step of the compound system; rule (Par$_3^s$) is responsible for the creation of non-singleton steps: if p can perform the step M_1 and q the step M_2, respectively, then the compound system $p \,|\, q$ can perform in parallel the step $M_1 \oplus M_2$, i.e., the multiset union of the two local steps M_1 and M_2.

The STS for CFM is fully concurrent (see Definition 2.25), as proven in the following proposition.

Proposition 5.3. (Fully concurrent STS) *$\mathscr{C}_{CFM}^{step}$ is fully concurrent.*

Proof. We prove that if $p \xrightarrow{M_1 \oplus M_2}_s p'$, where $M_1 \neq \emptyset \neq M_2$, then there exist r_1 and r_2 such that $p \xrightarrow{M_1}_s r_1 \xrightarrow{M_2}_s p'$ and $p \xrightarrow{M_2}_s r_2 \xrightarrow{M_1}_s p'$. The proof is by induction on the proof of $p \xrightarrow{M_1 \oplus M_2}_s p'$. Therefore, p must be of the form $p_1 \,|\, p_2$ and the transition is due to one of the following three cases:

- *$p_1 \xrightarrow{M_1}_s p_1'$, $p_2 \xrightarrow{M_2}_s p_2'$ and $p' = p_1' \,|\, p_2'$, i.e., rule (Par$_3^s$) has been applied; this is the base case of the induction. Then, also $p_1 \,|\, p_2 \xrightarrow{M_1}_s p_1' \,|\, p_2 \xrightarrow{M_2}_s p_1' \,|\, p_2'$ and $p_1 \,|\, p_2 \xrightarrow{M_2}_s p_1 \,|\, p_2' \xrightarrow{M_1}_s p_1' \,|\, p_2'$ are derivable transition sequences, by means of applications of rules (Par$_1^s$) and (Par$_2^s$), as required.*
- *$p_1 \xrightarrow{M_1 \oplus M_2}_s p_1'$ and $p' = p_1' \,|\, p_2$. By induction, we can assume that there exist q_1 and q_2 such that $p_1 \xrightarrow{M_1}_s q_1 \xrightarrow{M_2}_s p_1'$ and $p_1 \xrightarrow{M_2}_s q_2 \xrightarrow{M_1}_s p_1'$. Therefore, by rule (Par$_1^s$), also $p_1 \,|\, p_2 \xrightarrow{M_1}_s q_1 \,|\, p_2 \xrightarrow{M_2}_s p_1' \,|\, p_2$ and $p_1 \,|\, p_2 \xrightarrow{M_2}_s q_2 \,|\, p_2 \xrightarrow{M_1}_s p_1' \,|\, p_2$ are derivable, as required.*
- *$p_2 \xrightarrow{M_1 \oplus M_2}_s p_2'$ and $p' = p_1 \,|\, p_2'$. Symmetric to the above case, and so omitted.* □

Remark 5.3. **(Notation)** For any $p \in \mathscr{P}_{CFM}$, the STS reachable from p is $\mathscr{C}_p^{step} = (\mathscr{P}_p^{step}, sort(p)_s, \rightarrow_s^p, p)$, where $\mathscr{P}_p^{step}$ is the set of processes reachable from p, $sort(p)_s$ is the set of multisets of actions that can be performed by p (formally, $sort(p)_s = \{M \in \mathscr{M}_{fin}(Act) \mid \exists p'.p \longrightarrow^* p' \xrightarrow{M}_s \}$) and $\rightarrow_s^p$ is the restriction of the transition relation to $\mathscr{P}_p^{step} \times sort(p)_s \times \mathscr{P}_p^{step}$. □

Step bisimilarity, denoted $\sim_{step}$, is ordinary bisimulation equivalence over STSs. Step bisimilarity is more discriminating than interleaving bisimilarity $\sim$ for CFM. As a matter of fact, any interleaving transition $p \xrightarrow{\mu} p'$ has one corresponding step transition $p \xrightarrow{\{\mu\}}_s p'$, which is obtained by mimicking the proof of $p \xrightarrow{\mu} p'$, by using for each rule in Tables 4.2 and 5.1 the corresponding rule with the same name (plus the superscript s) in Table 5.2. Symmetrically, any step transition $p \xrightarrow{\{\mu\}}_s p'$ labeled with a singleton $\{\mu\}$ is derivable only by not using rule (Par$_3^s$); hence, its proof can be mimicked by the interleaving operational rules, as hinted above. Therefore, a step bisimulation $R \subseteq \mathscr{P}_{CFM} \times \mathscr{P}_{CFM}$ is also an interleaving bisimulation: if $(p,q) \in R$

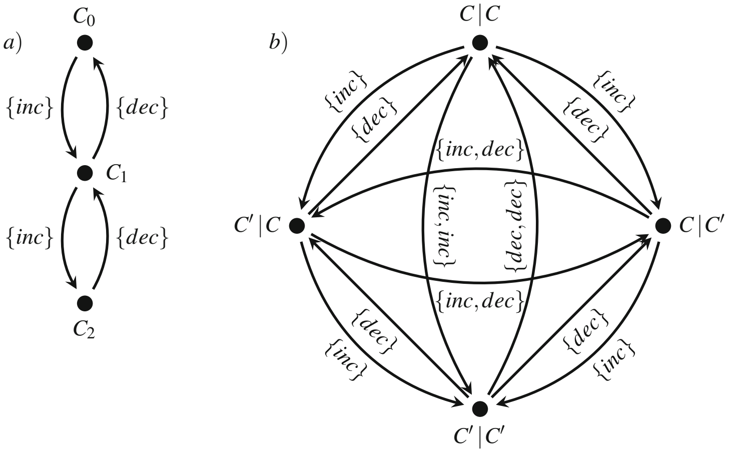


Fig. 5.3 Two 2-bounded counters: step transition systems

and $p \xrightarrow{\mu} p'$, then also $p \xrightarrow{\{\mu\}}_s p'$ by the argument above; as R is a step bisimulation, there exists q' such that $q \xrightarrow{\{\mu\}}_s q'$ with $(p', q') \in R$; and so, by the argument above, also $q \xrightarrow{\mu} q'$ with $(p', q') \in R$; symmetrically, if q moves first. Hence, R is also an interleaving bisimulation.

Proposition 5.4. *For any $p, q \in \mathscr{P}_{CFM}$, if $p \sim_{step} q$, then $p \sim q$.* □

Example 5.2. Continuing Example 5.1, Figure 5.3 outlines the step transition systems generated by the step semantics for the 2-bounded counters C_0 and $C|C$. Note that, although they are interleaving bisimilar, the two are not step bisimilar: $C|C$ can perform the step $\{inc, inc\}$, because we have two independent components C that can run in parallel, while this parallel step is not possible for the sequential process C_0. □

This example illustrates that CFM is more expressive than SFM when considering a non-interleaving semantics such as the step semantics outlined above, as any SFM process can generate only singleton-labeled steps.

Step bisimilarity is a congruence for the operators of CFM.

Theorem 5.3. (Step congruence) *For any $p, q \in \mathscr{P}_{CFM}^{seq}$, if $p \sim_{step} q$, then*

1) $\mu.p \sim_{step} \mu.q$ *for all $\mu \in Act$.*

Moreover, for any p, q in syntactic category s, if $p \sim_{step} q$, then

2) $p + r \sim_{step} q + r$ *for all r in syntactic category s.*

Additionally, for any $p, q \in \mathscr{P}_{CFM}$, if $p \sim_{step} q$, then

3) $p|r \sim_{step} q|r$ *for all $r \in \mathscr{P}_{CFM}$.*

Proof. Assume R is a step bisimulation such that $(p, q) \in R$.

$$dec(p \,|\, p') = dec(p) \oplus dec(p')$$

Table 5.3 Decomposition function for parallel composition

For case 1), the relation $R_1 = \{(\mu.p, \mu.q) \;\big|\; \mu \in Act\} \cup R$ *is a step bisimulation.*

For case 2), the relation $R_2 = \{(p+r, q+r) \;\big|\; r \text{ in syntactic category } s\} \cup R \cup \mathscr{I}$ *is a step bisimulation, where* $\mathscr{I} = \{(r,r) \;\big|\; r \in \mathscr{P}_{CFM}^{seq}\}$*. Note that the symmetric case* $r+p \sim_{step} r+q$ *is implied by the fact that the choice operator is commutative w.r.t.* $\sim_{step}$*, i.e.,* $p+q \sim_{step} q+p$*.*

For case 3), the relation $R_3 = \{(p' \,|\, r', q' \,|\, r') \;\big|\; (p', q') \in R, r' \in \mathscr{P}_{CFM}\}$ *is a step bisimulation. As* $(p,q) \in R$*, the thesis follows. Note that the symmetric case* $r \,|\, p \sim_{step} r \,|\, q$ *is implied by the fact that the parallel composition operator is commutative w.r.t.* $\sim_{step}$*, i.e.,* $p \,|\, q \sim_{step} q \,|\, p$*.* □

5.1.3 Operational Net Semantics

The operational net semantics for CFM is obtained by a little addition to the net semantics for SFM. As a matter of fact, the net $N_{CFM} = (S_{CFM}, Act, T_{CFM})$ is exactly the net N_{SFM} we have defined in Section 4.3. The infinite set S_{CFM} of CFM places is exactly $\mathscr{P}_{CFM}^{seq} \setminus \{\mathbf{0}\} = \mathscr{P}_{SFM} \setminus \{\mathbf{0}\} = S_{SFM}$, i.e., the set of all SFM places. This because function $dec : \mathscr{P}_{CFM} \to \mathscr{M}_{fin}(S_{CFM})$, defined in Table 4.3 for the SFM operators, is extended to parallel composition as outlined in Table 5.3. An easy induction proves that for any $p \in \mathscr{P}_{CFM}$, $dec(p)$ is a finite multiset of sequential processes.

Example 5.3. Let us consider the process $a.\mathbf{0} \,|\, (b.\mathbf{0} \,|\, a.\mathbf{0})$. The decomposition of this process is

$dec(a.\mathbf{0} \,|\, (b.\mathbf{0} \,|\, a.\mathbf{0})) = dec(a.\mathbf{0}) \oplus dec(b.\mathbf{0} \,|\, a.\mathbf{0}) = a.\mathbf{0} \oplus dec(b.\mathbf{0}) \oplus dec(a.\mathbf{0}) =$
$= \{a.\mathbf{0}, b.\mathbf{0}, a.\mathbf{0}\} = 2 \cdot a.\mathbf{0} \oplus b.\mathbf{0}$.

Moreover, if we consider the bounded counter $C \,|\, C$ of Example 5.1, we have that $dec(C \,|\, C) = 2 \cdot C$. □

Of course, function dec is not injective, because it considers the parallel operator to be commutative and associative, with $\mathbf{0}$ as neutral element. In fact, $dec((p \,|\, q) \,|\, r)$ $= dec(p \,|\, (q \,|\, r))$, $dec(p \,|\, q) = dec(q \,|\, p)$ and $dec(p \,|\, \mathbf{0}) = dec(p)$. However, dec is surjective. Take any finite multiset of places $m = k_1 \cdot s_1 \oplus \ldots \oplus k_n \cdot s_n$, for $n \geq 0$ (if $n = 0$, then $m = \emptyset$), where $s_i \in S_{CFM}$ and $k_i > 0$, for $i = 1, \ldots, n$. Then, the CFM process $p = s_1^{k_1} \,|\, \ldots \,|\, s_n^{k_n}$, where $s^1 = s$ and $s^{n+2} = s \,|\, s^{n+1}$, is such that $dec(p) = m$ (if $n = 0$, then $p = \mathbf{0}$). Hence, the following proposition follows.

Proposition 5.5. *Function* $dec : \mathscr{P}_{CFM} \to \mathscr{M}_{fin}(S_{CFM})$ *is surjective.* □

The set T_{CFM} of net transitions is the least set of transitions generated by means of the axiom and rules in Table 4.4, as each sequential process performs its activity in isolation, without interacting with other sequential components. Therefore, $T_{CFM} = T_{SFM}$, and so $N_{CFM} = N_{SFM}$. However, while for any $p \in \mathscr{P}_{SFM}$, $Net(p)$ is a *sequential* finite-state machine (as proved in Corollary 4.4), we have that for any $p \in \mathscr{P}_{CFM}$, $Net(p)$ is a *concurrent* finite-state machine.

Definition 5.1. Let p be a CFM process. The P/T system statically associated with p is $Net(p) = (S_p, A_p, T_p, m_0)$, where $m_0 = dec(p)$ and

$$\begin{aligned} S_p &= \llbracket dom(m_0) \rangle \quad \text{computed in } N_{CFM}, \\ T_p &= \{t \in T_{CFM} \mid S_p \llbracket t \rangle \}, \\ A_p &= \{\mu \in Act \mid \exists t \in T_p . \mu = l(t)\}. \end{aligned}$$ □

The following propositions present three facts that are obviously true by construction of the net $Net(p)$ associated with a CFM process p.

Proposition 5.6. *For any $p \in \mathscr{P}_{CFM}$, $Net(p)$ is a statically reduced P/T net.*[2] □

Proposition 5.7. *If $dec(p) = dec(q)$, then $Net(p) = Net(q)$.* □

Proposition 5.8. *For any $p_1, p_2 \in \mathscr{P}_{CFM}$, if $Net(p_i) = (S_i, A_i, T_i, m_i)$ for $i = 1, 2$, then $Net(p_1 \mid p_2) = (S_1 \cup S_2, A_1 \cup A_2, T_1 \cup T_2, m_1 \oplus m_2)$.*

Proof. *As $dec(p_i) = m_i$ for $i = 1, 2$, it follows that $dec(p_1 \mid p_2) = dec(p_1) \oplus dec(p_2) = m_1 \oplus m_2$. By induction on the static reachability relation $dom(m_1 \oplus m_2) \Longrightarrow^* V$, we shall prove that $V \subseteq S_1 \cup S_2$. This is enough, because all the places in S_1 or S_2 are statically reachable from $dom(m_1)$ or $dom(m_2)$, so that the set of places of $Net(p_1 \mid p_2)$ must be $S_1 \cup S_2$. The base case of the induction is $dom(m_1 \oplus m_2) \Longrightarrow^* dom(m_1 \oplus m_2)$ and this case is trivial: $dom(m_1 \oplus m_2) \subseteq S_1 \cup S_2$ because $dom(m_1 \oplus m_2) = dom(m_1) \cup dom(m_2)$ and $dom(m_i) \subseteq S_i$ for $i = 1, 2$. The inductive case is as follows: $dom(m_1 \oplus m_2) \Longrightarrow^* V_j \overset{t}{\Longrightarrow} V_{j+1}$. By induction, we can assume that $V_j \subseteq S_1 \cup S_2$, and so $V_j = V_j^1 \cup V_j^2$, with $V_j^i \subseteq S_i$ for $i = 1, 2$. Note that, by Proposition 4.6, any transition $t \in T_{CFM}$ is such that $|{}^\bullet t| = 1$. Therefore, if t is statically enabled at V_j, then ${}^\bullet t \subseteq V_j^i$ for some $i = 1, 2$ and so $dom(t^\bullet) \subseteq S_i$ because $Net(p_i)$ is statically reduced. Hence, the set $V_{j+1} = V_j \cup dom(t^\bullet)$ of places statically reachable in one step from V_j is a subset of $S_1 \cup S_2$. Therefore, the sets of places, labels and transitions of $Net(p_1 \mid p_2)$ are obtained by simply joining the corresponding sets of the two nets $Net(p_1)$ and $Net(p_2)$.* □

Example 5.4. Continuing Examples 5.1 and 5.2, Figure 5.4 outlines the nets associated with the sequential 2-bounded counter C_0, in (a), and with its parallel implementation $C \mid C$, in (b). Note that the latter net is a concurrent FSM, as the initial marking is not a singleton. □

[2] Since, for FSM nets, the notion of statically reachable subnet and dynamically reachable subnet coincide, $Net(p)$ is also dynamically reduced.

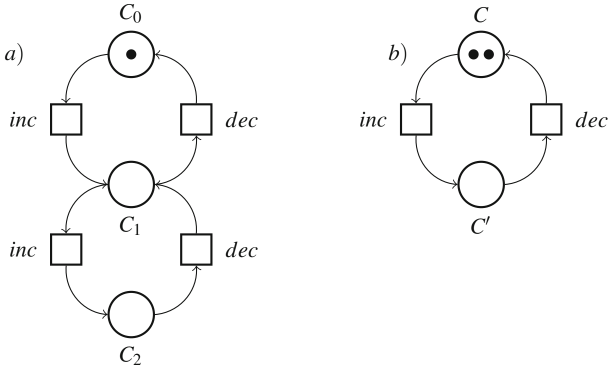


Fig. 5.4 Two 2-bounded counters: net semantics

It is an easy observation that all the reachable markings from the initial marking $dec(p)$ have a cardinality not greater than that of $dec(p)$, so that $Net(p)$ is a k-bounded net, with $k = |dec(p)|$.

Proposition 5.9. *Given a CFM process p, any marking m in the set $[dec(p)\rangle$ of reachable markings is such that $|m| \leq |dec(p)|$.*

Proof. *By induction on the length of the firing sequences. The base case is when the firing sequence is made of the initial marking only: $dec(p) \in [dec(p)\rangle$ and the thesis holds trivially. Now, assume that $m_1 \in [dec(p)\rangle$ and, for some transition $t \in T_{CFM}$, $m_1[t\rangle m_2$; we want to prove that $|m_2| \leq |dec(p)|$. By induction, we can assume that $|m_1| \leq |dec(p)|$. Since t is a transition with singleton pre-set and singleton or empty post-set by Proposition 4.6, it follows that $|m_2| = |m_1| - |{}^\bullet t| + |t^\bullet| \leq |m_1|$, and so the thesis follows by transitivity.* □

Lemma 5.1. *For any CFM process p, $\bigcup_{s \in dom(dec(p))} sub(s) \subseteq sub(p)$.*

Proof. *By induction on the structure of p. If p is $\mathbf{0}$, then $dec(p) = \emptyset$ and so $\emptyset \subseteq sub(\mathbf{0}) = \{\mathbf{0}\}$. If p is any other sequential process, then $dec(p) = \{p\}$, and so the thesis follows trivially. If $p = p_1 \,|\, p_2$, by induction we can assume that the thesis holds for p_1 and p_2. Hence, $\bigcup_{s \in dom(dec(p_1 \,|\, p_2))} sub(s) =$
$\bigcup_{s \in dom(dec(p_1)) \cup dom(dec(p_2))} sub(s) = \bigcup_{s \in dom(dec(p_1))} sub(s) \cup \bigcup_{s \in dom(dec(p_2))} sub(s)$
$\subseteq sub(p_1) \cup sub(p_2) = sub(p_1 \,|\, p_2)$.* □

Theorem 5.4. (Finite number of places) *For any CFM process p, the set S_p of its places statically reachable from $dom(dec(p))$ is finite.*

Proof. *By induction on the static reachability relation $\Longrightarrow^*$ (Definition 3.9), we prove that any place s that is statically reachable from $dom(dec(p))$ is a subterm of p. Since $sub(p)$ is finite by Theorem 5.1, the thesis follows trivially.*

The base case is $dom(dec(p)) \Longrightarrow^* dom(dec(p))$. As for any $s \in dom(dec(p))$, $s \in sub(s)$ by Proposition 5.1, we have that $dom(dec(p)) \subseteq \bigcup_{s \in dom(dec(p))} sub(s) \subseteq sub(p)$, the latter holding by Lemma 5.1.

Now, assume that $dom(dec(p)) \Longrightarrow^* S_1 \stackrel{t}{\Longrightarrow} S_2$ for some transition $t \in T_{CFM}$; we want to prove that any place in S_2 is a subterm of p.

By induction, we can assume that $S_1 \subseteq sub(p)$. Since $t \in T_{CFM} = T_{SFM}$, by Proposition 4.6, $^\bullet t$ is a singleton and $t^\bullet$ is a singleton or empty; the latter case is trivial, as no new place is reached. So, assume $t = (\{s_1\}, \mu, \{s_2\})$; hence, $s_1 \in S_1$ and $S_2 = S_1 \oplus \{s_2\}$. By Proposition 4.7, we have that $s_1 \xrightarrow{\mu} s_2$ is an LTS transition. By Lemma 4.1, $s_2 \in sub(s_1)$. Since $s_1 \in S_1$ and $S_1 \subseteq sub(p)$, we have that $s_1 \in sub(p)$ and so, by Proposition 5.1, also $sub(s_1) \subseteq sub(p)$; therefore, $s_2 \in sub(p)$ by transitivity, as required.

Hence, the set S_p of its reachable places is finite. □

As a corollary to the above, we have that for any CFM process p, the set $[dec(p)\rangle$ of its reachable markings is finite, because the set S_p of its statically reachable places coincides with the set of its dynamically reachable places (as for any FSM net).

Corollary 5.1. (Finite number of reachable markings) *For any CFM process p, the set $[dec(p)\rangle$ of its (dynamically) reachable markings is finite.*

Proof. *By Theorem 5.4, the set S_p of places is finite. By Proposition 5.9, any reachable marking has a cardinality less than or equal to $|dec(p)|$. Therefore, assuming $|S_p| = n$ and $|dec(p)| = k$, the number of reachable markings cannot be more than the binomial coefficient $\binom{n+k}{k} = \frac{(n+k)!}{n!k!}$. In fact, the distribution of i tokens over n places is given by the binomial coefficient $\binom{n+i-1}{i}$; as the tokens to be distributed over the n places is any $0 \leq i \leq k$, the total number of different distributions of tokens can be $\sum_{i=0}^{k} \binom{n+i-1}{i} = \binom{n+k}{k}$.* □

Theorem 5.5. (Finite P/T Petri net) *For any CFM process p, the P/T net reachable from p, $Net(p) = (S_p, A_p, T_p, dec(p))$, is finite.*

Proof. *By Theorem 5.4, the set S_p of the places reachable from p is finite. By Proposition 4.6 the pre-set of any derivable transition is a singleton. By Lemma 4.2, the set T^q of transitions with pre-set $\{q\}$ is finite, for any $q \in S_p$. Hence, also $T_p = \bigcup_{q \in S_p} T^q$ must be finite, being a finite union of finite sets.* □

Corollary 5.2. (Concurrent finite-state machine) *For any CFM process p, $Net(p)$ is a concurrent finite-state machine.*

Proof. *By Theorem 4.6, $Net(p)$ is a finite P/T net. By Proposition 4.6, for any transition $t \in T_p$, we have $|t^\bullet| \leq |^\bullet t| = 1$, i.e., $Net(p)$ is a finite-state machine. Finally, it is a concurrent FSM, as $dec(p)$ is not necessarily a singleton.* □

Hence, *only* concurrent finite-state machines can be represented by CFM processes.

5.1.4 Representing All Concurrent Finite-State Machines

It is not a surprise that *all* concurrent finite-state machines can be represented by CFM processes. The construction described in Section 4.4 for sequential FSMs can be easily extended to concurrent FSMs: the resulting process is not a single constant, rather the parallel composition of as many instances of each constant as there are tokens in its corresponding place.

Definition 5.2. (Translating concurrent FSMs into CFM process terms) Let $N(m_0) = (S, A, T, m_0)$ — with $S = \{s_1, \dots, s_n\}$, $A \subseteq Act$, $T = \{t_1, \dots, t_k\}$, and $l(t_j) = \mu_j$ — be a concurrent finite-state machine. Function $\mathscr{T}_{CFM}(-)$, from concurrent finite-state machines to CFM processes, is defined as

$$\mathscr{T}_{CFM}(N(m_0)) = \underbrace{C_1|\cdots|C_1}_{m_0(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m_0(s_n)}$$

where each C_i is equipped with a defining equation $C_i \doteq c_i^1 + \dots + c_i^k$ (with $C_i \doteq \mathbf{0}$ if $k = 0$), and each summand c_i^j, for $j = 1, \dots, k$, is equal to

- $\mathbf{0}$, if $s_i \not\in {}^\bullet t_j$;
- $\mu_j.\mathbf{0}$, if ${}^\bullet t_j = \{s_i\}$ and $t_j^\bullet = \emptyset$;
- $\mu_j.C_h$, if ${}^\bullet t_j = \{s_i\}$ and $t_j^\bullet = \{s_h\}$. □

Note that $\mathscr{T}_{CFM}(N(m_0))$ is a CFM process: in fact, there are finitely many constants involved (one for each place) and each constant has a body that is sequential; moreover, parallel composition is used at the top level only. Therefore, the following proposition holds by Proposition 5.6 and Corollary 5.2.

Proposition 5.10. *Given a concurrent finite-state machine $N(m_0) = (S, A, T, m_0)$, $Net(\mathscr{T}_{CFM}(N(m_0)))$ is a (statically) reduced, concurrent finite-state machine.* □

Example 5.5. Let us consider the net in Figure 5.5. The CFM process associated with such a concurrent finite-state machine is $p = C_1 \,|\, C_1 \,|\, C_2 \,|\, C_4 \,|\, C_4$, where the four constants C_i, corresponding to the four places s_i, for $i = 1, \dots, 4$, are defined as

$$\begin{aligned} C_1 &\doteq a.C_3 + \mathbf{0} + \mathbf{0}, \\ C_2 &\doteq \mathbf{0} + b.C_3 + \mathbf{0}, \\ C_3 &\doteq \mathbf{0} + \mathbf{0} + \mathbf{0}, \\ C_4 &\doteq \mathbf{0} + \mathbf{0} + c.C_4. \end{aligned}$$

It is obvious that $Net(p)$ is a net isomorphic to the original net in Figure 5.5, where the isomorphism f is the bijection $f(s_i) = C_i$ for $i = 1, \dots, 4$. □

Now we are ready to state our main result, the so-called *representability theorem*.

Theorem 5.6. (Representability theorem 2) *Let $N(m_0) = (S, A, T, m_0)$ be a (statically) reduced, concurrent finite-state machine such that $A \subseteq Act$, and let $p = \mathscr{T}_{CFM}(N(m_0))$. Then, $Net(p)$ is isomorphic to $N(m_0)$.*

Proof. The details of this proof are analogous to the proof of the following representability theorem for BPP, Theorem 5.13, in Section 5.2.3. □

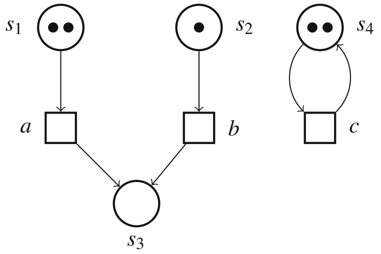


Fig. 5.5 A concurrent finite-state machine

5.1.5 Soundness

In this section, we want to prove that the operational net semantics preserves the step semantics of Section 5.1.2. More precisely, we want to show that any CFM process p, in the STS semantics, is step bisimilar to $dec(p)$, in the step marking graph $SMG(Net(p))$.

Proposition 5.11. *For any $p \in \mathscr{P}_{CFM}$, if $p \xrightarrow{M}_s p'$, then $dec(p)[G\rangle dec(p')$ with $l(G) = M$.*

Proof. *By induction on the proof of $p \xrightarrow{M}_s p'$. The base case is axiom (Pref^s): $\mu.p \xrightarrow{\{\mu\}}_s p$. As $dec(\mu.p) = \{\mu.p\}$, by axiom (pref) of Table 4.4, the net transition $t = \{\mu.p\} \xrightarrow{\mu} dec(p)$ is derivable, and so the singleton step $dec(\mu.p)[\{t\}\rangle dec(p)$ is also derivable, as required. If the last rule used in deriving $p \xrightarrow{M}_s p'$ is (Cons^s), then $M = \{\mu\}$ for some suitable μ, and $p = C$ for some constant C, with $C \doteq q$. The premise of the rule is $q \xrightarrow{\{\mu\}}_s p'$, and by induction we know that there exists a transition t' such that $dec(q)[\{t'\}\rangle dec(p')$ with $l(t') = \mu$. Since q is sequential, this step is possible only if $t' = \{q\} \xrightarrow{\mu} dec(p')$ is derivable by the rules in Table 4.4; hence, by rule (cons), also the net transition $t = \{C\} \xrightarrow{\mu} dec(p')$ is derivable, and so the singleton step $dec(C)[\{t\}\rangle dec(p')$ is also derivable, as required. The two cases when rule (Sum$^s_1$) or rule (Sum$^s_2$) is the last rule used in deriving $p \xrightarrow{M}_s p'$ are similar to the above, hence omitted. If the last rule used in deriving $p \xrightarrow{M}_s p'$ is (Par$^s_1$), then $p = p_1 \,|\, p_2$, $p' = p'_1 \,|\, p_2$ and the premise transition is $p_1 \xrightarrow{M}_s p'_1$. By induction, we know that $dec(p_1)[G\rangle dec(p'_1)$ with $l(G) = M$. Then, also $dec(p_1 \,|\, p_2) = dec(p_1) \oplus dec(p_2)[G\rangle dec(p'_1) \oplus dec(p_2) = dec(p'_1 \,|\, p_2)$, as required. The case of rule (Par$^s_2$) is symmetric, hence omitted. If the last rule used in deriving $p \xrightarrow{M}_s p'$ is (Par$^s_3$), then $p = p_1 \,|\, p_2$, $p' = p'_1 \,|\, p'_2$ and the two premise transitions are $p_1 \xrightarrow{M_1}_s p'_1$, $p_2 \xrightarrow{M_2}_s p'_2$, with $M = M_1 \oplus M_2$. By induction, we know that $dec(p_1)[G_1\rangle dec(p'_1)$, with $l(G_1) = M_1$, and $dec(p_2)[G_2\rangle dec(p'_2)$, with $l(G_2) = M_2$; therefore, $dec(p_1 \,|\, p_2) = dec(p_1) \oplus dec(p_2)[G_1 \oplus G_2\rangle dec(p'_1) \oplus dec(p'_2) = dec(p'_1 \,|\, p'_2)$, as required.* □

Proposition 5.12. *For any $p \in \mathscr{P}_{CFM}$, if $dec(p)[G\rangle m$, with $l(G) = M$, then there exists $p' \in \mathscr{P}_{CFM}$ such that $p \xrightarrow{M}_s p'$ and $dec(p') = m$.*

Proof. *By induction on the definition of $dec(p)$. We proceed by case analysis. If $p = \mathbf{0}$, then $dec(p) = \emptyset$; this case is vacuously true, as no transition is executable from $\emptyset$. If p is any other sequential process, then $dec(p) = \{p\}$. If $\{p\}[G\rangle m$, then G is a singleton $\{t\}$, with $t = (\{p\}, \mu, dec(p'))$ for some suitable μ and p', by Proposition 4.6. The proof of the net transition t can be mimicked, by the corresponding rules in Table 5.2, to produce the transition $p \xrightarrow{\{\mu\}}_s p'$, as required. If $p = p_1 \,|\, p_2$, then $dec(p) = dec(p_1) \oplus dec(p_2)$. As the pre-set of any net transition is a singleton by Proposition 4.6, only the following three sub-cases are possible:*

(i) $dec(p_1)[G\rangle m_1$ and $m = m_1 \oplus dec(p_2)$: in this case, by induction, we know that there exists p_1' such that $p_1 \xrightarrow{M}_s p_1'$ and $dec(p_1') = m_1$; therefore, by rule (Par_1^s), also $p_1 \,|\, p_2 \xrightarrow{M}_s p_1' \,|\, p_2$ is derivable, with $dec(p_1' \,|\, p_2) = m$, as required.

(ii) $dec(p_2)[G\rangle m_2$ and $m = dec(p_1) \oplus m_2$: this is the symmetric case of the above, hence omitted.

(iii) $dec(p_1)[G_1\rangle m_1$, with $l(G_1) = M_1$, $dec(p_2)[G_2\rangle m_2$, with $l(G_2) = M_2$, $M = M_1 \oplus M_2$ and $m = m_1 \oplus m_2$. In this case, by induction, we know that there exist p_1' and p_2' such that $p_1 \xrightarrow{M_1}_s p_1'$, $p_2 \xrightarrow{M_2}_s p_2'$, with $dec(p_1') = m_1$ and $dec(p_2') = m_2$. By rule (Par_3^s) of Table 5.2, also transition $p_1 \,|\, p_2 \xrightarrow{M_1 \oplus M_2}_s p_1' \,|\, p_2'$ is derivable, with $dec(p_1' \,|\, p_2') = m$, as required. □

Theorem 5.7. *For any $p \in \mathscr{P}_{CFM}$, $p \sim_{step} dec(p)$.*

Proof. *The relation $R = \{(p, dec(p)) \;\big|\; p \in \mathscr{P}_{CFM}\}$ is a bisimulation between the step transition system $\mathscr{C}_p^{step} = (\mathscr{P}_p^{step}, sort(p)_s, \rightarrow_s^p, p)$, originating from the step semantics of Section 5.1.2, and the step marking graph $SMG(Net(p))$. Given $(p, dec(p)) \in R$, if $p \xrightarrow{M}_s p'$, then by Proposition 5.11 we have $dec(p)[G\rangle dec(p')$, with $l(G) = M$ and $(p', dec(p')) \in R$; conversely, if $dec(p)[G\rangle m$ with $l(G) = M$, then by Proposition 5.12 there exists $p' \in \mathscr{P}_{CFM}$ such that $p \xrightarrow{M}_s p'$ and $dec(p') = m$, so that $(p', dec(p')) \in R$, as required.* □

Example 5.6. Consider the CFM process $A \,|\, B$, where $A \doteq a.B$ and $B \doteq b.A$. Its STS semantics is depicted in Figure 5.6(a), while the step marking graph SMG associated with $dec(A \,|\, B) = A \oplus B$ is depicted in (b). Note that the two structures are (step) bisimilar, but not isomorphic. In general, as dec is not injective, the step marking graph associated with a process p can be smaller, sometimes even considerably smaller, than the STS semantics of p. □

5.1.6 Denotational Net Semantics

The denotational net semantics for CFM is based on the definition described in Table 4.5 for the operators of SFM, while the denotational definition for the parallel

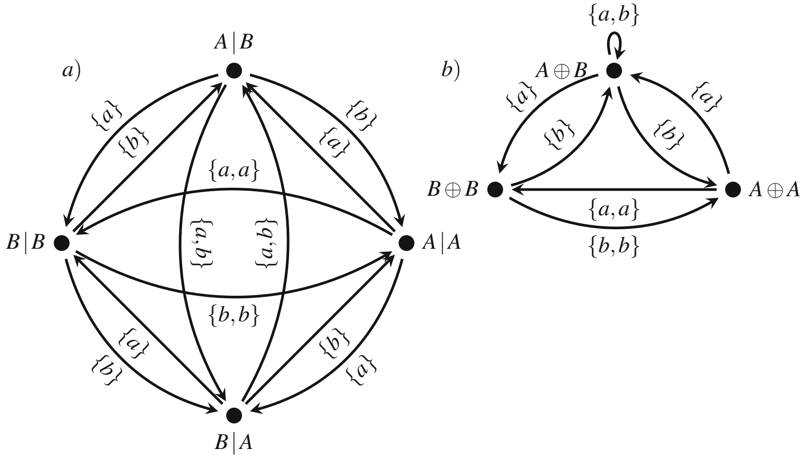


Fig. 5.6 STS semantics and SMG semantics for process $A \,|\, B$ of Example 5.6

$[\![p_1 \,|\, p_2]\!]_I = (S, A, T, m_0)$ given $[\![p_i]\!]_I = (S_i, A_i, T_i, m_i)$ for $i = 1, 2$, and where

$S = S_1 \cup S_2,\ A = A_1 \cup A_2,\ T = T_1 \cup T_2,\ m_0 = m_1 \oplus m_2$

Table 5.4 Denotational net semantics for parallel composition

operator is given in Table 5.4. This definition is very intuitive: when computing the net for $p_1 \,|\, p_2$, given the nets for p_1 and p_2, it is enough to join all the places and transitions of the two constituent nets, taking the multiset union of the respective initial markings to be the new initial marking. The following two examples may be helpful in clarifying the idea.

Example 5.7. Consider the CFM process $(b.\mathbf{0} \,|\, a.A) \,|\, b.\mathbf{0}$, where $A \doteq \mathbf{0}$. We first define the net for the constituent SFM processes $b.\mathbf{0}$ and $a.A$. They are:
$[\![b.\mathbf{0}]\!]_\emptyset = (\{b.\mathbf{0}\}, \{b\}, \{(\{b.\mathbf{0}\}, b, \emptyset)\}, \{b.\mathbf{0}\})$, and
$[\![a.A]\!]_\emptyset = (\{a.A, A\}, \{a\}, \{(\{a.A\}, a, \{A\})\}, \{a.A\})$.
Then, the concurrent FSM for $b.\mathbf{0} \,|\, a.A$ is
$[\![b.\mathbf{0} \,|\, a.A]\!]_\emptyset =$
$= (\{b.\mathbf{0}, a.A, A\}, \{a, b\}, \{(\{b.\mathbf{0}\}, b, \emptyset), (\{a.A\}, a, \{A\})\}, \{b.\mathbf{0}, a.A\})$.
And, finally, the net for $(b.\mathbf{0} \,|\, a.A) \,|\, b.\mathbf{0}$ is
$[\![(b.\mathbf{0} \,|\, a.A) \,|\, b.\mathbf{0}]\!]_\emptyset =$
$= (\{b.\mathbf{0}, a.A, A\}, \{a, b\}, \{(\{b.\mathbf{0}\}, b, \emptyset), (\{a.A\}, a, \{A\})\},\ \{b.\mathbf{0}, a.A, b.\mathbf{0}\})$,
which is the same net as in the previous step, but with one token more in $b.\mathbf{0}$. □

Example 5.8. Consider the CFM process $B \,|\, b.A \,|\, C \,|\, C$, where $B \doteq b.A$, $A \doteq a.b.A$ and $C \doteq a.C$. The nets for the SFM processes $b.A$ and B have been built in Example 4.6 and outlined in Figure 4.2. They are, respectively,

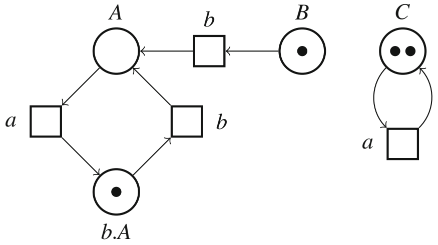


Fig. 5.7 The concurrent finite-state machine for $B \,|\, b.A \,|\, C \,|\, C$ of Example 5.8

$[\![b.A]\!]_\emptyset = (\{b.A, A\}, \{a, b\}, \{(\{A\}, a, \{b.A\}), (\{b.A\}, b, \{A\})\}, \{b.A\})$, and
$[\![B]\!]_\emptyset = (\{B, b.A, A\}, \{a, b\}, \{(\{B\}, b, \{A\}), (\{A\}, a, \{b.A\}), (\{b.A\}, b, \{A\})\}, \{B\})$.
The concurrent FSM associated with $B \,|\, b.A$ is
$[\![B \,|\, b.A]\!]_\emptyset =$
$= (\{B, b.A, A\}, \{a, b\}, \{(\{B\}, b, \{A\}), (\{A\}, a, \{b.A\}), (\{b.A\}, b, \{A\})\}, \{B, b.A\})$,
where the only addition to the net for B is one token in place $b.A$. The sequential FSM for C is
$[\![C]\!]_\emptyset = (\{C\}, \{a\}, \{(\{C\}, a, \{C\})\}, \{C\})$.
The concurrent FSM for $C \,|\, C$ is
$[\![C \,|\, C]\!]_\emptyset = (\{C\}, \{a\}, \{(\{C\}, a, \{C\})\}, \{C, C\})$.
And the whole net for $B \,|\, b.A \,|\, C \,|\, C$ is $[\![B \,|\, b.A \,|\, C \,|\, C]\!]_\emptyset = (S, A, T, m_0)$, where

$$\begin{aligned} S &= \{B, b.A, A, C\}, \\ A &= \{a, b\}, \\ T &= \{(\{B\}, b, \{A\}), (\{A\}, a, \{b.A\}), (\{b.A\}, b, \{A\}), (\{C\}, a, \{C\})\}, \\ m_0 &= \{B, b.A, C, C\}. \end{aligned}$$

The resulting net is depicted in Figure 5.7. □

Theorem 5.8. (Operational and denotational semantics coincide) *For any CFM process p, $Net(p) = [\![p]\!]_\emptyset$.*

Proof. As for the analogous Theorem 4.8, the proof is by induction on the definition of $[\![p]\!]_I$ and of $Net(p, I)$, where the latter is the operational net associated with p, assuming that the constants in I are undefined. Then, the thesis follows for $I = \emptyset$. For all the cases, except for parallel composition, we can resort to that proof. For the case of parallel composition, by induction, we can assume that $Net(p_i, I) = (S_i, A_i, T_i, m_i) = [\![p_i]\!]_I$ for $i = 1, 2$. Now, the thesis follows by Proposition 5.8: $Net(p_1 \,|\, p_2, I) = (S_1 \cup S_2, A_1 \cup A_2, T_1 \cup T_2, m_1 \oplus m_2) = [\![p_1 \,|\, p_2]\!]_I$, as required. □

5.2 BPP: Basic Parallel Processes

The CCS [Mil89, GV15] subcalculus of *Basic Parallel Processes* (BPP for short) is generated by the following abstract syntax:

$$\begin{array}{lll} s ::= \mathbf{0} \mid \mu.p \mid s+s & & \\ q ::= s \mid \ C & & \textit{sequential processes} \\ p ::= q \mid p \,|\, p & & \textit{parallel processes} \end{array}$$

which slightly generalizes that of CFM: the only difference is that now the action prefixing operator $\mu.-$ is applied to any possibly parallel process in syntactic category p. Hence, $\mu.p$ denotes a process which *starts sequentially*, but, after performing μ, it can spawn parallel subcomponents. For instance, $a.(b.\mathbf{0} \,|\, c.\mathbf{0})$. As usual, we assume that process constants are always defined and that *Const(p)* is finite for any process p. The body of a constant is assumed to be in syntactic category s; therefore, a constant is always guarded and starts sequentially. The set of BPP processes is denoted by $\mathscr{P}_{BPP}$, and the set of its sequential processes, i.e., of the processes in syntactic category q, by $\mathscr{P}_{BPP}^{seq}$. Alternatively, the syntax can be more succinctly given as follows:

$$p ::= \Sigma_{j \in J} \mu_j.p_j \mid C \mid p \,|\, p$$

where J is finite, $\Sigma_{j \in J} \mu_j.p_j = \mathbf{0}$ when $J = \emptyset$, and the body of any constant C is of the form $\Sigma_{j \in J} \mu_j.p_j$.

The definition of the set of subterms $sub(p)$ of a BPP process p is as for CFM processes (Section 5.1). Function $sub(p)$ computes the set of the *sequential* subterms of p, i.e., for any $p \in \mathscr{P}_{BPP}$, the set $sub(p)$ of its sequential subterms is a subset of $\mathscr{P}_{BPP}^{seq}$. As for SFM and CFM, also for BPP the following facts hold.

Theorem 5.9. *For any BPP process p, the set of its sequential subterms $sub(p)$ is finite.* □

Proposition 5.13. *For any BPP processes p and q, the following hold:*

(i) *if p is sequential, then $p \in sub(p)$, and*
(ii) *if $p \in sub(q)$, then $sub(p) \subseteq sub(q)$.* □

The LTS operational rules are those of CFM; they are described in Table 4.2 and Table 5.1. Similarly, the step operational semantics is defined as for CFM by means of the rules in Table 5.2.

5.2.1 Expressiveness

BPP is strictly more expressive than CFM: since the operator of parallel composition $-|-$ may occur inside the body of recursively defined constants, a BPP process may generate an LTS with infinitely many states, as illustrated by the example below [AILS07, San12, GV15].

Example 5.9. (**Semi-counter**) A semi-counter, i.e., a counter that cannot test for zero,[3] can be represented by means of an unbounded number of constants $SCount_i$

[3] A real counter that can test for zero is defined in NPL in Section 8.2.1 and in CCS in Section 9.3.

for $i \in \mathbb{N}$, as follows:

$$\begin{aligned} SCount_0 &\doteq inc.SCount_1, \\ SCount_n &\doteq inc.SCount_{n+1} + dec.SCount_{n-1} \quad n > 0. \end{aligned}$$

Note that this process is not an SFM process because the number of constants used is infinite. The reader can easily check that the LTS for $SCount_0$ is isomorphic to the one in Figure 2.3(b), where process $SCount_i$ is mapped to state q_i, for any $i \in \mathbb{N}$. Observe that, for any $i \in \mathbb{N}$, the longest trace composed only of occurrences of action dec that $SCount_i$ can perform is of length i; such a trace is denoted by dec^i (where $dec^0 = \varepsilon$, $dec^{i+1} = dec\, dec^i$). Therefore, $SCount_i$ cannot be trace equivalent to any $SCount_j$ for $j \neq i$ because if, say, $j > i$ then trace dec^j can be executed by $SCount_j$, but not by $SCount_i$. This means that, for any $i \in \mathbb{N}$, $SCount_i$ cannot be trace equivalent to any other constant $SCount_j$ with a different index. Since the LTS semantics of any CFM process is a finite-state LTS, we can conclude that no CFM process q can be trace equivalent to $SCount_0$.

We want to show that there exists a simple BPP process, defined by means of a single constant

$$SC \doteq inc.(SC \,|\, dec.\mathbf{0})$$

that is bisimulation equivalent to $SCount_0$, even if the LTS for SC is not boundedly branching. Processes $SCount_0$ and SC are strongly bisimilar, indeed, hence proving that BPP is strictly more expressive than CFM. Consider the relation

$$R = \{(SCount_n, SC \,|\, \Pi_{i=1}^{n} dec.\mathbf{0}) \;\big|\; n \geq 0\},$$

where $\Pi_{i=1}^{n} dec.\mathbf{0}$ denotes the parallel composition of n instances of process $dec.\mathbf{0}$. It is possible to show that this relation is a strong bisimulation up to $\sim$ (Definition 2.13). Observe that for $n = 0$, the pair in R is $(SCount_0, SC \,|\, \mathbf{0})$. If R is a bisimulation up to $\sim$, then $SCount_0 \sim SC \,|\, \mathbf{0}$. As $SC \,|\, \mathbf{0} \sim SC$, by transitivity we get our expected result: $SCount_0 \sim SC$. The formal proof that R is indeed a strong bisimulation up to $\sim$ can be found, e.g., in [GV15], Section 3.4.4; a similar proof (for a real counter) is given in Section 8.2.1. So, a semi-counter can be represented, up to $\sim$, by a simple BPP process. □

Definition 5.3. (BPP language) A language $L \subseteq \mathscr{L}^*$ is a *BPP language* if there exists a BPP process p such that the set of its weak completed traces is L, i.e., $WCTr(p) = L$. □

As BPP is a superset of SFM and CFM, the class of regular languages (which are the languages representable by SFM or CFM processes, as discussed in Remarks 4.3 and 5.2) is included in the class of BPP languages. However, the class of BPP languages includes also some non-regular languages.

Example 5.10. **(A BPP language may be not regular)** Consider the BPP process $A \doteq a.(A \,|\, b.\mathbf{0}) + c.0$, discussed in [Chr93, BCMS01]. Then, for any completed trace σ of A, it holds that $\sharp(b,\sigma) = \sharp(a,\sigma)$, where by the notation $\sharp(b,\sigma)$ we denote the number of occurrences of action b in trace σ. The set of completed traces $CTr(A)$ is not a *regular language*: if $CTr(A)$ were regular, then $L = CTr(A) \cap a^*cb^*$ should be regular, because the intersection of two regular languages is a regular language [HMU01]. But the resulting set is $L = \{a^k cb^k \mid k \geq 0\}$, which is a typical example of a non-regular language. □

Example 5.11. **(A BPP language may be not context-free)** It is also possible to show that some BPP language is not *context-free*. For instance, consider the following process, originally introduced in [BCMS01]:

$$B \doteq a.(B\,|\,b.\mathbf{0}) + c.(B\,|\,d.\mathbf{0}) + e.\mathbf{0}.$$

If $CTr(B)$ were a context-free language, then also $L = CTr(B) \cap a^*c^*b^*d^*e$ would be context-free, as the intersection of a context-free language with a regular one is a context-free language [HMU01]. However, L is the set $\{a^k c^n b^k d^n e \mid k,n \geq 0\}$, which is a well-known example of a context-dependent language. □

To complete the picture, there exist context-free languages not definable as the set of completed traces of any BPP process. For instance, in [Chr93, BCMS01] it is proved that the context-free language $L = \{a^n cb^n \mid n \geq 0\}$ is not a BPP language (see also Example 6.4). However, all BPP languages are context-dependent, at least if no τ-labeled transitions are present [Pet81].

The problem of checking bisimulation equivalence over BPP processes is decidable [CHM93], more precisely PSPACE-complete [Jan03]. Weak bisimilarity has been proved decidable in some restricted cases, e.g., when one of the two processes is finite-state [JKM01, KM02], but the problem in the general case is still open. On the contrary, trace equivalence over BPP is undecidable [Hir93].

5.2.2 *Operational Net Semantics*

The operational net semantics for BPP is described as for CFM. The decomposition function *dec* is defined in Tables 4.3 and 5.3. According to this definition, function *dec* maps a BPP process p into a finite multiset of sequential BPP processes, i.e., $S_{BPP} = \mathscr{P}^{seq}_{BPP} \setminus \{\mathbf{0}\}$ and $dec : \mathscr{P}_{BPP} \to \mathscr{M}_{fin}(S_{BPP})$.

Example 5.12. Consider the semi-counter $SC \doteq inc.(SC\,|\,dec.\mathbf{0})$, discussed in Example 5.9.[4] Then, the decomposition $dec(SC)$ is $\{SC\}$, $dec(inc.(SC\,|\,dec.\mathbf{0})) = \{inc.(SC\,|\,dec.\mathbf{0})\}$ and $dec(SC\,|\,dec.\mathbf{0}) = \{SC, dec.\mathbf{0}\}$. □

The net transition rules are those in Table 4.4. Note only that the axiom (pref)

$$\frac{}{\{\mu.p\} \xrightarrow{\mu} dec(p)}$$

[4] Please, do not confuse action *dec* (for decrement) of the semi-counter with the decomposition function $dec(-)$.

now allows for the creation of transitions with non-singleton post-set, as p can be a parallel process and so $dec(p)$ can be a non-singleton multiset. The set of BPP net transitions is denoted by T_{BPP}. The net for BPP is $N_{BPP} = (S_{BPP}, Act, T_{BPP})$. We now observe that any BPP net transition is such that its pre-set is a singleton.

Proposition 5.14. *For any $t \in T_{BPP}$, we have that $|{}^\bullet t| = 1$*

Proof. *By induction on the proof of t, according to the rules in Table 4.4.* □

Lemma 5.2. *For any BPP process p, $\bigcup_{s \in dom(dec(p))} sub(s) \subseteq sub(p)$.*

Proof. *By induction on the structure of p, where the base case is when p is sequential.* □

Proposition 5.15. *For any $t \in T_{BPP}$ of the form $t = (\{p\}, \mu, m)$, we have that for any $s \in dom(m)$, $s \in sub(p)$.*

Proof. *By induction on the proof of t. By axiom (pref), $p = \mu.p'$ and transition $(\{\mu.p'\}, \mu, dec(p'))$ is derivable. If $dec(p') = \emptyset$, then the thesis holds vacuously. Otherwise, for any $s \in dom(dec(p'))$, $s \in sub(s)$ by Proposition 5.13, and so we have that $dom(dec(p')) \subseteq \bigcup_{s \in dom(dec(p'))} sub(s) \subseteq sub(p')$, the latter holding by Lemma 5.2. Hence, any $s \in dom(dec(p'))$ is such that $s \in sub(p')$. As $sub(p') \subseteq sub(\mu.p')$, the thesis follows trivially.*

The other rules are proved by assuming that the thesis holds for the premise net transition. For (cons), $p = C$, $C \doteq q$ and the premise net transition is $(\{q\}, \mu, m)$; we can assume, by induction, that for any $s \in dom(m)$, $s \in sub(q)$. The conclusion of the rule is the transition $(\{C\}, \mu, m)$; the thesis holds because $sub(q) \subseteq sub(C)$. The cases for rule (sum$_1$) and (sum$_2$) are similar, hence omitted. □

Definition 5.4. Let p be a BPP process. The P/T system statically associated with p is $Net(p) = (S_p, A_p, T_p, m_0)$, where $m_0 = dec(p)$ and

$$\begin{aligned} S_p &= [\![dom(m_0)\rangle \quad \text{computed in } N_{BPP}, \\ T_p &= \{t \in T_{BPP} \mid S_p[\![t\rangle\}, \\ A_p &= \{\mu \in Act \mid \exists t \in T_p, \mu = l(t))\}. \end{aligned}$$ □

The following propositions present three facts that are obviously true by construction of the net $Net(p)$ associated with a BPP process p.

Proposition 5.16. *For any $p \in \mathscr{P}_{BPP}$, $Net(p)$ is a statically reduced P/T net.*[5] □

Proposition 5.17. *If $dec(p) = dec(q)$, then $Net(p) = Net(q)$.* □

Proposition 5.18. *For any $p_1, p_2 \in \mathscr{P}_{BPP}$, if $Net(p_i) = (S_i, A_i, t_i, m_i)$ for $i = 1, 2$, then $Net(p_1 \mid p_2) = (S_1 \cup S_2, A_1 \cup A_2, T_1 \cup T_2, m_1 \oplus m_2)$.* □

[5] As for BPP nets, the notion of statically reachable subnet and dynamically reachable subnet coincide, $Net(p)$ is also dynamically reduced.

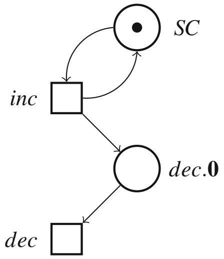


Fig. 5.8 The BPP net for the semi-counter *SC*

Example 5.13. Consider again the semi-counter $SC \doteq inc.(SC \,|\, dec.\mathbf{0})$. Then, by axiom (pref), the net transition $\{inc.(SC \,|\, dec.\mathbf{0})\} \xrightarrow{inc} \{SC, dec.\mathbf{0}\}$ is derivable. By using this net transition as the premise for rule (cons), $\{SC\} \xrightarrow{inc} \{SC, dec.\mathbf{0}\}$ is also derivable. Finally, the net transition $\{dec.\mathbf{0}\} \xrightarrow{dec} \emptyset$ is also derivable by axiom (pref). The net for the semi-counter *SC* is outlined in Figure 5.8. Note that this net is unbounded, because there is no upper limit to the number of tokens that can be accumulated in place $dec.\mathbf{0}$. □

Theorem 5.10. (Finite number of places) *For any BPP process p, the set S_p of its places statically reachable from $dom(dec(p))$ is finite.*

Proof. *By induction on the static reachability relation $\Longrightarrow^*$ (Definition 3.9), we can prove that any place s that is statically reachable from $dom(dec(p))$ is a subterm of p. Since $sub(p)$ is finite by Theorem 5.9, the thesis follows trivially.*

The base case is $dom(dec(p)) \Longrightarrow^* dom(dec(p))$. As for any $s \in dom(dec(p))$, $s \in sub(s)$ by Proposition 5.13, we have that $dom(dec(p)) \subseteq \bigcup_{s \in dom(dec(p))} sub(s) \subseteq sub(p)$, the latter holding by Lemma 5.2.

Now, assume that $dom(dec(p)) \Longrightarrow^* S_1 \overset{t}{\Longrightarrow} S_2$ for some transition $t \in T_{BPP}$; we want to prove that any place in S_2 is a subterm of p. By induction, we can assume that $S_1 \subseteq sub(p)$. Since $t \in T_{BPP}$, by Proposition 5.14, transition t has the form $(\{s_1\}, \mu, m)$, so that $s_1 \in S_1$ and $S_2 = S_1 \oplus dom(m)$. By Proposition 5.15, any $s \in dom(m)$ is such that $s \in sub(s_1)$. Since $s_1 \in sub(p)$, by Proposition 5.13 also $sub(s_1) \subseteq sub(p)$. Therefore, any $s \in dom(m)$ is such that $s \in sub(p)$ by transitivity, as required.

Hence, the set S_p of its reachable places is finite. □

Theorem 5.11. (Finite P/T Petri net) *For any BPP process p, the P/T net reachable from p, $Net(p) = (S_p, A_p, T_p, dec(p))$, is finite.*

Proof. *By Theorem 5.10, the set S_p of the places reachable from p is finite. By Proposition 5.14 the pre-set of any derivable transition is a singleton. By Lemma 4.2, the set T^q of transitions with pre-set $\{q\}$ is finite, for any $q \in S_p$. Hence, also $T_p = \bigcup_{q \in S_p} T^q$ must be finite, being a finite union of finite sets.* □

Corollary 5.3. (BPP nets) *For any BPP process p, $Net(p)$ is a BPP net.*

Proof. By Theorem 5.11, $Net(p)$ is a finite P/T net. By Proposition 5.14, for any $t \in T_p$, we have $|{}^\bullet t| = 1$, i.e., $Net(p)$ is a BPP net. □

Hence, *only* BPP nets can be represented by BPP processes.

As a final remark, we observe that the operational net semantics is sound w.r.t. the step semantics: more precisely, any BPP process p, in the STS semantics, is step bisimilar to $dec(p)$, in the step marking graph $SMG(Net(p))$. The proof of this fact is analogous to the corresponding proof outlined in Section 5.1.5 for CFM, and so it is omitted.

Theorem 5.12. *For any $p \in \mathscr{P}_{BPP}$, $p \sim_{step} dec(p)$.* □

5.2.3 Representing All BPP Nets

It is not a surprise that *all* BPP nets can be represented by BPP processes. The construction described in Section 5.1.4 for concurrent FSMs can easily be extended to BPP nets: the body of any involved constant is a sequential process that, after the first action, may become a parallel process.

Definition 5.5. (Translating BPP nets into BPP terms) Let $N(m_0) = (S, A, T, m_0)$ — with $S = \{s_1, \ldots, s_n\}$, $A \subseteq Act$, $T = \{t_1, \ldots, t_k\}$, and $l(t_j) = \mu_j$ — be a BPP net. Function $\mathscr{T}_{BPP}(-)$, from BPP nets to BPP processes, is defined as

$$\mathscr{T}_{BPP}(N(m_0)) = \underbrace{C_1 | \cdots | C_1}_{m_0(s_1)} | \cdots | \underbrace{C_n | \cdots | C_n}_{m_0(s_n)}$$

where each C_i is equipped with a defining equation $C_i \doteq c_i^1 + \cdots + c_i^k$ (with $C_i \doteq \mathbf{0}$ if $k = 0$), and each summand c_i^j, for $j = 1, \ldots, k$, is equal to

- $\mathbf{0}$, if $s_i \not\in {}^\bullet t_j$;
- $\mu_j.\Pi_j$, if ${}^\bullet t_j = \{s_i\}$, where process Π_j is $\underbrace{C_1 | \cdots | C_1}_{t_j^\bullet(s_1)} | \cdots | \underbrace{C_n | \cdots | C_n}_{t_j^\bullet(s_n)}$, meaning that $\Pi_j = \mathbf{0}$ if $t_j^\bullet = \emptyset$. □

Note that $\mathscr{T}_{BPP}(N(m_0))$ is a BPP process: in fact, there are finitely many constants involved (one for each place) and each constant has a body that starts sequentially. Therefore, the following proposition holds by Proposition 5.16 and Corollary 5.3.

Proposition 5.19. *Given a BPP net $N(m_0) = (S, A, T, m_0)$, $Net(\mathscr{T}_{BPP}(N(m_0)))$ is a (statically) reduced BPP net.* □

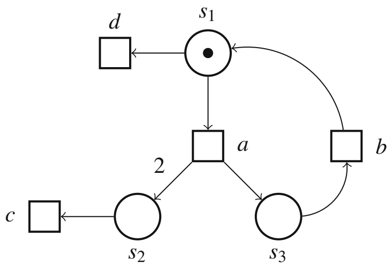


Fig. 5.9 A BPP net

Example 5.14. Let us consider the net $N(s_1)$ in Figure 5.9. The BPP process p associated with such a BPP net is C_1, where the three constants C_i, corresponding to the three places s_i, for $i = 1,2,3$, are defined as

$$\begin{aligned}
C_1 &\doteq a.(C_2\,|\,C_2\,|\,C_3) + \mathbf{0} + \mathbf{0} + d.\mathbf{0},\\
C_2 &\doteq \mathbf{0} + \mathbf{0} + c.\mathbf{0} + \mathbf{0},\\
C_3 &\doteq \mathbf{0} + b.C_1 + \mathbf{0} + \mathbf{0}.
\end{aligned}$$

It is obvious that $Net(p)$ is a net isomorphic to the original net in Figure 5.9, where the isomorphism f is the bijection $f(s_i) = C_i$ for $i = 1,2,3$. Note that this net is unbounded, as there is no limit on the number of tokens that can be accumulated in place s_2. Note also that the intersection of the set of its completed traces, $CTr(N(s_1))$, with the regular language $(ab)^*dc^*$ yields the context-free language $\{(ab)^n dc^{2n} \mid n \geq 0\}$, hence $CTr(N(s_1))$ is not a regular language. □

Theorem 5.13. (Representability theorem 3) *Let $N(m_0) = (S,A,T,m_0)$ be a (statically) reduced BPP net such that $A \subseteq Act$, and let $p = \mathscr{T}_{BPP}(N(m_0))$. Then, $Net(p)$ is isomorphic to $N(m_0)$.*

Proof. The proof follows the same pattern as the analogous Theorem 4.7. Let $N(m_0) = (S,A,T,m_0)$ be a reduced, BPP net, with $S = \{s_1,\ldots,s_n\}$, $A \subseteq Act$, $T = \{t_1,\ldots,t_k\}$ and $l(t_j) = \mu_j$ for $j = 1,\ldots,k$. The associated BPP process is

$$\mathscr{T}_{BPP}(N(m_0)) = \underbrace{C_1|\cdots|C_1}_{m_0(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m_0(s_n)},$$

where for each place s_i we have a corresponding constant $C_i \doteq \sum_{j=1}^{k} c_i^j$, defined as in Definition 5.5. For notational convenience, $\sum_{j=1}^{k} c_i^j$ is denoted by p_i, i.e., $C_i \doteq p_i$.

Let $Net(p)$ be the tuple (S',A',T',m_0'). Then, $m_0' = dec(p)$ is the marking $m_0(s_1)\cdot C_1 \oplus m_0(s_2)\cdot C_2 \oplus \ldots \oplus m_0(s_n)\cdot C_n$.

Note that, by Definition 5.5, any transition $t' \in T'$ with ${}^\bullet t' = \{C_i\} \subseteq dec(p)$ is such that $t'^\bullet = k_1\cdot C_1 \oplus k_2\cdot C_2 \oplus \ldots \oplus k_n\cdot C_n$ for suitable $k_i \geq 0$, $i = 1,\ldots,n$; by iterating this observation, each transition in T' has a singleton pre-set $\{C_i\}$ for some suitable C_i, and a post-set of the form $k_1\cdot C_1 \oplus k_2\cdot C_2 \oplus \ldots \oplus k_n\cdot C_n$, for suitable $k_i \geq 0$, $i = 1,\ldots,n$. As, by Proposition 5.19, $Net(p)$ is reduced, it follows that the places *in S' are exactly all the constants C_i for $i = 1, \ldots, n$. Note also that since $N(m_0)$ is reduced, all the places in S are reachable from the initial marking m_0. Hence, there is a bijection $f : S \to S'$ defined by $f(s_i) = C_i$, which is the natural candidate isomorphism function. To prove that f is an isomorphism, we have to prove that*

1. *$f(m_0) = m'_0$,*
2. *$t = (m, \mu, m') \in T$ implies $f(t) = (f(m), \mu, f(m')) \in T'$, and*
3. *$t' = (m'_1, \mu, m'_2) \in T'$ implies there exists $t = (m_1, \mu, m_2) \in T$ such that $f(t) = t'$, i.e., $f(m_1) = m'_1$ and $f(m_2) = m'_2$.*

From items 2) and 3) above, it follows that $A = A'$.

Proof of 1*: The mapping via f of the initial marking m_0 is*
$f(m_0) = f(m_0(s_1) \cdot s_1 \oplus m_0(s_2) \cdot s_2 \oplus \ldots \oplus m_0(s_n) \cdot s_n) =$
$= m_0(s_1) \cdot C_1 \oplus m_0(s_2) \cdot C_2 \oplus \ldots \oplus m_0(s_n) \cdot C_n = m'_0$*, as required.*

Proof of 2*: We prove that, for $j = 1, \ldots, k$, if $t_j = (m, \mu_j, m') \in T$, then $t'_j = (f(m), \mu_j, f(m')) \in T'$. By Proposition 5.14, t_j must be of the form $(\{s_i\}, \mu_j, m')$ for some suitable index i and multiset m'. Hence, $f({}^\bullet t_j) = \{C_i\}$. According to Definition 5.5, for $C_i = p_i$, we have in p_i a summand $c_i^j = \mu_j.\Pi_j$, with $\Pi_j = \underbrace{C_1|\cdots|C_1}_{m'(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m'(s_n)}$. Therefore, not only $\{c_i^j\} \xrightarrow{\mu_j} dec(\Pi_j)$ is derivable by axiom (pref), but also (by rules (sum$_1$) and (sum$_2$)) the transition $\{p_i\} \xrightarrow{\mu_j} dec(\Pi_j)$, and so $\{C_i\} \xrightarrow{\mu_j} dec(\Pi_j)$, by rule (cons). In conclusion, starting from transition $t_j = (\{s_i\}, \mu_j, m')$, we have shown that transition $t'_j = (\{C_i\}, \mu_j, dec(\Pi_j))$, with $f(m') = dec(\Pi_j)$, belongs to T', as required.*

Proof of 3*: We prove that if $t'_j = (m'_1, \mu_j, m'_2) \in T'$, then a transition $t_j = (m_1, \mu_j, m_2) \in T$ exists such that $f(m_1) = m'_1$ and $f(m_2) = m'_2$. We already observed that any t'_j (statically) reachable from the initial marking $dec(p)$ is such that ${}^\bullet t'_j = \{C_i\}$, for some index i, and $t'^\bullet_j = dec(\Pi)$, with $\Pi = \underbrace{C_1|\cdots|C_1}_{k_1}|\cdots|\underbrace{C_n|\cdots|C_n}_{k_n}$, $k_i \geq 0$ for $i = 1, \ldots, n$. Hence, $t'_j = (\{C_i\}, \mu_j, dec(\Pi))$. Such a transition is derivable, by rule (cons), only if also $\{p_i\} \xrightarrow{\mu_j} dec(\Pi)$, and so, by rules (sum$_1$) and (sum$_2$), only if $\{c_i^j\} \xrightarrow{\mu_j} dec(\Pi)$. Summand c_i^j must be of the form $\mu_j.\Pi$, and this is possible only if transition t_j is of the form $(\{s_i\}, \mu_j, m_2)$, with $m_2(s_i) = k_i$, for $i = 1, \ldots, n$, as required.* □

5.2.4 Denotational Net Semantics

The denotational net semantics for BPP is defined exactly as for CFM. The definitions of the SFM operators are described in Table 4.5, while the denotational definition for the parallel operator is given in Table 5.4.

Example 5.15. Consider again the semi-counter $SC \doteq inc.(SC \mid dec.\mathbf{0})$. We have that

$[\![SC]\!]_{\{SC\}} = (\{SC\}, \emptyset, \emptyset, \{SC\})$, and
$[\![dec.\mathbf{0}]\!]_{\{SC\}} = (\{dec.\mathbf{0}\}, \{dec\}, \{(\{dec.\mathbf{0}\}, dec, \emptyset)\}, \{dec.\mathbf{0}\})$.
Therefore, the net $[\![SC \,|\, dec.\mathbf{0}]\!]_{\{SC\}}$ is
$(\{SC, dec.\mathbf{0}\}, \{dec\}, \{(\{dec.\mathbf{0}\}, dec, \emptyset)\}, \{SC, dec.\mathbf{0}\})$.
The net $[\![inc.(SC \,|\, dec.\mathbf{0})]\!]_{\{SC\}}$ is
$(\{inc.(SC \,|\, dec.\mathbf{0}), SC, dec.\mathbf{0}\}, \{inc, dec\}, \{(\{inc.(SC \,|\, dec.\mathbf{0})\}, inc, \{SC, dec.\mathbf{0}\}),$
$(\{dec.\mathbf{0}\}, dec, \emptyset)\}, \{inc.(SC \,|\, dec.\mathbf{0})\})$.
Finally, the net $[\![SC]\!]_{\emptyset}$ is
$(\{SC, dec.\mathbf{0}\}, \{inc, dec\}, \{(\{SC\}, inc, \{SC, dec.\mathbf{0}\}), (\{dec.\mathbf{0}\}, dec, \emptyset)\}, \{SC\})$,
which is exactly the net in Figure 5.8. □

Theorem 5.14. (Operational and denotational semantics coincide) *For any BPP process p,* $Net(p) = [\![p]\!]_{\emptyset}$.

Proof. Similar to the proof of Theorems 4.8 and 5.8, hence omitted. □

Chapter 6
Adding Communication and Restriction: FNC

Abstract The parallel composition operator is enhanced to allow for synchronous, point-to-point (or handshake) communication; moreover, in order to force communication, also the CCS operator of restriction is added to BPP, but at the top level only. The resulting calculus is a subset of CCS, called *Finite-Net* CCS (FNC, for short), which is expressive enough to model the class of finite CCS nets.

6.1 Syntax

To model communication between two processes, we need to give structure to the set of observable actions, distinguishing between *input* actions and *output* actions.

Let $\mathscr{L}$ be a finite set of *names*, ranged over by $a, b, c, \ldots$, also called the *input actions*. Let $\overline{\mathscr{L}}$ be the set of *co-names*, ranged over by $\overline{a}, \overline{b}, \overline{c}, \ldots$, also called the *output actions*. The set $\mathscr{L} \cup \overline{\mathscr{L}}$, ranged over by $\alpha, \beta, \ldots$, is the set of *observable actions*. By $\overline{\alpha}$ we mean the complement of α, assuming that $\overline{\overline{\alpha}} = \alpha$. Let $Act = \mathscr{L} \cup \overline{\mathscr{L}} \cup \{\tau\}$, such that $\tau \notin \mathscr{L} \cup \overline{\mathscr{L}}$, be the finite set of *actions*, ranged over by μ. Action τ denotes an invisible, internal activity. Let $\mathscr{C}ons$ be a countable set of process *constants*, disjoint from Act, ranged over by $A, B, C, \ldots$, possibly indexed.

Finite-Net CCS (FNC for short) is the calculus whose terms are generated from actions and constants as described by the following abstract syntax:

$$
\begin{array}{llll}
s ::= \mathbf{0} \mid & \mu.t & \mid s+s & \\
q ::= s \mid & C & & \textit{sequential terms} \\
t ::= q \mid & t \,|\, t & & \textit{restriction-free terms} \\
p ::= t \mid & (\nu a)p & & \textit{general terms}
\end{array}
$$

where C is defined over the syntactic category s, as usual. The operator of parallel composition, $p_1 \,|\, p_2$, now allows for handshake communication between p_1 and p_2 on complementary input/output actions; the label of the resulting synchronization is the invisible action τ; hence, synchronization is strictly binary in FNC. The restric-

R. Gorrieri, *Process Algebras for Petri Nets*, Monographs in Theoretical
Computer Science. An EATCS Series, DOI 10.1007/978-3-319-55559-1_6

$Const(\mathbf{0}) = \emptyset$	$Const(p_1 + p_2) = Const(p_1) \cup Const(p_2)$
$Const(\mu.p) = Const(p)$	$Const(p_1 \mid p_2) = Const(p_1) \cup Const(p_2)$
$Const((\nu a)p) = Const(p)$	$Const(C) = \begin{cases} \{C\} & \text{if } C \text{ undef.} \\ \{C\} \cup Const(p) & \text{if } C \doteq p \end{cases}$

Table 6.1 Set of constants used by a process

tion operator $(\nu a)-$, parametrized on an action $a \in \mathscr{L}$, makes the name a private for a term; intuitively, $(\nu a)p$ can do whatever transition p can do, provided that the transition is not labeled with a or $\overline{a}$. Of course, if $p = p_1 \mid p_2$, then $(\nu a)p$ allows for the handshake synchronization on complementary actions $a/\overline{a}$ performed by p_1 and p_2, because the resulting synchronization transition is labeled τ, hence different from a or $\overline{a}$. In other words, the effect of applying the restriction on a over a parallel term $p_1 \mid p_2$ is to impede p_1 and p_2 from executing asynchronously action a or $\overline{a}$, hence forcing them to synchronize on such actions.

A term of the form $(\nu a_1)(\nu a_2)\dots(\nu a_n)p$ is usually denoted in a compact form by $(\nu L)p$, where $L = \{a_1, \dots, a_n\} \subseteq \mathscr{L}$. As for BPP, in FNC parallel composition may occur inside the body of a recursively defined constant C; on the contrary, the restriction operator $(\nu a)-$ is not allowed in the body of C. So, an FNC general term may be represented as $(\nu L)t$, where L is a set of actions (if L is empty, the restriction operator is not present) and t is a restriction-free term.

An FNC term p is an FNC *process* if it is fully defined and the set of constants $Const(p)$ used by p is finite, where by $Const(p)$ we denote the least set of process constants such that the equations in Table 6.1 are satisfied. $\mathscr{P}_{FNC}$ denotes the set of FNC processes, while $\mathscr{P}_{FNC}^{seq}$ denotes the set of its sequential processes, i.e., the processes in syntactic category q. Alternatively, the FNC syntax can be more succinctly given as follows:

$$\begin{aligned} s &::= \Sigma_{j \in J} \mu_j.t_j \mid C \\ t &::= s \mid t|t \\ p &::= t \mid (\nu a)p \end{aligned}$$

where J is finite, $\Sigma_{j \in J} \mu_j.t_j = \mathbf{0}$ when $J = \emptyset$, and the body of any constant C is of the form $\Sigma_{j \in J} \mu_j.t_j$.

Definition 6.1. (Bound names, free names and free outputs) Given a process $p = (\nu L)t$, where t is a restriction-free process, the set of its *bound names* is L, denoted $bn(p)$. The *free names* of p, denoted $fn(p)$, and the *free outputs* of p, denoted $fo(p)$, are defined as the sets $F(p, \emptyset)$ and $G(p, \emptyset)$, respectively, where $F(p, I)$ and $G(p, I)$, with I a set of process constants, are defined in Table 6.2. □

Proposition 6.1. *For any FNC process p, the sets $fn(p)$, $bn(p)$ and $fo(p)$ are finite.*

Proof. Trivial, since $\mathscr{L}$ is finite. □

$$
\begin{array}{rclrcl}
F(\mathbf{0},I) & = & \emptyset & G(\mathbf{0},I) & = & \emptyset \\
F(a.p,I) & = & F(p,I) \cup \{a\} & G(a.p,I) & = & G(p,I) \\
F(\overline{a}.p,I) & = & F(p,I) \cup \{a\} & G(\overline{a}.p,I) & = & G(p,I) \cup \{\overline{a}\} \\
F(\tau.p,I) & = & F(p,I) & G(\tau.p,I) & = & G(p,I) \\
F(p+q,I) & = & F(p,I) \cup F(q,I) & G(p+q,I) & = & G(p,I) \cup G(q,I) \\
F(p\,|\,q,I) & = & F(p,I) \cup F(q,I) & G(p\,|\,q,I) & = & G(p,I) \cup G(q,I) \\
F((\nu a)p,I) & = & F(p,I) \setminus \{a\} & G((\nu a)p,I) & = & G(p,I) \setminus \{\overline{a}\}
\end{array}
$$

$$
F(C,I) = \begin{cases} F(q, I \cup \{C\}) & \text{if } C \doteq q \text{ and } C \notin I \\ \emptyset & \text{if } C \in I \end{cases}
$$

$$
G(C,I) = \begin{cases} G(q, I \cup \{C\}) & \text{if } C \doteq q \text{ and } C \notin I \\ \emptyset & \text{if } C \in I \end{cases}
$$

Table 6.2 Free names and free outputs

Definition 6.2. (Closed processes and output-closed processes) A process p is *closed* if the set $fn(p)$ of its free names is empty. Process p is *output-closed* if the set $fo(p)$ of its free outputs is empty. □

6.1.1 Restricted Actions and Extended Processes

The FNC processes are built upon the set $\mathscr{L} \cup \overline{\mathscr{L}}$, ranged over by α, of visible actions. We assume we also have sets $\mathscr{L}' = \{a' \,|\, a \in \mathscr{L}\}$ and $\overline{\mathscr{L}'} = \{\overline{a'} \mid \overline{a} \in \overline{\mathscr{L}}\}$, where $\mathscr{L}' \cup \overline{\mathscr{L}'}$, ranged over by α', is the set of auxiliary *restricted* actions, i.e., actions that are only allowed to synchronize. By definition, each restricted action α' corresponds to exactly one visible action α.

$\mathscr{G} = \mathscr{L} \cup \mathscr{L}'$, ranged over by γ, is the set of input actions and their restricted counterparts. $\overline{\mathscr{G}} = \overline{\mathscr{L}} \cup \overline{\mathscr{L}'}$ is the set of output actions and their restricted counterparts. The set of all actions $Act_\gamma = \mathscr{G} \cup \overline{\mathscr{G}} \cup \{\tau\}$, ranged over by μ (with abuse of notation), is used to build the set of *extended terms*, whose syntax is as follows:

$$
\begin{array}{lcl}
s ::= \mathbf{0} \mid \ \mu.t \ \mid s+s & & \mu \in Act_\gamma \\
q ::= s \mid \ \ C & & \textit{sequential extended terms} \\
t \ ::= q \mid \ \ t\,|\,t & & \textit{restriction-free extended terms} \\
p ::= t \ \mid (\nu a)p & & \textit{general extended terms}
\end{array}
$$

where the prefixes are taken from the set Act_γ and the bound action a is in $\mathscr{L}$. An extended FNC term $p = (\nu L)t$ is an *extended process* if it is fully defined, $Const(p)$ is finite and t is *admissible*, i.e.,

$$\forall a \in \mathscr{L},\ \{a, a'\} \not\subseteq fn(t),$$

where the function $fn(-)$ is defined on extended terms in the obvious way. The admissibility condition expresses a sort of sanity check on any restriction-free, extended term t: it is not possible that, for any action $a \in \mathscr{L}$, there are occurrences in t of both a and its associated restricted action a'; this because each action type can occur in t only in one of the two modalities: either restricted or *unrestricted* (i.e., normally visible). For instance, $a.a'.\mathbf{0}$ is not admissible, while $a.\mathbf{0} \,|\, b'.\mathbf{0}$ is admissible. By the notation $ad(t)$ we mean that t is admissible.

By $\mathscr{P}_{FNC}^{\gamma}$ we denote the set of all extended FNC processes. By $\mathscr{P}_{FNC}^{\gamma,par}$ we denote the set of all restriction-free, extended FNC processes, i.e., those extended processes in syntactic category t. By $\mathscr{P}_{FNC}^{\gamma,seq}$ we denote the set of all sequential, extended FNC processes, i.e., those extended processes in syntactic category q.

6.1.2 Syntactic Substitution

In order to define the net semantics for FNC, we need to introduce the concept of syntactic substitution. Intuitively, by the notation $p\{a'/a\}$ we denote the process p where each occurrence of the free name a has been replaced by the corresponding restricted action a'. We will see that any FNC process can be mapped to a restriction-free, extended FNC process via syntactic substitution.

Definition 6.3. (Substitutions) A substitution is a set $\{a_i'/a_i\}_{i \in I}$ of associations of the form a_i'/a_i for $i \in I$, meaning that action $a_i \in \mathscr{L}$ is to be replaced by the corresponding restricted action $a_i' \in \mathscr{L}'$, when applied to some term. If $L = \{a_1, a_2, \ldots, a_n\}$, then the substitution $\{a_1'/a_1, \ldots, a_n'/a_n\}$ is shortened as $\{L'/L\}$. We use θ to range over the set of substitutions.

A substitution $\{a_i'/a_i\}_{i \in I}$ is *empty*, denoted by ε, if $|I| = 0$, i.e., there is no association in the set. A substitution $\{a_i'/a_i\}_{i \in I}$ is *unary* if $|I| = 1$, i.e., it is of the form $\{a'/a\}$. The composition of a substitution $\theta = \{a_1'/a_1, \ldots, a_n'/a_n\}$ with a unary substitution $\{b'/b\}$, denoted by $\theta \circ \{b'/b\}$, is defined as follows:

- $\{a_1'/a_1, \ldots, a_n'/a_n\}$ if there exists an index j such that $b = a_j$;
- $\{a_1'/a_1, \ldots, a_n'/a_n, b'/b\}$ if $b \neq a_j$ for all $j = 1, \ldots, n$.

Examples of substitution composition are the following: $\varepsilon \circ \{b'/b\} = \{b'/b\}$, $\{a'/a\} \circ \{a'/a\} = \{a'/a\}$ and $\{a'/a, b'/b\} \circ \{c'/c\} = \{a'/a, b'/b, c'/c\}$.

A nonemtpy substitution $\theta = \{a_1'/a_1, \ldots, a_n'/a_n\}$ can be represented as the composition of the unary substitution $\{a_1'/a_1\}$ and $\theta' = \{a_2'/a_2, \ldots, a_n'/a_n\}$, i.e., $\theta = \{a_1'/a_1\} \circ \theta'$. □

Remark 6.1. **(Parametrized constants)** In the following definition of syntactic substitution, we are applying a unary substitution $\{a'/a\}$ also to any constant C, resulting in the new constant $C_{\{a'/a\}}$. In order to get a completely satisfactory definition, we have to assume that constants are parametrized by substitutions, with the proviso that a normal constant C is simply parametrized by the empty substitution ε, i.e., C_ε. Then, by applying a unary substitution $\{a'/a\}$ to C_θ, a new constant is generated with index $\theta \circ \{a'/a\}$, namely $C_{\theta \circ \{a'/a\}}$, whose definition is $C_{\theta \circ \{a'/a\}} \doteq q\{a'/a\}$ if $C_\theta \doteq q$. □

$$
\begin{aligned}
\mathbf{0}\{a'/a\} &= \mathbf{0} \\
(a.p)\{a'/a\} &= a'.(p\{a'/a\}) \\
(\overline{a}.p)\{a'/a\} &= \overline{a'}.(p\{a'/a\}) \\
(\mu.p)\{a'/a\} &= \mu.(p\{a'/a\}) \qquad \text{if } \mu \neq a, \overline{a}, \mu \in Act_\gamma \\
(p+q)\{a'/a\} &= p\{a'/a\} + q\{a'/a\} \\
(p \,|\, q)\{a'/a\} &= p\{a'/a\} \,|\, q\{a'/a\} \\
((\nu b)p)\{a'/a\} &= (\nu b)(p\{a'/a\}) \qquad \text{if } b \neq a \\
((\nu a)p)\{a'/a\} &= (\nu a)p \\
C_\theta\{a'/a\} &= \begin{cases} C_\theta & \text{if } a \notin fn(C_\theta) \\ C_{\theta\circ\{a'/a\}} & \text{otherwise, with } C_{\theta\circ\{a'/a\}} \doteq q\{a'/a\} \text{ if } C_\theta \doteq q \end{cases}
\end{aligned}
$$

Table 6.3 Syntactic substitution of a restricted name a' for its corresponding action a

Definition 6.4. (Syntactic substitution) The syntactic substitution $p\{a'/a\}$ of the restricted action a' for the visible action a inside an extended FNC process p is defined in Table 6.3. The application of a substitution $\theta = \{a'/a\} \circ \theta'$ to a process p can be computed as follows: $p\theta = (p\{a'/a\})\theta'$, with the proviso that $p\varepsilon = p$. □

Example 6.1. Consider the FNC process $p = (\nu a)((\nu b)((a.b.\mathbf{0} \,|\, \overline{a}.\mathbf{0}) \,|\, c.\mathbf{0}))$ and the substitution $\theta = \{a'/a, c'/c\}$. The result of the application of θ to p is the extended process $(\nu a)((\nu b)((a.b.\mathbf{0} \,|\, \overline{a}.\mathbf{0}) \,|\, c'.\mathbf{0}))$.

Consider now the FNC process constant $C \doteq a.(b.\mathbf{0} \,|\, c.C)$ and the substitution $\theta = \{a'/a, c'/c\}$. The result of the application of θ to C is the extended process constant $C_\theta \doteq a'.(b.\mathbf{0} \,|\, c'.C_\theta)$. □

Proposition 6.2. *For any extended FNC process p and for any $a \in \mathscr{L}$, $p\{a'/a\}$ is an extended FNC process.*

Proof. The proof is by case analysis, following the definition of syntactic substitution. As Const(p) is finite, the recursive application of the syntactic substitution to the body of each constant will eventually terminate. □

Remark 6.2. **(Inverse substitution)** By Definition 6.3, in a substitution $\{a'_i/a_i\}_{i \in I}$ of associations of the form a'_i/a_i for $i \in I$, the visible action $a_i \in \mathscr{L}$ is to be replaced by the corresponding restricted action $a'_i \in \mathscr{L}'$. This is the kind of substitution we mainly use in this chapter and in the following ones. However, it is sometimes convenient to define the *inverse* substitution $\{a_i/a'_i\}_{i \in I}$, where the restricted action $a'_i \in \mathscr{L}'$ is to be replaced by the corresponding action $a_i \in \mathscr{L}$. The definition of syntactic substitution can easily be adapted to inverse substitutions. If an extended, restriction-free process t is such that $L' = fn(t) \cap \mathscr{L}'$, then $t\{L/L'\}$, where $L = \{a \mid a' \in L'\}$, is an FNC process with no occurrences of restricted actions. The composition of a substitution $\{L'/L\}$ with its inverse $\{L/L'\}$ yields the empty substitution ε. So, for instance, $\{a'/a, b'/b\} \circ \{b/b'\} = \{a'/a\}$; hence, $C_{\{a'/a\}}\{a/a'\} = C_\varepsilon$. □

Proposition 6.3. *For any extended, restriction-free FNC process t such that $L' = fn(t) \cap \mathscr{L}'$, $(t\{L/L'\})\{L'/L\} = t$.*

Proof. The thesis follows by admissibility of t: as no $a \in L$ is a free name of t, with $t\{L/L'\}$ we turn all the occurrences of $a' \in L'$ into the corresponding $a \in L$, which are the only occurrences of a in $t\{L/L'\}$; by applying the substitution $\{L'/L\}$, we revert all the occurrences of a to a'. Moreover, each constant $C_{\{L/L'\}}$ in $Const(t\{L/L'\})$ is reverted to C_ε in $Const(t)$ by the substitution $\{L'/L\}$. □

Admissibility is essential for the correctness of the proposition above: the term $t = a.a'.\mathbf{0}$, which is not admissible, is such that $(t\{a/a'\})\{a'/a\} = a'.a'.\mathbf{0} \neq t$.

Definition 6.5. (Mapping processes to restriction-free, extended processes) Function $i : \mathscr{P}_{FNC} \to \mathscr{P}_{FNC}^{\gamma,par}$, defined as $i((\nu L)t) = t\{L'/L\}$, maps FNC processes to restriction-free, extended FNC processes. □

Function i is the identity over restriction-free processes: $i(t) = t$. In general, i is not injective; for instance, $i((\nu a)b.\mathbf{0}) = b.\mathbf{0} = i(b.\mathbf{0}) = i((\nu c)b.\mathbf{0})$ and $i((\nu a)a.\mathbf{0}) = a'.\mathbf{0} = i((\nu a)((\nu a)a.\mathbf{0}))$. However, it is possible to prove that i is surjective. Given any restriction-free, extended FNC process t, let $L' = fn(t) \cap \mathscr{L}'$; since t is an extended process, t is admissible, i.e., there is no $a \in \mathscr{L}$ such that $a \in fn(t)$ and $a' \in fn(t)$; hence, for any $a' \in L'$, we have $a \notin fn(t)$. Therefore, the FNC process $p = (\nu L)(t\{L/L'\})$ is such that $i(p) = i((\nu L)(t\{L/L'\})) = i(t\{L/L'\})\{L'/L\} = (t\{L/L'\})\{L'/L\} = t$, by Proposition 6.3.

We will see that the net semantics for an FNC process p is actually given to its corresponding restriction-free, extended FNC process $i(p)$, because it holds that $dec(p) = dec(i(p))$ (see Proposition 6.8).

6.1.3 Sequential Subterms

We are interested in singling out the set of sequential subterms of an FNC process p. We will prove that, for any p, the set of its sequential subterms is finite and each of its sequential subterms is a sequential, extended FNC process.

Definition 6.6. (Subterm) For any FNC process p, the set of its sequential subterms $sub(p)$ is defined by means of the auxiliary function (with the same name, with abuse of notation) $sub(p, \emptyset)$, whose second parameter is a set of already known constants, initially empty, described in Table 6.4. □

While for the BPP operators the definition is as expected, the definition of the sequential subterms of a restricted term is unusual. In fact, with the equality $sub((\nu a)p, I) = sub(p, I)\{a'/a\}$ we state that the sequential subterms of $(\nu a)p$ are the subterms of p, where each occurrence of the bound name a (or $\overline{a}$) is replaced by the corresponding restricted action a' (or $\overline{a'}$).

Example 6.2. Consider $p = (\nu a)(b.a.\mathbf{0} \,|\, c.\mathbf{0})$. The set $sub(p)$ is $sub(b.a.\mathbf{0} \,|\, c.\mathbf{0})\{a'/a\}$. Since $sub(b.a.\mathbf{0} \,|\, c.\mathbf{0}) = \{b.a.\mathbf{0}, a.\mathbf{0}, \mathbf{0}, c.\mathbf{0}\}$, we get $sub(p) = \{b.a'.\mathbf{0}, a'.\mathbf{0}, \mathbf{0}, c.\mathbf{0}\}$. □

$sub(\mathbf{0},I) = \{\mathbf{0}\}$	$sub(p_1+p_2,I) = \{p_1+p_2\} \cup sub(p_1,I) \cup sub(p_2,I)$
$sub(\mu.p,I) = \{\mu.p\} \cup sub(p,I)$	$sub(p_1 \mid p_2,I) = sub(p_1,I) \cup sub(p_2,I)$
$sub((\nu a)p,I) = sub(p,I)\{a'/a\}$	$sub(C,I) = \begin{cases} \emptyset & C \in I, \\ \{C\} \cup sub(p, I \cup \{C\}) & C \notin I \wedge C \doteq p \end{cases}$

Table 6.4 Sequential subterms of a process

Example 6.3. Consider the process $q = (\nu a)(C \,|\, \overline{a}.\mathbf{0})$, where $C \doteq a.C + b.\mathbf{0}$. The set $sub(q)$ is computed as follows. First, note that

$$\begin{aligned} sub(C,\emptyset) &= \{C\} \cup sub(a.C+b.\mathbf{0}, \{C\}) \\ &= \{C\} \cup \{a.C+b.\mathbf{0}\} \cup sub(a.C, \{C\}) \cup sub(b.\mathbf{0}, \{C\}) \\ &= \{C, a.C+b.\mathbf{0}\} \cup \{a.C\} \cup sub(C, \{C\}) \cup \{b.\mathbf{0}\} \cup sub(\mathbf{0}, \{C\}) \\ &= \{C, a.C+b.\mathbf{0}, a.C, b.\mathbf{0}, \mathbf{0}\}. \end{aligned}$$

Then, $sub(\overline{a}.\mathbf{0},\emptyset) = \{\overline{a}.\mathbf{0}\} \cup sub(\mathbf{0},\emptyset) = \{\overline{a}.\mathbf{0}, \mathbf{0}\}$. Hence,

$$\begin{aligned} sub(q,\emptyset) &= sub(C \,|\, \overline{a}.\mathbf{0}, \emptyset)\{a'/a\} \\ &= (sub(C,\emptyset) \cup sub(\overline{a}.\mathbf{0},\emptyset))\{a'/a\} \\ &= \{C, a.C+b.\mathbf{0}, a.C, b.\mathbf{0}, \mathbf{0}, \overline{a}.\mathbf{0}\}\{a'/a\} \\ &= \{C_{\{a'/a\}}, a'.C_{\{a'/a\}} + b.\mathbf{0}, a'.C_{\{a'/a\}}, b.\mathbf{0}, \mathbf{0}, \overline{a'}.\mathbf{0}\}, \end{aligned}$$

where the new constant $C_{\{a'/a\}}$ is defined as $C_{\{a'/a\}} \doteq a'.C_{\{a'/a\}} + b.\mathbf{0}$.

Now, we want to show that the same set of sequential processes can be computed by first mapping q to $i(q) = (C \,|\, \overline{a}.\mathbf{0})\{a'/a\} = C_{\{a'/a\}} \,|\, \overline{a'}.\mathbf{0}$, i.e., it holds that $sub(q) = sub(i(q))$. The details are as follows. First note that, as we did above for constant C, we can compute

$$sub(C_{\{a'/a\}},\emptyset) = \{C_{\{a'/a\}}, a'.C_{\{a'/a\}} + b.\mathbf{0}, a'.C_{\{a'/a\}}, b.\mathbf{0}, \mathbf{0}\}.$$

Therefore,

$$\begin{aligned} sub(i(q),\emptyset) &= sub(C_{\{a'/a\}} \,|\, \overline{a'}.\mathbf{0}, \emptyset) \\ &= sub(C_{\{a'/a\}},\emptyset) \cup sub(\overline{a'}.\mathbf{0},\emptyset) \\ &= \{C_{\{a'/a\}}, a'.C_{\{a'/a\}} + b.\mathbf{0}, a'.C_{\{a'/a\}}, b.\mathbf{0}, \mathbf{0}\} \cup \{\overline{a'}.\mathbf{0}, \mathbf{0}\} \\ &= \{C_{\{a'/a\}}, a'.C_{\{a'/a\}} + b.\mathbf{0}, a'.C_{\{a'/a\}}, b.\mathbf{0}, \mathbf{0}, \overline{a'}.\mathbf{0}\} \\ &= sub(q,\emptyset). \end{aligned}$$

□

Lemma 6.1. *For any extended, restriction-free FNC process t and for any $L \subseteq \mathscr{L}$, $sub(t)\{L'/L\} = sub(t\{L'/L\})$.*

Proof. By induction on the definition of $sub(t,\emptyset)$. We will prove that $sub(t,I)\{L'/L\} = sub(t\{L'/L\}, I\{L'/L\})$, where $I\{L'/L\} = \{C_{\{L'/L\}} \mid C \in I\}$.

The two base cases are the following. If $t = \mathbf{0}$, then $t\{L'/L\} = \mathbf{0}$, and so $sub(t\{L'/L\}, I\{L'/L\}) = \{\mathbf{0}\}$. The thesis follows as $sub(t,I)\{L'/L\} = \{\mathbf{0}\}\{L'/L\} =$

$\{\mathbf{0}\}$. If $t = C$ and $C \in I$, then $C\{L'/L\} = C_{\{L'/L\}}$ and $C_{\{L'/L\}} \in I\{L'/L\}$. Hence, $sub(C\{L'/L\}, I\{L'/L\}\}) = \emptyset$. The thesis follows as $sub(C,I)\{L'/L\} = \emptyset\{L'/L\} = \emptyset$.

Then, the inductive cases follow. If $t = \mu.t'$, then $t\{L'/L\} = \mu.(t'\{L'/L\})$, if $\mu \neq a, \overline{a}$ for all $a \in L$. Hence, $sub(t\{L'/L\}, I\{L'/L\}) = sub(\mu.(t'\{L'/L\}), I\{L'/L\}) = \{\mu.(t'\{L'/L\})\} \cup sub(t'\{L'/L\}, I\{L'/L\})$. By induction, we have $sub(t', I)\{L'/L\} = sub(t'\{L'/L\}, I\{L'/L\})$; therefore, $sub((\mu.t')\{L'/L\}, I\{L'/L\}) = \{\mu.(t'\{L'/L\})\} \cup sub(t'\{L'/L\}, I\{L'/L\}) = \{\mu.t'\}\{L'/L\} \cup sub(t', I)\{L'/L\} = sub(\mu.t', I)\{L'/L\}$. The case when $\mu = a$ or $\overline{a}$ for some $a \in L$ is similar, hence omitted.

If $t = s_1 + s_2$, then $sub(t\{L'/L\}, I\{L'/L\}) = sub(s_1\{L'/L\} + s_2\{L'/L\}, I\{L'/L\})$ $= \{s_1\{L'/L\} + s_2\{L'/L\}\} \cup sub(s_1\{L'/L\}, I\{L'/L\}) \cup sub(s_2\{L'/L\}), I\{L'/L\})$. By induction, we have $sub(s_i\{L'/L\}, I\{L'/L\}) = sub(s_i, I)\{L'/L\}$ for $i = 1,2$. Therefore, $\{s_1\{L'/L\} + s_2\{L'/L\}\} \cup sub(s_1\{L'/L\}, I\{L'/L\}) \cup sub(s_2\{L'/L\}, I\{L'/L\}) = \{s_1 + s_2\}\{L'/L\} \cup sub(s_1, I)\{L'/L\} \cup sub(s_2, I)\{L'/L\} = sub(s_1 + s_2, I)\{L'/L\}$.

The case when $t = t_1 \,|\, t_2$ is similar to the above, hence omitted.

If $t = C$, with $C \doteq s$, then $C\{L'/L\} = C_{\{L'/L\}}$, where $C_{\{L'/L\}} \doteq s\{L'/L\}$. By induction, we have that $sub(s\{L'/L\}, I\{L'/L\} \cup \{C_{\{L'/L\}}\}) = sub(s, I \cup \{C\})\{L'/L\}$. Therefore, the thesis follows: $sub(C\{L'/L\}, I\{L'/L\}) = sub(C_{\{L'/L\}}, I\{L'/L\}) = \{C_{\{L'/L\}}\} \cup sub(s\{L'/L\}, I\{L'/L\} \cup \{C_{\{L'/L\}}\}) = \{C\}\{L'/L\} \cup sub(s, I \cup \{C\})\{L'/L\}$ $= sub(C,I)\{L'/L\}$, as required. □

Proposition 6.4. *For any FNC process p, $sub(p) = sub(i(p))$.*

Proof. *If p is restriction-free, then $i(p) = p$, and so the thesis follows trivially. If $p = (\nu L)t$, then $sub(p) = sub(t)\{L'/L\}$. Moreover, $i(p) = t\{L'/L\}$. So, the thesis follows by Lemma 6.1: $sub(i(p)) = sub(t\{L'/L\}) = sub(t)\{L'/L\} = sub(p)$.* □

Theorem 6.1. *For any FNC process p, the set of its sequential subterms $sub(p)$ is finite; moreover, $sub(p) \subseteq \mathscr{P}_{FNC}^{\gamma,seq}$.*

Proof. *By induction on the definition of $sub(p, \emptyset)$. The first base case is when $p = \mathbf{0}$; in this case, $sub(\mathbf{0}, I) = \{\mathbf{0}\}$ for any $I \subseteq \mathscr{C}ons$, and the thesis follows trivially: $\{\mathbf{0}\}$ is a finite set and $\mathbf{0} \in \mathscr{P}_{FNC}^{\gamma,seq}$. The second base case is when $p = C$ and $C \in I$; in this case, $sub(C,I) = \emptyset$, and the thesis follows trivially.*

The inductive cases are as follows. If $p = \mu.p'$, then $sub(\mu.p', I) = \{\mu.p'\} \cup sub(p', I)$. By induction, we can assume that $sub(p', I)$ is finite and a subset of $\mathscr{P}_{FNC}^{\gamma,seq}$. Hence, $sub(\mu.p', I)$ is finite and a subset of $\mathscr{P}_{FNC}^{\gamma,seq}$ because also $\mu.p'$ is a sequential process.

If $p = C$, with $C \doteq s$, then $sub(C,I) = \{C\} \cup sub(s, I \cup \{C\})$. By induction, we can assume that $sub(s, I \cup \{C\})$ is finite and a subset of $\mathscr{P}_{FNC}^{\gamma,seq}$. Hence, the thesis follows trivially, because C is a sequential process.

If $p = s_1 + s_2$, then $sub(s_1 + s_2, I) = \{s_1 + s_2\} \cup sub(s_1, I) \cup sub(s_2, I)$. By induction, we can assume that, for $i = 1,2$, $sub(s_i, I)$ is finite and a subset of $\mathscr{P}_{FNC}^{\gamma,seq}$. Hence, the thesis follows trivially, because $s_1 + s_2$ is a sequential process.

If $p = p_1 \,|\, p_2$, then the proof is similar to the above, hence omitted.

If $p = (\nu a)p_1$, then $sub(p, I) = sub(p_1, I)\{a'/a\}$. By induction, we can assume that $sub(p_1, I)$ is finite and a subset of $\mathscr{P}_{FNC}^{\gamma,seq}$. Hence, also $sub(p, I)$ is finite and, by Proposition 6.2, a subset of $\mathscr{P}_{FNC}^{\gamma,seq}$. □

(Pref)	$\dfrac{}{\mu.p \xrightarrow{\mu} p}$	(Cons)	$\dfrac{p \xrightarrow{\mu} p'}{C \xrightarrow{\mu} p'}$	$C \doteq p$
(Sum$_1$)	$\dfrac{p \xrightarrow{\mu} p'}{p+q \xrightarrow{\mu} p'}$	(Sum$_2$)	$\dfrac{q \xrightarrow{\mu} q'}{p+q \xrightarrow{\mu} q'}$	
(Par$_1$)	$\dfrac{p \xrightarrow{\mu} p'}{p\,\vert\,q \xrightarrow{\mu} p'\,\vert\,q}$	(Par$_2$)	$\dfrac{q \xrightarrow{\mu} q'}{p\,\vert\,q \xrightarrow{\mu} p\,\vert\,q'}$	
(Com)	$\dfrac{p \xrightarrow{\alpha} p' \qquad q \xrightarrow{\overline{\alpha}} q'}{p\,\vert\,q \xrightarrow{\tau} p'\,\vert\,q'}$	(Res)	$\dfrac{p \xrightarrow{\mu} p'}{(\nu a)p \xrightarrow{\mu} (\nu a)p'}$	$\mu \neq a, \bar{a}$

Table 6.5 Structural operational semantics for FNC

Proposition 6.5. *For any FNC processes p and q, the following hold:*

(i) *if p is sequential, then $p \in sub(p)$, and*
(ii) *if $p \in sub(q)$, then $sub(p) \subseteq sub(q)$.*

Proof. The proof of (i) follows directly from the definition of $sub(-)$ in Table 6.4. The proof of (ii) follows by the observation that the definition of $sub(q)$ recursively calls itself on all of its subterms. □

6.2 Operational LTS Semantics

Definition 6.7. The FNC labeled transition system $\mathscr{C}_{FNC}$ is the triple $(\mathscr{P}_{FNC}, Act, \rightarrow)$ where $\rightarrow \subseteq \mathscr{P}_{FNC} \times Act \times \mathscr{P}_{FNC}$ is the least transition relation generated by the axiom and rules in Table 6.5. □

Let us comment on the rules for the parallel composition operator and the restriction operator in Table 6.5.

(Par$_1$) and (Par$_2$) are the rules describing the asynchronous execution of an action by one of the two subcomponents of a parallel process. Specifically, (Par$_1$) states that if $p \xrightarrow{\mu} p'$, then $p\,|\,q \xrightarrow{\mu} p'\,|\,q$. Note that q is not discarded by the transition, and for this reason the operator is called *static*. (Par$_2$) is symmetric. These two rules together state that $p\,|\,q$ can do whatever p and q can do, possibly interleaving their executions. Rule (Com) describes how interaction can take place: if the two subcomponents can execute complementary input/output actions at the same time, then a synchronization is possible and the resulting transition, labeled τ, cannot be used for further interaction with another parallel component, as rule (Com) requires that each premise transition be labeled with a visible action. Hence, communication in FNC, like in CCS [Mil89, GV15], is strictly binary (or point-to-point) and synchronous, sometimes called *handshake* communication.

Rule (Res) explains that the role of a restriction is to bind a name, so that it is not freely available for the external environment. Clearly, $(\nu a)p$ impedes any transition labeled a or $\overline{a}$ that p might produce, while having no effect on the other transitions of p, in particular on possible synchronizations (labeled τ) along channel a between parallel subcomponents of p. Restriction is a *static* operator, too.

Proposition 6.6. *For any FNC process p, if $p \xrightarrow{\mu} p'$, then $fn(p') \subseteq fn(p)$, and if $\mu \in \{a, \overline{a}\}$, then $a \in fn(p)$.*

Proof. *By induction on the proof of transition $p \xrightarrow{\mu} p'$.* □

Corollary 6.1. *For any FNC process p, the set $sort(p)$ is finite.*

Proof. *By Proposition 6.6, we know that any action $\mu \in sort(p)$ is also an action in $fn(p) \cup \overline{fn(p)} \cup \{\tau\}$. The thesis follows by Proposition 6.1, which ensures that $fn(p)$ is finite for any FNC process p.* □

Remark 6.3. The operational LTS rules of Table 6.5 can be used to give an operational semantics also to extended FNC processes. There is only one caution: of the transitions derivable from the rules, we should take only those that are labeled over Act, thus discarding those transitions labeled with a restricted action. However, derivable transitions labeled with complementary restricted actions give rise, by rule (Com), to a synchronization transition, which is to be taken, being labeled τ. This issue will be clarified better in Section 6.4.2, about FNC net transitions. □

Interleaving bisimilarity, denoted by $\sim$, is a congruence for the action prefixing operator and for the choice operator, as proved in Theorem 4.4; it is also a congruence for the parallel composition operator: the proof of Theorem 5.2 can cope also with the communication capability of the FNC parallel composition operator. Now we show that it is also a congruence for the restriction operator.

Theorem 6.2. (Congruence) *For any $p, q \in \mathscr{P}_{FNC}$ and for any $a \in \mathscr{L}$, if $p \sim q$, then $(\nu a)p \sim (\nu a)q$.*

Proof. *Assume R is a bisimulation such that $(p,q) \in R$. We show that relation $R' = \{((\nu a)p', (\nu a)q') \mid a \in \mathscr{L} \text{ and } (p',q') \in R\}$ is a bisimulation. Since $(p,q) \in R$, the thesis follows. Consider $((\nu a)p', (\nu a)q') \in R'$, so that $(p',q') \in R$. If $(\nu a)p' \xrightarrow{\mu} s$, then this is due, by rule (Res), to transition $p' \xrightarrow{\mu} p''$, with $\mu \neq a, \overline{a}$ and $s = (\nu a)p''$. As $(p',q') \in R$, there exists q'' such that $q' \xrightarrow{\mu} q''$ and $(p'',q'') \in R$; by rule (Res), also $(\nu a)q' \xrightarrow{\mu} (\nu a)q''$ is derivable, with $((\nu a)p'', (\nu a)q'') \in R'$, as required. The symmetric case when $(\nu a)q'$ moves first is analogous, and so omitted.* □

6.2.1 Expressiveness

In the theory of formal languages [HMU01], the alphabet is a set of unstructured symbols; hence, in our setting, we assume that the alphabet is the set of actions $\mathscr{L}$.

So, an FNC language, being a subset of $\mathscr{L}^*$, is defined by means of an output-closed FNC process.

Definition 6.8. (FNC language) A language $L \subseteq \mathscr{L}^*$ is an *FNC language* if there exists an output-closed FNC process p such that the set of its weak completed traces is L, i.e., $WCTr(p) = L$. □

The class of FNC languages includes the class of BPP languages, as the set of BPP processes is a subset of (output-closed) FNC processes. As an instance of an FNC language that is not a BPP language, consider the following example.

Example 6.4. It has been proved in [Chr93, BCMS01] that $L = \{a^n cb^n \mid n \geq 0\}$ is not a BPP language. However, an FNC process for this language can be

$$\begin{aligned} F &\doteq (\nu d)A, \\ A &\doteq a.(A \,|\, d.b.\mathbf{0}) + c.B, \\ B &\doteq \overline{d}.B. \end{aligned}$$

The set $WCTr(F)$ is composed of all the weak completed traces of the form $a^n cb^n$ for any $n \in \mathbb{N}$. This is a typical example of a context-free language. □

However, the class of FNC languages does not include all the context-free languages. It has been proved in [Pet81] that the typical context-free language $L = \{ww^R \mid w \in \{a,b\}^*\}$, where w^R stands for the reverse of w (formally, $\varepsilon^R = \varepsilon$ and $(\alpha w)^R = w^R \alpha$) cannot be represented by means of a finite P/T Petri net. Since FNC processes may originate, via the semantics in Section 6.4, only finite CCS Petri nets, then they cannot represent the language L. Nonetheless, the class of finite-net CCS languages also includes some context-dependent languages, as the following example shows.

Example 6.5. **(The language $a^n b^m c^m$ with $0 \leq m \leq n$)** An FNC process whose weak completed traces are of the form $a^n b^m c^m$, with $0 \leq m \leq n$, is

$$\begin{aligned} ABC &\doteq (\nu d,e,f)A, \\ A &\doteq a.(A \,|\, d.b.\overline{e}.\mathbf{0}) + \tau.B, \\ B &\doteq \overline{d}.e.(B \,|\, f.c.\mathbf{0}) + \tau.C, \\ C &\doteq \overline{f}.C. \end{aligned}$$

Observe that first a certain number of occurrences of action a are generated, as well as the same number of subprocesses $d.b.\overline{e}.\mathbf{0}$. Then, when A performs the internal τ-labeled transition and becomes B, a certain number of activations of action b via a synchronization on d can be performed; these synchronizations cannot be more than the number of a's. When each occurrence of action b is performed, a *call-back* synchronization on e is executed, which activates a new instance of B, as well as of $f.c.\mathbf{0}$; hence, the number of processes $f.c.\mathbf{0}$ in parallel is equal to the number of b's executed. Finally, when B internally moves to C, perhaps before having completed the activations of all the available occurrences of b, the same number of c's can be performed after activation via synchronization on f. (See also Examples 3.10 and 8.5.) □

$$
\begin{array}{llllll}
(\mathrm{Pref}^s) & \dfrac{}{\mu.p \xrightarrow{\{\mu\}}_s p} & (\mathrm{Cons}^s) & \dfrac{p \xrightarrow{\{\mu\}}_s p'}{C \xrightarrow{\{\mu\}}_s p'} & C \doteq p \\[2ex]
(\mathrm{Sum}_1^s) & \dfrac{p \xrightarrow{\{\mu\}}_s p'}{p+q \xrightarrow{\{\mu\}}_s p'} & (\mathrm{Res}^s) & \dfrac{p \xrightarrow{M}_s p'}{(\nu a)p \xrightarrow{M}_s (\nu a)p'} & a,\overline{a} \notin M \\[2ex]
(\mathrm{Sum}_2^s) & \dfrac{q \xrightarrow{\{\mu\}}_s q'}{p+q \xrightarrow{\{\mu\}}_s q'} & (\mathrm{Com}^s) & \dfrac{p \xrightarrow{M_1}_s p' \qquad q \xrightarrow{M_2}_s q'}{p\,|\,q \xrightarrow{M}_s p'\,|\,q'} & MSync(M_1 \oplus M_2, M) \\[2ex]
(\mathrm{Par}_1^s) & \dfrac{p \xrightarrow{M}_s p'}{p\,|\,q \xrightarrow{M}_s p'\,|\,q} & (\mathrm{Par}_2^s) & \dfrac{q \xrightarrow{M}_s q'}{p\,|\,q \xrightarrow{M}_s p\,|\,q'} &
\end{array}
$$

Table 6.6 Step operational semantics for FNC

$$
\frac{}{MSync(M,M)} \qquad \frac{MSync(M \oplus \{\tau\}, M')}{MSync(M \oplus \{\alpha, \overline{\alpha}\}, M')}
$$

Table 6.7 Step synchronization relation

6.3 Step Semantics

The step operational semantics for FNC is given by the step transition system $\mathscr{C}_{FNC}^{step} = (\mathscr{P}_{FNC}, \mathscr{B}, \longrightarrow_s)$, where $\mathscr{B} = \mathscr{M}_{fin}(Act)$ — i.e., the set of all finite multisets over Act — is the set of labels (ranged over by M, possibly indexed), and the transition relation $\longrightarrow_s \subseteq \mathscr{P}_{FNC} \times \mathscr{B} \times \mathscr{P}_{FNC}$ is the least relation generated by the rules listed in Table 6.6.

Axiom (Pref^s) states that $\mu.p$ can perform the singleton $\{\mu\}$, reaching p. Rules (Sum$_1^s$) and (Sum$_2^s$) assume that the transition in the premise is sequential, i.e., composed of one single action. Similarly, since the body of a constant C is a sequential process, it is assumed in the premise of rule (Cons^s) that the label is composed of one single action. Rule (Res^s) requires that M contains no occurrences of action a or $\overline{a}$ in M. The highlight of this semantics is rule (Com^s): it allows for the generation of multisets as labels, by using an additional auxiliary relation $MSync$, defined in Table 6.7, where $\oplus$ denotes multiset union. The intuition behind the definitions of rule (Com^s) and $MSync$ is that, whenever two parallel processes p and q perform steps M_1 and M_2, respectively, then we can put all the actions together — yielding $M_1 \oplus M_2$ — and see whether $MSync(M_1 \oplus M_2, \overline{M})$ holds. The resulting multiset $\overline{M}$ may be just $M_1 \oplus M_2$ (hence no synchronization takes place), according to axiom $MSync(M,M)$, or the multiset M' we obtain from the application of the rule: select two actions α and $\overline{\alpha}$ from $M_1 \oplus M_2$, synchronize them, producing τ, then recursively apply $MSync$ to $\{\tau\} \oplus (M_1 \oplus M_2) \ominus \{\alpha, \overline{\alpha}\}$ to obtain M'. This procedure of synchronizing complementary actions may go on until no pair of synchronizable actions can be found, but it may also stop at any moment due to the axiom $MSync(M,M)$.

It is possible to prove that $\mathscr{C}_{FNC}^{step}$ is *fully concurrent*, i.e., that for any $p \in \mathscr{P}_{FNC}$, if $p \xrightarrow{M_1 \oplus M_2}_s p'$, with $M_1 \neq \emptyset \neq M_2$, then there exist q_1 and q_2 such that $p \xrightarrow{M_1}_s q_1 \xrightarrow{M_2}_s p'$ and $p \xrightarrow{M_2}_s q_2 \xrightarrow{M_1}_s p'$.

Remark 6.4. The operational STS rules of Table 6.6 can be used to give operational semantics also to extended FNC processes. There is only one caution: of the step transitions derivable from the rules, we should take only those that are labeled over $\mathscr{B}$, thus discarding those transitions labeled with multisets containing some restricted actions. However, two derivable transitions labeled with multisets containing complementary restricted actions give rise, by rule (Com^s), to a new step transition where these two complementary restricted actions disappear in favor of an occurrence of τ, so that the resulting new transition may become acceptable. □

Remark 6.5. **(Notation)** For any $p \in \mathscr{P}_{FNC}$, the STS reachable from p is $\mathscr{C}_p^{step} = (\mathscr{P}_p^{step}, sort(p)_s, \rightarrow_s^p, p)$, where $\mathscr{P}_p^{step}$ is the set of processes reachable from p, $sort(p)_s$ is the set of multisets of actions that can be performed by p (formally, $sort(p)_s = \{M \in \mathscr{M}_{fin}(Act) \mid \exists p'.p \longrightarrow^* p' \xrightarrow{M}_s \}$) and $\rightarrow_s^p$ is the restriction of the transition relation to $\mathscr{P}_p^{step} \times sort(p)_s \times \mathscr{P}_p^{step}$. □

Step bisimilarity, denoted by $\sim_{step}$, is ordinary bisimulation equivalence over step transition systems. Step bisimilarity is more discriminating than interleaving bisimilarity $\sim$. As a matter of fact, any interleaving transition $p \xrightarrow{\mu} p'$ has one corresponding step transition $p \xrightarrow{\{\mu\}}_s p'$, which is obtained by mimicking the proof of $p \xrightarrow{\mu} p'$, by using for each rule in Table 6.5 the corresponding rule with the same name (plus the superscript s) in Table 6.6. A bit of care is needed for rule (Com): if $p|q \xrightarrow{\tau} p'|q'$ because $p \xrightarrow{\alpha} p'$ and $q \xrightarrow{\overline{\alpha}} q'$, then by induction we can assume that $p \xrightarrow{\{\alpha\}}_s p'$ and $q \xrightarrow{\{\overline{\alpha}\}}_s q'$, so that by rule (Com^s) also the step transition $p|q \xrightarrow{\{\tau\}}_s p'|q'$ is derivable, as $MSync(\{\alpha, \overline{\alpha}\}, \{\tau\})$ holds. Symmetrically, the proof of any step transition $p \xrightarrow{\{\mu\}}_s p'$ labeled with a singleton $\{\mu\}$ can be mimicked by the interleaving operational rules, as hinted above. Again, a bit of care is needed for rule (Com^s): if $p|q \xrightarrow{\{\tau\}}_s p'|q'$ is derivable by this rule, then necessarily the two premises must be $p \xrightarrow{\{\alpha\}}_s p'$ and $q \xrightarrow{\{\overline{\alpha}\}}_s q'$, because in the rule neither M_1 nor M_2 can be empty. Therefore, a step bisimulation $R \subseteq \mathscr{P}_{FNC} \times \mathscr{P}_{FNC}$ is also an interleaving bisimulation: if $(p,q) \in R$ and $p \xrightarrow{\mu} p'$, then also $p \xrightarrow{\{\mu\}}_s p'$ by the argument above; as R is a step bisimulation, there exists q' such that $q \xrightarrow{\{\mu\}}_s q'$ with $(p',q') \in R$; and then also $q \xrightarrow{\mu} q'$ is derivable, by the argument above, with $(p',q') \in R$; symmetrically, if q moves first. Hence, R is also an interleaving bisimulation.

Proposition 6.7. *For any $p,q \in \mathscr{P}_{FNC}$, if $p \sim_{step} q$, then $p \sim q$.* □

Step bisimilarity is a congruence for the operators of FNC; for the operators of action prefixing, choice and parallel composition, this was proved in Theorem

$dec(\mathbf{0}) = \emptyset$	$dec(p+q) = \{p+q\}$	
$dec(\mu.p) = \{\mu.p\}$	$dec(p\,\|\,q) = dec(p) \oplus dec(q)$	
$dec(C) = \{C\}$	$dec((\nu a)p) = dec(p)\{a'/a\}$	$a' \in \mathscr{L}'$

Table 6.8 Decomposition function for FNC

5.3 (where the proof for the parallel operator is correct also for the FNC parallel operator with communication capabilities). Now we show that it is a congruence for the restriction operator, too.

Theorem 6.3. (Step congruence) *For any $p,q \in \mathscr{P}_{FNC}$ and for any $a \in \mathscr{L}$, if $p \sim_{step} q$, then $(\nu a)p \sim_{step} (\nu a)q$.*

Proof. Assume R is a step bisimulation with $(p,q) \in R$. It is an easy exercise (similar to the proof of Theorem 6.2) to check that $R' = \{((\nu a)p', (\nu a)q') \mid a \in \mathscr{L}$ and $(p',q') \in R\}$ is a step bisimulation. Since $((\nu a)p, (\nu a)q) \in R'$, the thesis follows. □

6.4 Operational Net Semantics

In this section, we describe a technique for building a P/T net for the whole of FNC, starting from a description of its places and its net transitions. The resulting net $N_{FNC} = (S_{FNC}, Act, T_{FNC})$ is such that, for any $p \in \mathscr{P}_{FNC}$, the net system $N_{FNC}(dec(p))$ statically reachable from the initial marking $dec(p)$ is a statically reduced, finite CCS net; such a net system is denoted by $Net(p)$.

6.4.1 Places and Markings

The infinite set of FNC places, ranged over by s, is $S_{FNC} = \mathscr{P}^{\gamma,seq}_{FNC} \setminus \{\mathbf{0}\}$, i.e., the set of all sequential, *extended* FNC processes, except $\mathbf{0}$.

Function $dec : \mathscr{P}^{\gamma}_{FNC} \to \mathscr{M}_{fin}(S_{FNC})$, which defines the decomposition of extended processes into markings, is outlined in Table 6.8. Process $\mathbf{0}$ generates no places. The decomposition of a sequential process p produces one place with name p. This is the case of $\mu.p$ (where μ can be any action in Act_γ), a constant C and $p+q$. Parallel composition is interpreted as multiset union; e.g., the decomposition of $a.\mathbf{0} \,|\, a.\mathbf{0}$ produces the marking $a.\mathbf{0} \oplus a.\mathbf{0} = 2 \cdot a.\mathbf{0}$. The decomposition of a general process $(\nu a)p$ — where $a \in \mathscr{L}$ — generates the multiset obtained from the decomposition of p, to which the substitution $\{a'/a\}$ is applied; the application of the substitution $\{a'/a\}$ to a multiset is performed element-wise, as shown in the examples below. We assume that, in decomposing $(\nu a)p$, the choice of the restricted name is fixed by the rule that associates with a visible action a its *unique* corresponding restricted action a'. As a process is of the form $(\nu L)t$ with $L = \{a_1, a_2, \ldots, a_n\}$, it can first be mapped, via function i of Definition 6.5, to the extended, restriction-free process $t\{a'_1/a_1\}\dots\{a'_n/a_n\}$ (shortened as $t\{L'/L\}$, for $L' = \{a'_1,\dots,a'_n\} \subseteq \mathscr{L}'$), and then decomposed to obtain a multiset. Function *dec* essentially performs this decomposition, by removing the restriction (which can occur only externally, by syntactic definition) and by replacing the bound names in L with the corresponding restricted names in L'.

Lemma 6.2. *For any restriction-free, extended FNC process t, $dec(t)\{L'/L\} = dec(t\{L'/L\})$.*

Proof. *By induction on the definition of $dec(t)$. If $t = \mathbf{0}$, then $dec(\mathbf{0})\{L'/L\} = \emptyset\{L'/L\} = \emptyset = dec(\mathbf{0}) = dec(\mathbf{0}\{L'/L\})$. If t is sequential, then $dec(t) = \{t\}$; moreover, $t\{L'/L\}$ is still sequential, so that $dec(t\{L'/L\}) = \{t\{L'/L\}\}$. Therefore, the thesis follows trivially, because $\{t\}\{L'/L\} = \{t\{L'/L\}\}$. If $t = t_1 \,|\, t_2$, then $dec(t) = dec(t_1) \oplus dec(t_2)$; by induction, we have $dec(t_i)\{L'/L\} = dec(t_i\{L'/L\})$ for $i = 1,2$. Therefore,*

$$\begin{aligned}
dec(t)\{L'/L\} &= (dec(t_1) \oplus dec(t_2))\{L'/L\}\\
&= dec(t_1)\{L'/L\} \oplus dec(t_2)\{L'/L\}\\
&= dec(t_1\{L'/L\}) \oplus dec(t_2\{L'/L\})\\
&= dec(t_1\{L'/L\} \,|\, t_2\{L'/L\})\\
&= dec((t_1 \,|\, t_2)\{L'/L\})\\
&= dec(t\{L'/L\}).
\end{aligned}$$

□

Proposition 6.8. *For any restriction-free $t \in \mathscr{P}_{FNC}$, $dec((\nu L)t) = dec(i((\nu L)t)) = dec(t\{L'/L\})$.*

Proof. *By definition, $dec((\nu L)t) = dec(t)\{L'/L\}$. Then, by Lemma 6.2, $dec(t)\{L'/L\} = dec(t\{L'/L\})$.* □

This means that we can restrict our attention to extended, restriction-free processes built over the set of visible and restricted actions Act_γ, as a general process $(\nu L)t$ in $\mathscr{P}_{FNC}$ is mapped via *dec* to the same marking associated with $i((\nu L)t) = t\{L'/L\}$ in $\mathscr{P}_{FNC}^{\gamma,par}$.

Example 6.6. Consider the FNC process $p = (\nu a)p'$, where $p' = (a.\mathbf{0} \,|\, (a.\mathbf{0} \,|\, \overline{a}.\mathbf{0}))$. Then,

$$\begin{aligned}
dec(p) &= dec(p')\{a'/a\} = dec(a.\mathbf{0} \,|\, (a.\mathbf{0} \,|\, \overline{a}.\mathbf{0}))\{a'/a\}\\
&= (dec(a.\mathbf{0}) \oplus dec(a.\mathbf{0} \,|\, \overline{a}.\mathbf{0}))\{a'/a\}\\
&= (dec(a.\mathbf{0}) \oplus dec(a.\mathbf{0}) \oplus dec(\overline{a}.\mathbf{0}))\{a'/a\}\\
&= (a.\mathbf{0} \oplus a.\mathbf{0} \oplus \overline{a}.\mathbf{0})\{a'/a\} = a'.\mathbf{0} \oplus a'.\mathbf{0} \oplus \overline{a'}.\mathbf{0}\\
&= dec(a'.\mathbf{0} \,|\, (a'.\mathbf{0} \,|\, \overline{a'}.\mathbf{0})) = dec(i(p))
\end{aligned}$$

where a' is the restricted name in $\mathscr{L}'$ corresponding to a. □

Syntactic substitution of the restricted name a' for the visible action a is defined as expected, as the following example shows.

Example 6.7. Consider process $p = (\nu a)q$, where $q = (\nu a)(a.\mathbf{0}\,|\,b.\mathbf{0}\,|\,b.\mathbf{0})$. Then,

$$dec(q) = dec(a.\mathbf{0}\,|\,b.\mathbf{0}\,|\,b.\mathbf{0})\{a'/a\} = (a.\mathbf{0} \oplus 2\cdot b.\mathbf{0})\{a'/a\} = a'.\mathbf{0} \oplus 2\cdot b.\mathbf{0},$$
$$dec(p) = dec(q)\{a'/a\} = (a'.\mathbf{0} \oplus 2\cdot b.\mathbf{0})\{a'/a\} = a'.\mathbf{0} \oplus 2\cdot b.\mathbf{0}$$

so that the second application of the same substitution has no effect. □

It is easily seen that the decomposition function *dec* is well defined.

Proposition 6.9. *For any $p \in \mathscr{P}^{\gamma}_{FNC}$, $dec(p)$ is a finite multiset of places.*

Proof. By induction on the definition of $dec(p)$. If $p = \mathbf{0}$, then $dec(\mathbf{0}) = \emptyset$, which is a finite multiset. If p is sequential, then $dec(p) = \{p\}$, which is a finite multiset. If $p = p_1\,|\,p_2$, then $dec(p) = dec(p_1) \oplus dec(p_2)$; by induction, we can assume that $dec(p_i)$ is a finite multiset, for $i = 1,2$, so that the thesis follows trivially. If $p = (\nu a)p'$, then $dec(p) = dec(p')\{a'/a\}$; by induction, we can assume that $dec(p')$ is a finite multiset, so that the thesis follows trivially. □

Of course, function *dec* is not injective, because it considers the parallel operator to be commutative and associative, with $\mathbf{0}$ as neutral element. In fact, $dec((p\,|\,q)\,|\,r) = dec(p\,|\,(q\,|\,r))$, $dec(p\,|\,q) = dec(q\,|\,p)$ and $dec(p\,|\,\mathbf{0}) = dec(p)$. Further equalities are induced by function *dec* such as $dec((\nu a)p) = dec(p)$ if $a \notin fn(p)$ and $dec((\nu a)((\nu b)p)) = dec((\nu b)((\nu a)p))$. However, one can prove that function *dec* is surjective over *admissible* markings.

Definition 6.9. (Free names, admissible marking, complete marking) Function $fn(-)$ can be extended to markings as follows: $fn(m) = \bigcup_{s \in dom(m)} fn(s)$, with $fn(\emptyset) = \emptyset$. A marking $m \in \mathscr{M}_{fin}(S_{FNC})$ is *admissible*, denoted by $ad(m)$, if for all $a \in \mathscr{L}$, $\{a, a'\} \not\subseteq fn(m)$. A marking $m \in \mathscr{M}_{fin}(S_{FNC})$ is *complete* if an FNC process $p \in \mathscr{P}_{FNC}$ exists such that $dec(p) = m$. □

Note that, for any two markings m_1 and m_2, if $fn(m_2) \subseteq fn(m_1)$ and $ad(m_1)$ holds, then also $ad(m_2)$ holds. This fact will often be used in the following proofs.

Lemma 6.3. *For any $t \in \mathscr{P}^{\gamma,par}_{FNC}$, $fn(t) = fn(dec(t))$.*

Proof. By induction on the definition of $dec(t)$. If $t = \mathbf{0}$, then $fn(\mathbf{0}) = \emptyset = fn(\emptyset) = fn(dec(\mathbf{0}))$. If t is sequential, then $dec(t) = \{t\}$, and the thesis follows trivially. If $t = t_1\,|\,t_2$, then $dec(t) = dec(t_1) \oplus dec(t_2)$. By induction, we have that $fn(t_i) = fn(dec(t_i))$ for $i = 1,2$. Therefore, $fn(t) = fn(t_1) \cup fn(t_2) = fn(dec(t_1)) \cup fn(dec(t_2)) = fn(dec(t_1) \oplus dec(t_2)) = fn(dec(t))$, as required. □

Theorem 6.4. *A marking $m \in \mathscr{M}_{fin}(S_{FNC})$ is admissible iff it is complete.*

Proof. ($\Leftarrow$) We want to prove that if m is complete, then m is admissible. Hence, we assume that an FNC process $p = (\nu L)t$ exists such that $dec(p) = dec(t)\{L'/L\} = m$. By Proposition 6.8, $m = dec(t\{L'/L\})$. The extended, restriction-free term $t\{L'/L\}$ is an extended process because it satisfies the admissibility condition: if $a \in L$, then no occurrence of a is present in $t\{L'/L\}$, while if $a \notin L$, then no occurrence of a' is present in $t\{L'/L\}$, because t is an FNC process. The thesis then follows because

$fn(t\{L'/L\}) = fn(dec(t\{L'/L\})) = fn(m)$ by Lemma 6.3, and so also m is admissible.

($\Rightarrow$) We want to prove that if m is admissible, then m is complete. Take any marking $m = k_1 \cdot s_1 \oplus \ldots \oplus k_n \cdot s_n$, for $n \geq 0$ ($m = \emptyset$ if $n = 0$), where each $s_i \in S_{FNC}$ and $k_i > 0$, for $i = 1, \ldots, n$. Let $L' = \{a'_1, \ldots, a'_k\}$ be the set of restricted actions occurring in m, i.e., $L' = fn(m) \cap \mathscr{L}'$, and let $L = \{a_1, \ldots, a_k\}$. Since m is admissible, for any $a'_j \in L'$, no corresponding visible action $a_j \in L$ occurs free in m. Therefore, it is possible to find, for $i = 1, \ldots, n$, a process $p_i \in \mathscr{P}^{seq}_{FNC}$ such that $s_i = p_i\{L'/L\}$. Then, take process $p = (\nu L)(p_1^{k_1} \,|\, \ldots \,|\, p_n^{k_n})$, where $q^1 = q$ and $q^{n+2} = q \,|\, q^{n+1}$, assuming that no restriction is present if $L = \emptyset$ and that, if $n = 0$, $(p_1^{k_1} \,|\, \ldots \,|\, p_n^{k_n}) = \mathbf{0}$. It is easy to observe that $dec(p) = m$; in fact, $dec(p) = dec(p_1^{k_1} \,|\, \ldots \,|\, p_n^{k_n})\{L'/L\}$ $= (k_1 \cdot p_1 \oplus \ldots \oplus k_n \cdot p_n)\{L'/L\} = k_1 \cdot s_1 \oplus \ldots \oplus k_n \cdot s_n = m$. □

Hence, this theorem states not only that function *dec* maps FNC processes to admissible markings over S_{FNC}, but also that *dec* is surjective over this set.

We extend the definition of sequential subterm of a process p to a set of places S. The goal is to prove that the sequential subterms of $dom(dec(p))$ are the sequential subterms of p. This property will be useful in proving (Theorem 6.6) that each place statically reachable from $dom(dec(p))$ is a sequential subterm of p, so that, since $sub(p)$ is finite for any p (Theorem 6.1), the set of all the places statically reachable from $dom(dec(p))$ is finite, too.

Definition 6.10. Function $sub(-)$, defined over FNC processes in Table 6.4, can be extended to a finite set S of places (i.e., of sequential, extended processes) as follows: $sub(\emptyset) = \emptyset$ and $sub(S) = \bigcup_{s \in S} sub(s)$. □

Proposition 6.10. *For any finite set of places S_1 and S_2, if $S_1 \subseteq sub(S_2)$, then $sub(S_1) \subseteq sub(S_2)$.*

Proof. *By induction on the cardinality of S_1. If $|S_1| = 0$, then the thesis follows trivially. If $|S_1| = n+1$, let $S_1 = S'_1 \cup \{s\}$, with $s \notin S'_1$. If $S_1 \subseteq sub(S_2)$, then also $S'_1 \subseteq sub(S_2)$, and so, by induction, $sub(S'_1) \subseteq sub(S_2)$. As $sub(S'_1 \cup \{s\}) = sub(S'_1) \cup sub(s)$, it remains to prove that $sub(s) \subseteq sub(S_2)$, given $s \in sub(S_2)$, which follows by the observation that the definition of $sub(S_2)$ recursively calls itself on all of its subterms.* □

Proposition 6.11. *For any set of places S, $S \subseteq sub(S)$.*

Proof. *For any sequential process s, Definition 6.6 ensures that $s \in sub(s)$; hence, the thesis follows trivially.* □

Proposition 6.12. *For any $t \in \mathscr{P}^{\gamma,par}_{FNC}$, $sub(dom(dec(t))) \subseteq sub(t)$.*

Proof. *By induction on the definition of $dec(t)$.*

If $t = \mathbf{0}$, then $sub(dom(dec(\mathbf{0}))) = sub(\emptyset) = \emptyset \subseteq \{\mathbf{0}\} = sub(\mathbf{0})$, as required. If t is sequential, then $sub(dom(dec(t))) = sub(\{t\}) = sub(t)$, hence the thesis holds. If

$$\text{(pref)}\quad \frac{}{\{\mu.p\} \xrightarrow{\mu} dec(p)} \qquad\qquad \text{(cons)}\quad \frac{dec(p) \xrightarrow{\mu} m}{\{C\} \xrightarrow{\mu} m}\ C \doteq p$$

$$\text{(sum}_1\text{)}\quad \frac{dec(p) \xrightarrow{\mu} m}{\{p+q\} \xrightarrow{\mu} m} \qquad\qquad \text{(sum}_2\text{)}\quad \frac{dec(q) \xrightarrow{\mu} m}{\{p+q\} \xrightarrow{\mu} m}$$

$$\text{(com)}\quad \frac{m_1 \xrightarrow{\gamma} m_1' \;\; m_2 \xrightarrow{\overline{\gamma}} m_2'}{m_1 \oplus m_2 \xrightarrow{\tau} m_1' \oplus m_2'}\quad ad(m_1 \oplus m_2)$$

Table 6.9 Rules for net transitions

$t = t_1 \,|\, t_2$, then $dec(t) = dec(t_1) \oplus dec(t_2)$ and $sub(t) = sub(t_1) \cup sub(t_2)$. By Definition 6.10, $sub(dom(dec(t))) = sub(dom(dec(t_1)) \cup sub(dom(dec(t_2))$; by induction, we have that $sub(dom(dec(t_i)) \subseteq sub(t_i)$, for $i = 1,2$; hence, $sub(dom(dec(t))) = sub(dom(dec(t_1))) \cup sub(dom(dec(t_2))) \subseteq sub(t_1) \cup sub(t_2) = sub(t)$. □

Corollary 6.2. *For any $p \in \mathscr{P}_{FNC}$, $sub(dom(dec(p))) \subseteq sub(p)$.*

Proof. *If p is restriction-free, then the thesis follows by Proposition 6.12. If $p = (\nu L)t$, then $dec(p) = dec(t\{L'/L\})$ by Proposition 6.8. By Proposition 6.4, $sub(p) = sub(t\{L'/L\})$. By Proposition 6.12, $sub(dom(dec(t\{L'/L\}))) \subseteq sub(t\{L'/L\})$. So, $sub(dom(dec(p))) = sub(dom(dec(t\{L'/L\}))) \subseteq sub(t\{L'/L\}) = sub(p)$.* □

6.4.2 Net Transitions

Let $\rightarrow \subseteq \mathscr{M}_{fin}(S_{FNC}) \times Act_\gamma \times \mathscr{M}_{fin}(S_{FNC})$ be the least set of transitions generated by the axiom and rules in Table 6.9, where in a transition $m_1 \xrightarrow{\mu} m_2$, m_1 is the pre-set, $\mu \in Act_\gamma$ is the label and m_2 is the post-set.

Let us comment on the rules of Table 6.9. Axiom (pref) states that if one token is present in the place $\mu.p$, then a μ-labeled transition is derivable from marking $\{\mu.p\}$, producing the marking $dec(p)$. This holds for any μ, i.e., for the invisible action τ, for any visible action α as well as for any restricted action α'. Rule (sum$_1$) is as expected: the transitions from the place $p+q$ are those from the marking $dec(p)$; as p is sequential, $dec(p)$ is $\{p\}$ if $p \neq \mathbf{0}$, while, if $p = \mathbf{0}$, $dec(p) = \emptyset$, but no transition is derivable from the empty set, so that the rule is really applicable only when $dec(p) = \{p\}$. Rule (sum$_2$) is symmetric. Similarly, rule (cons) states that the transitions derivable from $\{C\}$ are those derivable from the place $\{p\}$, if $C \doteq p$ with $p \neq \mathbf{0}$. Finally, rule (com) requires that the pre-set of the transition be admissible in order to avoid producing synchronized transitions that have no counterpart in the LTS semantics (Proposition 6.18). Rule (com) explains how synchronization takes place: it is required that m_1 and m_2 perform complementary actions γ and $\overline{\gamma}$, producing τ. As an example, the net transition $\{b'.p, \overline{b'}.q\} \xrightarrow{\tau} dec(p) \oplus dec(q)$ is derivable as follows:

$$(\text{com})\ \dfrac{(\text{pref})\ \dfrac{}{\{b'.p\} \xrightarrow{b'} dec(p)} \qquad (\text{pref})\ \dfrac{}{\{\overline{b'}.q\} \xrightarrow{\overline{b'}} dec(q)}}{\{b'.p, \overline{b'}.q\} \xrightarrow{\tau} dec(p) \oplus dec(q)}$$

Transitions labeled with a restricted action must not be taken in the resulting net, as we accept only transitions labeled over $Act = \{\tau\} \cup \mathscr{L} \cup \overline{\mathscr{L}}$. However, they are useful in producing acceptable transitions, as two complementary restricted actions can synchronize, producing a τ-labeled transition. For instance, in the example above, the derivable transition $\{b'.p\} \xrightarrow{b'} dec(p)$ is not an acceptable transition because its label is not in Act, while $\{b'.p, \overline{b'}.q\} \xrightarrow{\tau} dec(p) \oplus dec(q)$ is so. Hence, the P/T Petri net for FNC is the triple $N_{FNC} = (S_{FNC}, Act,\ T_{FNC})$, where the set

$$T_{FNC} = \{(m_1, \mu, m_2) \ \big|\ m_1 \xrightarrow{\mu} m_2 \text{ is derivable by the rules and } \mu \in Act\}$$

is obtained by filtering out those transitions derivable by the rules that are labeled with a restricted action.

Some useful properties of net transitions are listed here. First, given a transition $t = (m_1, \mu, m_2)$, derivable by the rules in Table 6.9, we show that the subterms of marking m_2 are already present in the set of subterms of marking m_1.

Proposition 6.13. *Let $t = m_1 \xrightarrow{\mu} m_2$ be a transition derivable by the rules in Table 6.9. Then, $sub(dom(m_2)) \subseteq sub(dom(m_1))$.*

Proof. *By induction on the proof of transition t. The base case is axiom (pref), stating that $\{\mu.p\} \xrightarrow{\mu} dec(p)$. By Definition 6.10, $sub(\{\mu.p\}) = sub(\mu.p)$. By Definition 6.6, $sub(\mu.p) = \{\mu.p\} \cup sub(p)$. By Proposition 6.12, as p is restriction-free, we have $sub(dom(dec(p))) \subseteq sub(p)$; the thesis — $sub(dom(dec(p))) \subseteq sub(dom(\{\mu.p\}))$ – follows trivially. For rule (sum$_1$), the premise transition is $\{p\} \xrightarrow{\mu} m_2$ and $m_1 = \{p+q\}$. By induction, $sub(dom(m_2)) \subseteq sub(dom(\{p\})) = sub(p)$. By Definition 6.10, $sub(\{p+q\}) = sub(p+q)$. By Definition 6.6, $sub(p+q) = \{p+q\} \cup sub(p) \cup sub(q)$; hence, the thesis follows by transitivity. The cases of rules (sum$_2$) and (cons) are analogous, hence omitted. For rule (com), the premise transitions are $m_1' \xrightarrow{\gamma} m_2'$ and $m_1'' \xrightarrow{\overline{\gamma}} m_2''$, with $m_1 = m_1' \oplus m_1''$ and $m_2 = m_2' \oplus m_2''$. By induction, we have that $sub(dom(m_2')) \subseteq sub(dom(m_1'))$ as well as $sub(dom(m_2'')) \subseteq sub(dom(m_1''))$. Hence, $sub(dom(m_2)) = sub(dom(m_2' \oplus\ m_2'')) = sub(dom(m_2') \cup dom(m_2'')) = sub(dom(m_2')) \cup sub(dom(m_2')) \subseteq sub(dom(m_1')) \cup sub(dom(m_1'')) = sub(dom(m_1') \cup dom(m_1'')) = sub(dom(m_1' \oplus m_1'')) = sub(dom(m_1))$, as required.* □

Lemma 6.4. *Let $t = m_1 \xrightarrow{\mu} m_2$ be a transition derivable by the rules in Table 6.9. Then, $fn(m_2) \subseteq fn(m_1)$.*

Proof. *By induction on the proof of transition t. The base case is axiom (pref), stating that $\{\mu.p\} \xrightarrow{\mu} dec(p)$. By Definition 6.9, $fn(\{\mu.p\}) = fn(\mu.p)$. By Definition*

6.1, $fn(p) \subseteq fn(\mu.p)$. As p is restriction-free, by Lemma 6.3, $fn(p) = fn(dec(p))$. Hence, the thesis — $fn(dec(p)) \subseteq fn(\{\mu.p\})$ – follows trivially. For rule (sum$_1$), the premise transition is $\{p\} \xrightarrow{\mu} m_2$ and $m_1 = \{p+q\}$. By induction, we have that $fn(m_2) \subseteq fn(\{p\})$. By Definition 6.9, $fn(\{p\}) = fn(p)$ and $fn(\{p+q\}) = fn(p+q)$. By Definition 6.1, $fn(p+q) = fn(p) \cup fn(q)$; hence, the thesis — $fn(m_2) \subseteq fn(\{p+q\})$ — follows by transitivity. The cases of rules (sum$_2$) and (cons) are analogous, hence omitted. For rule (com), the premise transitions are $m_1' \xrightarrow{\gamma} m_2'$ and $m_1'' \xrightarrow{\overline{\gamma}} m_2''$, with $m_1 = m_1' \oplus m_1''$ and $m_2 = m_2' \oplus m_2''$. By induction, $fn(m_2') \subseteq fn(m_1')$ and $fn(m_2'') \subseteq fn(m_1'')$. Hence, as required, $fn(m_2) = fn(m_2' \oplus m_2'') = fn(m_2') \cup fn(m_2'') \subseteq fn(m_1') \cup fn(m_1'') = fn(m_1' \oplus m_1'') = fn(m_1)$. □

Proposition 6.14. *Let $t = m_1 \xrightarrow{\mu} m_2$ be a transition derivable by the rules in Table 6.9. Then, m_1 and m_2 are admissible.*

Proof. By induction on the proof of t, we can easily conclude that m_1 is admissible. As a matter of fact, any place $s \in S_{FNC}$ is a sequential, extended FNC process, *which is admissible by definition. So, the source marking of the transitions derivable by axiom (pref) and rules (cons), (sum$_1$) or (sum$_2$) is a singleton marking $\{p\}$ with p an admissible sequential term. For rule (com), the side condition requires that the source marking $m_1 \oplus m_2$ be admissible. Therefore, for any $t = m_1 \xrightarrow{\mu} m_2$ derivable by the rules in Table 6.9, m_1 is admissible; moreover, by Lemma 6.4, we have that $fn(m_2) \subseteq fn(m_1)$, and so also m_2 is admissible.* □

Proposition 6.15. *Let $t = m_1 \xrightarrow{\mu} m_2$ be a transition derivable by the rules in Table 6.9. Let m be an admissible marking such that $m[t\rangle m'$. Then, m' is admissible.*

Proof. If t is enabled at m, then ${}^\bullet t \subseteq m$; hence, $m = m_0 \oplus {}^\bullet t$, for some suitable m_0. By definition of transition firing, $m' = m \ominus {}^\bullet t \oplus t^\bullet = m_0 \oplus t^\bullet$. By Lemma 6.4, $fn(t^\bullet) \subseteq fn({}^\bullet t)$; consequently, $fn(m') = fn(m_0) \cup fn(t^\bullet) \subseteq fn(m_0) \cup fn({}^\bullet t) = fn(m)$. Therefore, also m' is admissible. □

As a set of places can be seen as a marking, we can generalize the results above about admissibility to statically reachable sets of places.

Theorem 6.5. *If S_1 is admissible and $S_1 \Longrightarrow^* S_k$, then S_k is admissible.*

Proof. By induction on the static reachability relation $\Longrightarrow^*$. The base case is $S_1 \Longrightarrow^* S_1$ and the thesis holds trivially. The inductive case is $S_1 \Longrightarrow^* S_{k-1} \overset{t}{\Longrightarrow} S_k$, for some $t \in T_{FNC}$, $t = m_1 \xrightarrow{\mu} m_2$.

By induction we can assume that S_{k-1} is admissible. Since t is statically enabled at S_{k-1}, we have that $dom(m_1) \subseteq S_{k-1}$, and so $fn(m_1) \subseteq fn(S_{k-1})$. The set $fn(S_k)$ is $fn(S_{k-1}) \cup fn(m_2)$. By Lemma 6.4, we have $fn(m_2) \subseteq fn(m_1)$. Therefore, $fn(S_k) = fn(S_{k-1}) \cup fn(m_2) \subseteq fn(S_{k-1}) \cup fn(m_1) = fn(S_{k-1})$; therefore, we can conclude that also S_k is admissible. □

Proposition 6.16. *If $t = m_1 \xrightarrow{\mu} m_2$ is a transition derivable by the rules in Table 6.9 and $L \subseteq \mathscr{L}$, then transition $t\{L'/L\} = m_1\{L'/L\} \overset{\mu\{L'/L\}}{\longrightarrow} m_2\{L'/L\}$ is derivable as well, where $\mu\{L'/L\} = \mu$ if $\mu, \overline{\mu} \notin L$, while $\mu\{L'/L\} = \mu'$ otherwise. And vice versa, if $t\{L'/L\}$ is derivable, then also t is derivable.*

Proof. By induction on the proof of transition t. The base case is axiom (pref), stating that $\{\mu.p\} \xrightarrow{\mu} dec(p)$. In this case, $\{\mu.p\}\{L'/L\} = \mu.p\{L'/L\}$, if $\mu, \overline{\mu} \notin L$, while $\{\mu.p\}\{L'/L\} = \mu'.p\{L'/L\}$ otherwise; also, $dec(p)\{L'/L\} = dec(p\{L'/L\})$ by Lemma 6.2; in any case, $\{\mu.p\}\{L'/L\} \overset{\mu\{L'/L\}}{\longrightarrow} dec(p)\{L'/L\}$ is derivable. For the inductive case of rule (sum$_1$), we want to prove that, if $\{p+q\} \xrightarrow{\mu} m$, then $\{p+q\}\{L'/L\} \overset{\mu\{L'/L\}}{\longrightarrow} m\{L'/L\}$ is derivable. The premise is $dec(p) \xrightarrow{\mu} m$ and we can assume, by induction, that $dec(p)\{L'/L\} \overset{\mu\{L'/L\}}{\longrightarrow} m\{L'/L\}$ is derivable; as p is sequential, $dec(p) = \{p\}$, and, by Lemma 6.2, $dec(p)\{L'/L\} = dec(p\{L'/L\})$, so that $\{p\}\{L'/L\} = \{p\{L'/L\}\}$. Hence, $\{p\{L'/L\} + q\{L'/L\}\} \overset{\mu\{L'/L\}}{\longrightarrow} m\{L'/L\}$ is derivable, where $\{p\{L'/L\} + q\{L'/L\}\} = \{p+q\}\{L'/L\}$, as required. The cases of rules (sum$_2$) and (cons) are similar, hence omitted. For rule (com), we want to prove that $(m_1 \oplus m_2)\{L'/L\} \xrightarrow{\tau} (m_1' \oplus m_2')\{L'/L\}$ is derivable, if $m_1 \oplus m_2 \xrightarrow{\tau} m_1' \oplus m_2'$, whose premises are $m_1 \xrightarrow{\gamma} m_1'$ and $m_2 \xrightarrow{\overline{\gamma}} m_2'$, with $ad(m_1 \oplus m_2)$. By induction, we can assume that $m_1\{L'/L\} \overset{\gamma\{L'/L\}}{\longrightarrow} m_1'\{L'/L\}$ and $m_2\{L'/L\} \overset{\overline{\gamma}\{L'/L\}}{\longrightarrow} m_2'\{L'/L\}$ are derivable; and so, since $ad(m_1\{L'/L\} \oplus m_2\{L'/L\})$ holds, by rule (com), also transition $m_1\{L'/L\} \oplus m_2\{L'/L\} \xrightarrow{\tau} m_1'\{L'/L\} \oplus m_2'\{L'/L\}$ is derivable, where $m_1\{L'/L\} \oplus m_2\{L'/L\} = (m_1 \oplus m_2)\{L'/L\}$ and $m_1'\{L'/L\} \oplus m_2'\{L'/L\} = (m_1' \oplus m_2')\{L'/L\}$, as required.

For the converse — if $t\{L'/L\}$ is derivable, then also t is derivable — observe that this is similar to the above: if $t\{L'/L\}$ is derivable, then $(t\{L'/L\})\{L/L'\} = t$ is derivable, where an inverse substitution $\{L/L'\}$ is used instead. □

The net transitions have a very restrictive form: either they have a singleton pre-set, or they have a pre-set composed of two tokens, but in this case the label of the transition is τ. This is proved in the following proposition.

Proposition 6.17. *For any $t \in T_{FNC}$, $1 \leq |^\bullet t| \leq 2$ and if $|^\bullet t| = 2$, then $l(t) = \tau$.*

Proof. By induction on the proof of t. The base case is when t is derived by axiom (pref). In this case, $^\bullet t = \{\mu.p\}$, and so $1 \leq |^\bullet t| \leq 2$ as required. If the last rule used for deriving t is (sum$_1$) or (sum$_2$), then $^\bullet t = \{p+q\}$, and so the thesis follows trivially. Similarly, for rule (cons). If rule (com) is the last rule used to derive t, then there exist two transitions $t_1 = m_1 \xrightarrow{\gamma} m_1'$ and $t_2 = m_2 \xrightarrow{\overline{\gamma}} m_2'$ such that $t = m_1 \oplus m_2 \xrightarrow{\tau} m_1' \oplus m_2'$. By induction, as $\gamma \neq \tau$, we can assume that $|m_1| = |m_2| = 1$, so that the thesis follows trivially: $|^\bullet t| = 2$ and $l(t) = \tau$. □

The following proposition states that the net N_{FNC} contains only transitions that have a counterpart in the LTS semantics for FNC, as outlined in Section 6.2; this depends crucially on the admissibility condition over rule (com), as illustrated in the following Example 6.8. This proposition exploits a lemma that states a correspondence between net transitions derivable by the rules in Table 6.9 — which can be labeled over Act_γ — and LTS transitions derivable by the rules in Table 6.5 from extended, restriction-free processes (see Remark 6.3).

Lemma 6.5. *For any $t = (m_1, \mu, m_2)$ derivable by the rules in Table 6.9, there exist two extended, restriction-free FNC processes p_1 and p_2 such that $p_1 \xrightarrow{\mu} p_2$ is derivable by the rules in Table 6.5, with $dec(p_1) = m_1$ and $dec(p_2) = m_2$.*

Proof. By induction on the proof of transition t, we show a proof for $p_1 \xrightarrow{\mu} p_2$, for suitable extended, restriction-free processes p_1 and p_2 such that $dec(p_1) = m_1$ and $dec(p_2) = m_2$. The base case is axiom (pref): $t = \{\mu.p\} \xrightarrow{\mu} dec(p)$, with $\mu \in Act_\gamma$; in this case, axiom (Pref) ensures that $\mu.p \xrightarrow{\mu} p$; note that $\mu.p$ is an extended, sequential process and so p is an extended, restriction-free process, as required. If the last rule used to derive t is (sum$_1$), then $t = \{p+q\} \xrightarrow{\mu} m_2$, with a premise $t' = \{p\} \xrightarrow{\mu} m_2$. By induction, we know that $p \xrightarrow{\mu} p_2$ is derivable, with p_2 an extended, restriction-free process such that $dec(p_2) = m_2$; hence, by rule (Sum$_1$), also transition $p+q \xrightarrow{\mu} p_2$ is derivable, as required. The cases of rules (sum$_2$) and (cons) are similar, hence omitted. If the last rule used to derive t is (com), then there exist two transitions $t_1 = m_1' \xrightarrow{\gamma} m_2'$ and $t_2 = m_1'' \xrightarrow{\overline{\gamma}} m_2''$ such that $t = m_1' \oplus m_1'' \xrightarrow{\tau} m_2' \oplus m_2''$, and $ad(m_1' \oplus m_1'')$. By induction, there exist transition $q_1 \xrightarrow{\gamma} q_2$ — with q_1 and q_2 extended, restriction-free processes such that $dec(q_1) = m_1'$ and $dec(q_2) = m_2'$ — and transition $r_1 \xrightarrow{\overline{\gamma}} r_2$ — with r_1 and r_2 extended, restriction-free processes such that $dec(r_1) = m_1''$ and $dec(r_2) = m_2''$. Therefore, by rule (Com), also transition $q_1 \,|\, r_1 \xrightarrow{\tau} q_2 \,|\, r_2$ is derivable. Note that, by Lemma 6.3, $fn(q_1 \,|\, r_1) = fn(dec(q_1 \,|\, r_1)) = fn(m_1' \oplus m_1'')$; therefore, $q_1 \,|\, r_1$ is an extended, restriction-free process because it is admissible, since the marking $m_1' \oplus m_1''$ is assumed admissible by rule (com). Similarly, also $q_2 \,|\, r_2$ is admissible, because $fn(q_2 \,|\, r_2) = fn(dec(q_2 \,|\, r_2)) = fn(m_2' \oplus m_2'')$ and the marking $m_2' \oplus m_2''$ is admissible by Proposition 6.14. □

Proposition 6.18. *For any $t \in T_{FNC}$, where $t = (m_1, \mu, m_2)$ with $\mu \in Act$, there exist two FNC processes p_1 and p_2 such that $p_1 \xrightarrow{\mu} p_2$, $dec(p_1) = m_1$ and $dec(p_2) = m_2$.*

Proof. By Lemma 6.5, transition $q_1 \xrightarrow{\mu} q_2$ is derivable by the rules in Table 6.5, for some suitable extended, restriction-free processes q_1 and q_2 such that $dec(q_1) = m_1$ and $dec(q_2) = m_2$. Let $L' = fn(m_1) \cap \mathscr{L}'$ and $L = \{a \mid a' \in L'\}$. If $L' = \emptyset$, then q_1 is actually an FNC process and the required transition is exactly $q_1 \xrightarrow{\mu} q_2$, with q_2 also an FNC process by Proposition 6.6. On the contrary, if $L' \neq \emptyset$, then for all $a \in L, a \notin fn(m_1)$ because m_1 is admissible, so that $\mu \neq a, \overline{a}$. Moreover, by Proposition 6.16, also $t\{L/L'\}$ is derivable and the same proof of transition $q_1 \xrightarrow{\mu} q_2$ can be used to prove that transition $q_1\{L/L'\} \xrightarrow{\mu\{L/L'\}} q_2\{L/L'\}$

is derivable by the rules in Table 6.5, where $\mu\{L/L'\} = \mu$ as $\mu \in Act$. Note that $q_1\{L/L'\}$ and $q_2\{L/L'\}$ are FNC processes, because, for $i = 1,2$, $fn(q_i) = fn(dec(q_i)) = fn(m_i)$ by Lemma 6.3, and so all the restricted actions a' occurring in q_i are turned into their corresponding actions a in $q_i\{L/L'\}$. Finally, by rule (Res), also $(\nu L)(q_1\{L/L'\}) \xrightarrow{\mu} (\nu L)(q_2\{L/L'\})$ is derivable, where, for $i = 1,2$, $(\nu L)(q_i\{L/L'\})$ is an FNC process. Note that $dec((\nu L)(q_i\{L/L'\})) = m_i$, as required, because $dec((\nu L)(q_i\{L/L'\})) = dec(q_i\{L/L'\})\{L'/L\}$ by definition of function dec, and $dec(q_i\{L/L'\})\{L'/L\} = dec((q_i\{L/L'\})\{L'/L\})$ by Lemma 6.2 and $dec((q_i\{L/L'\})\{L'/L\}) = dec(q_i)$ by Proposition 6.3. □

Example 6.8. **(Admissibility condition for rule (com))** The following net transition $\{a.b'.\mathbf{0}, \overline{a}.b.\mathbf{0}\} \xrightarrow{\tau} \{b'.\mathbf{0}, b.\mathbf{0}\}$ would be derivable if the side condition of rule (com) about admissibility of the pre-set were removed. It is easily seen that there are no FNC processes p_1 and p_2 such that $p_1 \xrightarrow{\tau} p_2$, with $dec(p_1) = \{a.b'.\mathbf{0}, \overline{a}.b.\mathbf{0}\}$ and $dec(p_2) = \{b'.\mathbf{0}, b.\mathbf{0}\}$, because $dec(p_1)$ and $dec(p_2)$ must be admissible by Theorem 6.4, while $\{a.b'.\mathbf{0}, \overline{a}.b.\mathbf{0}\}$ and $\{b'.\mathbf{0}, b.\mathbf{0}\}$ are not admissible. □

We now want to prove that for any finite set of places $S \subseteq S_{FNC}$, the set of transitions statically enabled at S is finite. An auxiliary lemma is necessary. Given a single place $s \in S$, by $s \vdash t$ we mean that transition $t = (\{s\}, \mu, m)$ is derivable by the rules in Table 6.9, hence with $\mu \in Act_\gamma$.

Lemma 6.6. *The set $T_s = \{t \mid s \vdash t\}$ is finite, for any $s \in S_{FNC}$.*

Proof. By induction on the axiom and rules in Table 6.9. We proceed by case analysis. If $s = \mu.p$, by (pref), only one transition is derivable: $(\{\mu.p\}, \mu, dec(p))$. If $s = p_1 + p_2$, then we can assume, by induction, that there are finitely many transitions derivable by the premise places $\{p_1\}$ and $\{p_2\}$; for any such transition, say $(\{p_1\}, \mu, m)$ w.l.o.g., rule (sum$_1$) generates a transition $(\{p_1 + p_2\}, \mu, m)$, so that there are finitely many transitions from $p_1 + p_2$ as well. If $s = C$, with $C \doteq p$, then we can assume, by induction, that there are finitely many transitions derivable by place $\{p\}$; for any such transition, say $(\{p\}, \mu, m)$, rule (cons) generates a transition $(\{C\}, \mu, m)$, so that there are finitely many transitions from C, too. □

Given a finite set of places $S \subseteq S_{FNC}$, let T_1^S be $\bigcup_{s \in S} T_s$, i.e., the set of all transitions, with a singleton pre-set in S and labeling in Act_γ, derivable by the rules. The set T_1^S is finite, being the finite union (as S is finite) of finite sets (as T_s is finite for any s). The set

$$T_2^S = \{(m_1 \oplus m_2, \tau, m_1' \oplus m_2') \mid \exists \gamma. (m_1, \gamma, m_1') \in T_1^S, (m_2, \overline{\gamma}, m_2') \in T_1^S, ad(m_1 \oplus m_2)\}$$

defines all the transitions with a pre-set of cardinality two and derivable by means of rule (com) that are statically enabled at S. Note that T_2^S is finite, because T_1^S is finite. Note that no further transitions are derivable by the net rules. Therefore, the set of all the transitions statically enabled at S is

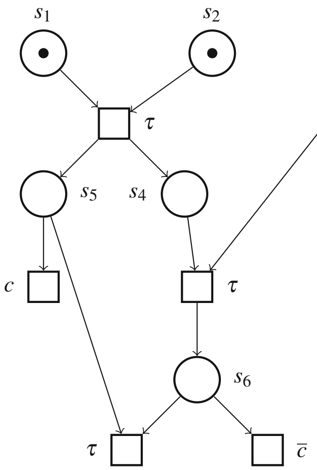


Fig. 6.1 The finite, acyclic CCS net for process $p = (\nu a)((\nu b)(a.b.\mathbf{0} \,|\, \overline{a}.c.\mathbf{0} \,|\, \overline{b}.\overline{c}.\mathbf{0}))$

$$T_S = \{t \;\big|\; t \in T_1^S \wedge l(t) \in Act\} \cup T_2^S,$$

where the only transitions of T_1^S which are taken are those labeled over Act. T_S is finite, being the union of two finite sets; therefore, we have the following result.

Proposition 6.19. *If $S \subseteq S_{FNC}$ is a finite set of places, then the set $T_S \subseteq T_{FNC}$ of all the transitions statically enabled at S is finite.* □

Example 6.9. **(Self-synchronization)** Consider the place $s = a.\mathbf{0} + \overline{a}.\mathbf{0}$ and the finite set of places $S = \{s\}$. The set T_1^S is $\{(\{s\}, a, \emptyset), (\{s\}, \overline{a}, \emptyset)\}$. The set T_2^S is the singleton $\{(\{s, s\}, \tau, \emptyset)\}$; this transition represents a self-synchronization of two instances of the same sequential process s and it is statically enabled at S, even if it is not dynamically enabled at the marking S. The set T_S is $T_1 \cup T_2$, because all the transitions are labeled over Act. The resulting net is outlined in Figure 1.6(b). □

A more complex example follows, where we show how to construct the net statically reachable from a given initial marking. This example anticipates the subject of the next section.

Example 6.10. Consider the FNC process $p = (\nu a)((\nu b)(a.b.\mathbf{0} \,|\, \overline{a}.c.\mathbf{0} \,|\, \overline{b}.\overline{c}.\mathbf{0}))$. The decomposition of p is the set $S = \{s_1, s_2, s_3\}$, where $s_1 = a'.b'.\mathbf{0}, s_2 = \overline{a'}.c.\mathbf{0}$, $s_3 = \overline{b'}.\overline{c}.\mathbf{0}$. The set T_1^S is $\{(\{s_1\}, a', \{s_4\}), (\{s_2\}, \overline{a'}, \{s_5\}),\ (\{s_3\}, \overline{b'}, \{s_6\})\}$, where $s_4 = b'.\mathbf{0}, s_5 = c.\mathbf{0}, s_6 = \overline{c}.\mathbf{0}$. Note that none of the transitions in T_1^S is labeled over Act. The set T_2^S is $\{t_0\}$, where $t_0 = (\{s_1, s_2\}, \tau, \{s_4, s_5\})$. The set T_S is actually T_2^S.

From S, we can perform the only statically enabled transition, namely t_0; hence, $S \overset{t_0}{\Longrightarrow} S'$, where $S' = S \cup \{s_4, s_5\}$. Now $T_1^{S'} = T_1^S \cup \{(\{s_4\}, b', \emptyset), (\{s_5\}, c, \emptyset)\}$ and $T_2^{S'} = T_2^S \cup \{(\{s_3, s_4\}, \tau, \{s_6\})\}$. The set $T_{S'}$ is $T_S \cup \{(\{s_5\}, c, \emptyset), (\{s_3, s_4\}, \tau, \{s_6\})\}$.

From S', there is only one statically enabled transition that produces some new places, namely $t_1 = (\{s_3, s_4\}, \tau, \{s_6\})$; hence, $S' \overset{t_1}{\Longrightarrow} S''$, where $S'' = S' \cup \{s_6\}$. Now $T_1^{S''} = T_1^{S'} \cup \{(\{s_6\}, \overline{c}, \emptyset)\}$ and $T_2^{S''} = T_2^{S'} \cup \{(\{s_5, s_6\}, \tau, \emptyset)\}$. The set $T_{S''}$ is $T_{S'} \cup \{(\{s_6\}, \overline{c}, \emptyset), (\{s_5, s_6\}, \tau, \emptyset)\}$. As no new places are reached by the newly added transitions, the construction of the net associated with p stops. The resulting net is $(S'', \{\tau, c, \overline{c}\}, T_{S''}, S)$, where $S = dec(p)$, and it is depicted in Figure 6.1. □

6.4.3 The Reachable Subnet $Net(p)$

The P/T net system associated with a process $p \in \mathscr{P}_{FNC}$ is the subnet of N_{FNC} statically reachable from the initial marking $dec(p)$, denoted by $Net(p)$.

Definition 6.11. Let p be a process in $\mathscr{P}_{FNC}$. The P/T net system statically associated with p is $Net(p) = (S_p, A_p, T_p, m_0)$, where $m_0 = dec(p)$ and

$$\begin{aligned} S_p &= [\![dom(m_0)\rangle \quad \text{computed in } N_{FNC}, \\ T_p &= \{t \in T_{FNC} \mid S_p[\![t\rangle\}, \\ A_p &= \{\mu \in Act \mid \exists t \in T_p \text{ such that } l(t) = \mu\}. \end{aligned}$$

□

The following propositions present three facts that are obviously true by construction of the net $Net(p)$ associated with an FNC process p.

Proposition 6.20. *For any $p \in \mathscr{P}_{FNC}$, $Net(p)$ is a statically reduced P/T net.* □

Proposition 6.21. *If $dec(p) = dec(q)$, then $Net(p) = Net(q)$.* □

Proposition 6.22. *For any restriction-free $t \in \mathscr{P}_{FNC}$ and for any $L \subseteq \mathscr{L}$, the following hold.*

i) If $Net(t) = (S, A, T, m_0)$, then, for any $n \geq 1$, $Net(t^n) = (S, A, T, n \cdot m_0)$, where $t^1 = t$ and $t^{n+1} = t \,|\, t^n$.
ii) If $Net((\nu L)t) = (S, A, T, m_0)$, then $Net((\nu L)(t^n)) = (S, A, T, n \cdot m_0)$, for any $n \geq 1$.

Proof. If $dec(t) = m_0$, then $dec(t^n) = n \cdot m_0$. The thesis follows by observing that $dom(m_0) = dom(n \cdot m_0)$, so that the starting set of places, from which to compute all the statically reachable places, is the same for both nets. Similarly, when considering processes $(\nu L)t$ and $(\nu L)(t^n)$. □

Definition 6.11 suggests a way of generating $Net(p)$ with an algorithm based on the inductive definition of the static reachability relation (see Definition 3.9): Start with the initial set of places $dom(dec(p))$, and then apply the rules in Table 6.9 in order to produce the set of transitions (labeled over Act) statically enabled at $dom(dec(p))$, as well as the additional places statically reachable by means of such transitions. Then repeat this procedure from the set of places statically reached so far. An instance of this procedure was given in Example 6.10. There are two problems with this algorithm:

- the obvious *halting condition* is “until no new places are statically reachable”; of course, the algorithm terminates if we know that the set S_p of places statically reachable from $dom(dec(p))$ is finite; additionally,
- at each step of the algorithm, we have to be sure that the set of transitions derivable from the current set of statically reachable places is finite.

We are going to prove only the first requirement — S_p is finite for any $p \in \mathscr{P}_{FNC}$ — because it implies also the second one. As a matter of fact, if S_p is finite, then any set S of places statically reachable from $dom(dec(p))$ is finite and, by Proposition 6.19, the set T_S of transitions statically enabled at S is finite as well.

Theorem 6.6. *For any $p \in \mathscr{P}_{FNC}$, let $Net(p) = (S_p, A_p, T_p, m_0)$ be defined as in Definition 6.11. Then, the set S_p is finite.*

Proof. *We prove, by induction on the static reachability relation $\Longrightarrow^*$, that any set S_i of places that is statically reachable from $dom(m_0) = dom(dec(p))$ is a subset of $sub(dom(m_0))$. This is enough as, by Corollary 6.2, $sub(dom(m_0)) \subseteq sub(p)$; moreover, by Theorem 6.1, $sub(p)$ is finite and so the thesis follows trivially.*

The base case is $dom(m_0) \Longrightarrow^* dom(m_0)$. By Proposition 6.11, we have the required $dom(m_0) \subseteq sub(dom(m_0))$. Now, let us assume that S_i is a set of places statically reachable from $dom(m_0)$ and let $t = m_1 \xrightarrow{\sigma} m_2$ be such that $S_i \overset{t}{\Longrightarrow} S_{i+1}$. By induction, we know that $S_i \subseteq sub(dom(m_0))$. So, we have to prove that the new places reached via t are in $sub(dom(m_0))$. Note that since $dom(m_1) \subseteq S_i$, it follows that $dom(m_1) \subseteq sub(dom(m_0))$ and also that $sub(dom(m_1)) \subseteq sub(dom(m_0))$, by Proposition 6.10. By Proposition 6.11, we have that $dom(m_2) \subseteq sub(dom(m_2))$; by Proposition 6.13, we have that $sub(dom(m_2)) \subseteq sub(dom(m_1))$; by transitivity, $dom(m_2) \subseteq sub(dom(m_0))$, and so $S_{i+1} = S_i \cup dom(m_2) \subseteq sub(dom(m_0))$, as required.

Summing up, any place statically reachable from $dom(m_0)$ is a sequential subterm of p. Since, by Theorem 6.1, $sub(p)$ is finite, then also S_p (the largest set of places statically reachable from $dom(m_0)$) is finite. □

Theorem 6.7. *For any FNC process p, $Net(p)$ is a finite P/T net.*

Proof. *The set $S_0 = dom(dec(p))$ is finite, by Proposition 6.9. By Proposition 6.19, the set T_{S_0} is finite. Let S_1 be the set of places $S_0 \cup \bigcup_{t \in T_{S_0}} dom(t^\bullet)$. If $S_1 = S_0$, then $S_p = S_0$ and $T_p = T_{S_0}$. Otherwise, repeat the step above for S_1; in fact, S_1 is a finite set of places, because S_0 is finite, the set T_{S_0} is finite and each transition has a finite post-set. By repeating the step above for S_1, we compute a new finite set T_{S_1} of transitions statically enabled at S_1, and a new finite set S_2 of places statically reachable from S_1 via the transitions in T_{S_1}; if $S_2 = S_1$, then $S_p = S_1$ and $T_p = T_{S_1}$. Otherwise, repeat the step above for S_2. This procedure will end eventually because, by Theorem 6.6, we are sure that S_p is a finite set.* □

Corollary 6.3. (Finite CCS nets) *For any $p \in \mathscr{P}_{FNC}$, $Net(p)$ is a finite CCS net.*

Proof. *By Theorem 6.7, $Net(p)$ is a finite P/T net. By Proposition 6.17, any transition $t \in T_{FNC}$ is such that $1 \leq |{}^\bullet t| \leq 2$ and if $|{}^\bullet t| = 2$, then $l(t) = \tau$. Hence, $Net(p)$ is a finite CCS net.* □

6.5 Representing All Finite CCS Nets

Corollary 6.3 ensures that *only* finite CCS nets can be represented by FNC processes. It is not completely obvious that *all* finite CCS nets can be represented by FNC processes. The construction described in Section 5.2.3 for BPP nets is to be modified non-trivially in order to cope with finite CCS nets.

As for the previous cases, the translation from nets to processes defines a constant C_i in correspondence to each place s_i; the constant C_i has a summand c_i^j for each transition t_j, which is $\mathbf{0}$ when s_i is not in the pre-set of t_j. Differently from the previous case, the FNC process $\mathscr{T}_{FNC}(N(m_0))$ associated with the finite CCS net system $N(m_0)$, labeled over $\mathscr{L} \cup \{\tau\}$, has a bound name x_i^j for each pair (s_i, t_j), where s_i is a place and t_j is a transition; such bound names are used to force synchronization between the two components participating in transition t_j with pre-set of cardinality two. Among the two places in the pre-set of t_j, the one with lower index (as we assume that places are indexed) plays the role of *leader* of the synchronization: the leader of index i has a summand $x_{i+h}^j.\Pi_j$, performing the input x_{i+h}^j — to be synchronized with the output performed by the *servant* participant $\bar{x}_{i+h}^j.\mathbf{0}$ of index $i+h$ — and specifying the continuation of the transition, namely process Π_j.

Definition 6.12. (Translating finite CCS nets into FNC processes) Given a set of labels $A \subseteq \mathscr{L} \cup \{\tau\}$, let $N(m_0) = (S, A, T, m_0)$ — with $S = \{s_1, \ldots, s_n\}$, $T = \{t_1, \ldots, t_k\}$, and $l(t_j) = \mu_j$ — be a finite CCS net. Function $\mathscr{T}_{FNC}(-)$, from finite CCS nets to FNC processes, is defined as

$$\mathscr{T}_{FNC}(N(m_0)) = (\nu L)(\underbrace{C_1|\cdots|C_1}_{m_0(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m_0(s_n)})$$

where $L = \{x_1^1, \ldots, x_n^1, x_1^2, \ldots, x_n^2, \ldots, x_1^k, \ldots, x_n^k\}$ is such that $L \cap A = \emptyset$, each C_i is equipped with a defining equation $C_i \doteq c_i^1 + \cdots + c_i^k$ (with $C_i \doteq \mathbf{0}$ if $k = 0$), and each summand c_i^j, for $j = 1, \ldots, k$, is equal to

- $\mathbf{0}$, if $s_i \not\in {}^\bullet t_j$;
- $\mu_j.\Pi_j$, if ${}^\bullet t_j = \{s_i\}$;
- $\bar{x}_i^j.\mathbf{0}$, if ${}^\bullet t_j(s_i) = 1$ and ${}^\bullet t_j(s_{i-h}) = 1$ for some $h > 0$ (i.e., s_i is not the leader of the synchronization on t_j);
- $x_{i+h}^j.\Pi_j$, if ${}^\bullet t_j(s_i) = 1$ and ${}^\bullet t_j(s_{i+h}) = 1$ for some $h > 0$ (i.e., s_i is the leader of the synchronization);
- $\bar{x}_i^j.\mathbf{0} + x_i^j.\Pi_j$, if ${}^\bullet t_j(s_i) = 2$ (i.e., in case of self-synchronization).

Finally, process Π_j is $\underbrace{C_1|\cdots|C_1}_{t_j^\bullet(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{t_j^\bullet(s_n)}$, meaning that $\Pi_j = \mathbf{0}$ if $t_j^\bullet = \emptyset$. □

Note that $\mathscr{T}_{FNC}(N(m_0))$ is an (output-closed) FNC process: in fact, restriction occurs only at the top level, there are finitely many constants involved (one for each place) and each constant has a body that starts sequentially. Therefore, the following proposition holds by Proposition 6.20 and Corollary 6.3.

Proposition 6.23. *Given a finite CCS net $N(m_0) = (S,A,T,m_0)$, $Net(\mathscr{T}_{FNC}(N(m_0)))$ is a statically reduced, finite CCS net.* □

Moreover, $\mathscr{T}_{FNC}(N(m_0))$ is output-closed, as $fn(\mathscr{T}_{FNC}(N(m_0))) \cap \overline{\mathscr{L}} = \emptyset$; in fact, any output occurring in this term is of the form $\overline{x}_i^j$ for suitable i and j, and such a name is bound.

Example 6.11. The net $N(m_0)$ in Figure 6.1, with $m_0 = s_1 \oplus s_2 \oplus s_3$, has six places $\{s_1,s_2,s_3,s_4,s_5,s_6\}$ and five transitions $\{(s_1 \oplus s_2, \tau, s_4 \oplus s_5), (s_3 \oplus s_4, \tau, s_6), (s_5 \oplus s_6, \tau, \emptyset), (s_5, c, \emptyset), (s_6, \overline{c}, \emptyset)\}$. The FNC process $\mathscr{T}_{FNC}(N(m_0))$ is $(\nu L)(C_1 \,|\, C_2 \,|\, C_3)$, where $L = \{x_1^1, \ldots, x_6^1, x_1^2, \ldots, x_6^2, \ldots, x_1^5, \ldots, x_6^5\}$ and

$$\begin{array}{ll}
C_1 \doteq x_2^1.(C_4 \,|\, C_5) + \mathbf{0} + \mathbf{0} + \mathbf{0} + \mathbf{0}, & C_2 \doteq \overline{x}_2^1.\mathbf{0} + \mathbf{0} + \mathbf{0} + \mathbf{0} + \mathbf{0}, \\
C_3 \doteq \mathbf{0} + x_4^2.C_6 + \mathbf{0} + \mathbf{0} + \mathbf{0}, & C_4 \doteq \mathbf{0} + \overline{x}_4^2.\mathbf{0} + \mathbf{0} + \mathbf{0} + \mathbf{0}, \\
C_5 \doteq \mathbf{0} + \mathbf{0} + x_6^3.\mathbf{0} + c.\mathbf{0} + \mathbf{0}, & C_6 \doteq \mathbf{0} + \mathbf{0} + \overline{x}_6^3.\mathbf{0} + \mathbf{0} + \overline{c}.\mathbf{0},
\end{array}$$

where only three bound names are actually used, so that the term can be largely simplified, by omitting the unused bound names and also all the useless $\mathbf{0}$-summands in the body of the constants; nonetheless, the resulting simplified process is still rather different from the process $(\nu a)((\nu b)(a.b.\mathbf{0} \,|\, \overline{a}.c.\mathbf{0} \,|\, \overline{b}.\overline{c}.\mathbf{0}))$, of which $N(m_0)$ is the semantics, as described in Example 6.10. □

Example 6.12. The net $N(m_0)$ in Figure 3.3, with $m_0 = P \oplus C$, has four places $\{P, C, P', C'\}$, which are ordered as listed (e.g., P' is the third place), and three transitions $\{(P, prod, P \oplus P'), (C \oplus P', \tau, C'), (C', cons, C)\}$. The process $\mathscr{T}_{FNC}(N(m_0))$ is $(\nu L)(C_1 \,|\, C_2)$, where $L = \{x_1^1, \ldots, x_4^1, x_1^2, \ldots, x_4^2, x_1^3, \ldots, x_4^3\}$ and

$$\begin{array}{ll}
C_1 \doteq prod.(C_1 \,|\, C_3) + \mathbf{0} + \mathbf{0}, & C_2 \doteq \mathbf{0} + x_3^2.C_4 + \mathbf{0}, \\
C_3 \doteq \mathbf{0} + \overline{x}_3^2.\mathbf{0} + \mathbf{0}, & C_4 \doteq \mathbf{0} + \mathbf{0} + cons.C_2,
\end{array}$$

where only one bound name is actually used; by removing all the unused bound names and the useless $\mathbf{0}$-summands, the resulting FNC process is very similar to $(\nu send)(P \,|\, C)$ — where $P \doteq prod.(P \,|\, P')$, $C \doteq send.C'$, $P' \doteq \overline{send}.\mathbf{0}$ and $C' \doteq cons.C$ — whose semantics is exactly the net in Figure 3.3. □

Now we are ready to state our main result, the so-called *representability theorem*.

Theorem 6.8. (Representability theorem 4) *Let $N(m_0) = (S,A,T,m_0)$ be a statically reduced, finite CCS net system with $A \subseteq \mathscr{L} \cup \{\tau\}$, and let $p = \mathscr{T}_{FNC}(N(m_0))$. Then, $Net(p)$ is isomorphic to $N(m_0)$.*

Proof. Let $N(m_0) = (S,A,T,m_0)$ be a reduced, finite CCS net, with $S = \{s_1, \ldots, s_n\}$, $A \subseteq \mathscr{L} \cup \{\tau\}$, $T = \{t_1, \ldots, t_k\}$ and $l(t_j) = \mu_j$ for $j = 1, \ldots, k$. The associated FNC process is

$$\mathscr{T}_{FNC}(N(m_0)) = (\nu L)(\underbrace{C_1 | \cdots | C_1}_{m_0(s_1)} | \cdots | \underbrace{C_n | \cdots | C_n}_{m_0(s_n)}),$$

where $L = \{x_1^1, \ldots, x_n^1, x_1^2, \ldots, x_n^2, \ldots, x_1^k, \ldots, x_n^k\}$, $L \cap A = \emptyset$, *and for each place* s_i *we have a corresponding constant* $C_i \doteq \sum_{j=1}^k c_i^j$, *defined as in Definition 6.12. For notational convenience,* $\sum_{j=1}^k c_i^j$ *is denoted by* p_i, *i.e.,* $C_i \doteq p_i$*; for the same reason, we use* p *to denote* $\mathscr{T}_{FNC}(N(m_0))$.

Let $\theta = \{L'/L\}$ *be a substitution that maps each bound name* x_i^j *to its corresponding restricted name* $x_i'^j$ *in* $\mathscr{L}'$*, for* $i = 1, \ldots, n$ *and* $j = 1, \ldots, k$*. Let* $Net(p) = (S', A', T', m_0')$*. Then,* $m_0' = dec(p)$ *is the multiset*

$$dec((\nu L)(\underbrace{C_1|\cdots|C_1}_{m_0(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m_0(s_n)})) = dec(\underbrace{C_1|\cdots|C_1}_{m_0(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m_0(s_n)})\theta =$$
$$m_0(s_1)\cdot C_1\theta \oplus \cdots \oplus m_0(s_n)\cdot C_n\theta$$

where $C_i\theta$ *gives rise to the new constant* $C_{i,\theta} \doteq p_i\theta$*. Hence, the initial places are all of the form* $C_i\theta$*, where such a place is present in* m_0' *only if* $m_0(s_i) > 0$.

Note that, by Definition 6.12, any transition $t' \in T'$ *with* ${}^\bullet t' \subseteq dec(p)$ *is such that, for some suitable* j, $t'^\bullet = dec(\Pi_j)\theta$*, and so equal to* $k_1 \cdot C_1\theta \oplus k_2 \cdot C_2\theta \oplus \ldots \oplus k_n \cdot C_n\theta$ *for suitable* $k_i \geq 0$, $i = 1, \ldots, n$*; by iterating this observation, each transition in* T' *has a post-set of the form* $dec(\Pi_j)\theta$*, for some suitable* $j = 1, \ldots, k$*. Hence, each statically reachable place* s_i' *in* S' *is of the form* $C_i\theta$*. Moreover, by Proposition 6.23,* $Net(p)$ *is statically reduced, implying that all the* $C_i\theta$*'s are statically reachable, for* $i = 1, \ldots, n$*. Hence, there is a bijection* $f : S \to S'$ *defined by* $f(s_i) = s_i' = C_i\theta$*, which is the natural candidate isomorphism function. To prove that* f *is an isomorphism, we have to prove that*

1. $f(m_0) = m_0'$,
2. $t = (m, \mu, m') \in T$ *implies* $f(t) = (f(m), \mu, f(m')) \in T'$*, and*
3. $t' = (m_1', \mu, m_2') \in T'$ *implies there exists* $t = (m_1, \mu, m_2) \in T$ *such that* $f(t) = t'$*, i.e.,* $f(m_1) = m_1'$ *and* $f(m_2) = m_2'$.

From items 2) and 3) above, it follows that $A = A'$.

Proof of *1: Let* $m_0 = k_1 \cdot s_1 \oplus k_2 \cdot s_2 \oplus \cdots \oplus k_n \cdot s_n$*, where* $k_i = m_0(s_i) \geq 0$ *for* $i = 1, \ldots, n$*. The mapping via* f *of the initial marking* m_0 *is*

$$f(m_0) = k_1 \cdot f(s_1) \oplus \cdots \oplus k_n \cdot f(s_n) = k_1 \cdot C_1\theta \oplus \cdots \oplus k_n \cdot C_n\theta$$
$$= dec(\underbrace{C_1|\cdots|C_1}_{k_1\, times}|\cdots|\underbrace{C_n|\cdots|C_n}_{k_n\, times})\theta = dec(p) = m_0'.$$

Proof of 2*: We prove that, for* $j = 1, \ldots, k$*, if* $t_j = (m, \mu_j, m') \in T$*, then* $t_j' = (f(m), \mu_j, f(m')) \in T'$*. As* $N(m_0)$ *is a CCS net,* t_j *must be of the form* $(\{s_i\}, \mu_j, m')$ *or of the form* $(\{s_i, s_{i+h}\}, \tau, m')$ *for some suitable indexes* $i > 0$, $h \geq 0$ *and multiset* m'*. Hence,* $f({}^\bullet t_j)$ *is equal to* $\{C_i\theta\}$*, or to* $\{C_i\theta, C_{i+h}\theta\}$ *with* $h > 0$*, or to* $\{C_i\theta, C_i\theta\}$*. Let us examine the three cases separately.*

- *$f({}^\bullet t_j) = \{C_i\theta\}$: According to Definition 6.12, for $C_i = p_i$, we have in p_i a summand $c_i^j = \mu_j.\Pi_j$, with $\Pi_j = \underbrace{C_1|\cdots|C_1}_{m'(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m'(s_n)}$. Therefore, not only is $\{c_i^j\} \xrightarrow{\mu_j} dec(\Pi_j)$ derivable by axiom (pref), but also $\{p_i\} \xrightarrow{\mu_j} dec(\Pi_j)$ by rules (sum$_1$) and (sum$_2$), and so $\{C_i\} \xrightarrow{\mu_j} dec(\Pi_j)$, by rule (cons). Hence, by Proposition 6.16, $\{C_i\theta\} \xrightarrow{\mu_j} dec(\Pi_j)\theta$ is derivable, as $\mu_j \notin L$. In conclusion, starting from transition $t_j = (\{s_i\}, \mu_j, m')$, we have shown that transition $t'_j = (\{C_i\theta\}, \mu_j, dec(\Pi_j)\theta)$, with $f(t_j^\bullet) = dec(\Pi_j)\theta$, belongs to T', as required.*
- *$f({}^\bullet t_j) = \{C_i\theta, C_{i+h}\theta\}$ with $h > 0$ and $l(t_j) = \tau$: By Definition 6.12, for $C_i = p_i$, we have in p_i a summand $c_i^j = x_{i+h}^j.\Pi_j$, with $\Pi_j = \underbrace{C_1|\cdots|C_1}_{m'(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m'(s_n)}$, while for $C_{i+h} = p_{i+h}$, we have in p_{i+h} a summand $c_{i+h}^j = \overline{x}_{i+h}^j.\mathbf{0}$. Therefore, not only is $\{c_i^j\} \xrightarrow{x_{i+h}^j} dec(\Pi_j)$ derivable by axiom (pref), but also, by rules (sum$_1$) and (sum$_2$), $\{p_i\} \xrightarrow{x_{i+h}^j} dec(\Pi_j)$ is derivable, and so $\{C_i\} \xrightarrow{x_{i+h}^j} dec(\Pi_j)$ by rule (cons). Similarly, not only is $\{c_{i+h}^j\} \xrightarrow{\overline{x}_{i+h}^j} \emptyset$ derivable by axiom (pref), but also, by rules (sum$_1$) and (sum$_2$), $\{p_{i+h}\} \xrightarrow{\overline{x}_{i+h}^j} \emptyset$ is derivable, and so $\{C_{i+h}\} \xrightarrow{\overline{x}_{i+h}^j} \emptyset$ by rule (cons). Finally, also $\{C_i, C_{i+h}\} \xrightarrow{\tau} dec(\Pi_j)$ is derivable by rule (com), and so $\{C_i\theta, C_{i+h}\theta\} \xrightarrow{\tau} dec(\Pi_j)\theta$ by Proposition 6.16. In conclusion, starting from transition $t_j = (\{s_i, s_{i+h}\}, \tau, m')$, we have shown that transition $t'_j = (\{C_i\theta, C_{i+h}\theta\}, \tau, dec(\Pi_j)\theta)$, with $f(t_j^\bullet) = dec(\Pi_j)\theta$, belongs to T', as required.*
- *$f({}^\bullet t_j) = \{C_i\theta, C_i\theta\}$ with $l(t_j) = \tau$: According to Definition 6.12, for $C_i = p_i$, we have in p_i a summand $c_i^j = \overline{x}_i^j.\mathbf{0} + x_i^j.\Pi_j$, with $\Pi_j = \underbrace{C_1|\cdots|C_1}_{m'(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m'(s_n)}$. Therefore, not only is $\{x_i^j.\Pi_j\} \xrightarrow{x_i^j} dec(\Pi_j)$ derivable by axiom (pref), but also $\{c_i^j\} \xrightarrow{x_i^j} dec(\Pi_j)$ is derivable by rule (sum$_2$), $\{p_i\} \xrightarrow{x_i^j} dec(\Pi_j)$ by rules (sum$_1$) and (sum$_2$), and so $\{C_i\} \xrightarrow{x_i^j} dec(\Pi_j)$ is derivable by rule (cons). Similarly, not only is $\{\overline{x}_i^j.\mathbf{0}\} \xrightarrow{\overline{x}_i^j} \emptyset$ derivable by axiom (pref), but also $\{c_i^j\} \xrightarrow{\overline{x}_i^j} \emptyset$ is derivable by rule (sum$_1$), $\{p_i\} \xrightarrow{\overline{x}_i^j} \emptyset$ by rules (sum$_1$) and (sum$_2$), and so $\{C_i\} \xrightarrow{\overline{x}_i^j} \emptyset$ is derivable by rule (cons). Finally, by rule (com), also $\{C_i, C_i\} \xrightarrow{\tau} dec(\Pi_j)$, and so $\{C_i\theta, C_i\theta\} \xrightarrow{\tau} dec(\Pi_j)\theta$ is derivable by Proposition 6.16. In conclusion, starting from transition $t_j = (\{s_i, s_i\}, \tau, m')$, we have shown that transition $t'_j = (\{C_i\theta, C_i\theta\}, \tau, dec(\Pi_j)\theta)$, with $f(t_j^\bullet) = dec(\Pi_j)\theta$, belongs to T', as required.*

Proof of *3: We prove that if $t'_j = (m'_1, \mu_j, m'_2) \in T'$, then there exists a transition $t_j = (m_1, \mu_j, m_2) \in T$ such that $f(m_1) = m'_1$ and $f(m_2) = m'_2$. This is proved by case* *analysis on the shape of the marking m'_1, which can be $\{C_i\theta\}$, or $\{C_i\theta, C_{i+h}\theta\}$, with $h > 0$ and $\mu_j = \tau$, or $\{C_i\theta, C_i\theta\}$, with $\mu_j = \tau$, for some $i = 1, \ldots, n$.*

- *If $m'_1 = \{C_i\theta\}$ for some $i = 1, \ldots, n$, then $t'_j = \{C_i\theta\} \xrightarrow{\mu_j} m'_2$. By Proposition 6.16, also transition $\{C_i\} \xrightarrow{\mu_j} m''_2$ is derivable, with $m''_2\theta = m'_2$. According to Definition 6.12, such a transition is derivable by the rules only if among the many summands composing p_i, there exists a summand $c^j_i = \mu_j.\Pi_j$ with $dec(\Pi_j) = m''_2$, which is possible only if in $N(m_0)$ we have a transition t_j with ${}^\bullet t_j = \{s_i\}$, $f(\{s_i\}) = \{C_i\theta\}$, $f(t^\bullet_j) = dec(\Pi_j)\theta = m'_2$ and $l(t_j) = \mu_j$, as required.*
- *If $m'_1 = \{C_i\theta, C_{i+h}\theta\}$, with $h > 0$ and $\mu_j = \tau$, then $t'_j = \{C_i\theta, C_{i+h}\theta\} \xrightarrow{\tau} m'_2$. By Proposition 6.16, also transition $\{C_i, C_{i+h}\} \xrightarrow{\tau} m''_2$ is derivable, with $m''_2\theta = m'_2$. According to Definition 6.12, such a transition is derivable by the rules only if among the many summands composing p_i, there exists a summand $c^j_i = x^j_{i+h}.\Pi_j$, and among the many summands of p_{i+h} there exists a summand $c^j_{i+h} = \overline{x}^j_{i+h}.\mathbf{0}$. These summands are present only if in $N(m_0)$ we have a transition t_j with ${}^\bullet t_j = \{s_i, s_{i+h}\}$, $f(\{s_i\}) = \{C_i\theta\}$, $f(\{s_{i+h}\}) = \{C_{i+h}\theta\}$, $f(t^\bullet_j) = dec(\Pi_j)\theta = m'_2$ and $l(t_j) = \tau$, as required.*
- *If $m'_1 = \{C_i\theta, C_i\theta\}$, with $\mu_j = \tau$, then $t'_j = \{C_i\theta, C_i\theta\} \xrightarrow{\tau} m'_2$. By Proposition 6.16, also transition $\{C_i, C_i\} \xrightarrow{\tau} m''_2$ is derivable, with $m''_2\theta = m'_2$. According to Definition 6.12, such a transition is derivable by the rules only if among the many summands composing p_i, there exists a summand $c^j_i = \overline{x}^j_i.\mathbf{0} + x^j_i.\Pi_j$. This summand is present only if in $N(m_0)$ we have a transition t_j with ${}^\bullet t_j = \{s_i, s_i\}$, $f(\{s_i\}) = \{C_i\theta\}$, $f(t^\bullet_j) = dec(\Pi_j)\theta = m'_2$ and $l(t_j) = \tau$, as required.*

No further cases are possible, because a transition $t'_j = (m'_1, \mu_j, m'_2) \in T'$ must be derivable by the rules and Proposition 6.17 ensures that the pre-set of a transition can only be of the forms listed above. □

Example 6.13. Function $\mathscr{T}_{FNC}(-)$ can be applied to any finite CCS net $N(m_0)$. However, if $N(m_0)$ is not *statically reduced*, the representability theorem does not hold. Not surprisingly, the same net discussed in Example 4.3 for the case of SFM shows the problem also in this case. The net $N(\{s_1\}) = (\{s_1, s_2\}, \{a\}, \{(s_1, a, \emptyset), (s_2, a, \emptyset)\}, \{s_1\})$ is not statically reduced because place s_2 is not statically reachable from the initial marking. The FNC term $\mathscr{T}_{FNC}(N(\{s_1\}))$ would be $(\nu L)(C_1)$, where $L = \{x^1_1, x^1_2, x^2_1, x^2_2\}$ and

$$C_1 \doteq a.\mathbf{0} + \mathbf{0}, \qquad C_2 \doteq \mathbf{0} + a.\mathbf{0},$$

but now $Net(\mathscr{T}_{FNC}(N(\{s_1\})))$ is the net $(\{C_1\theta\}, \{a\}, \{(C_1\theta, a, \emptyset)\}, \{C_1\theta\})$, with $\theta = \{L'/L\}$, which has one place and one transition only, i.e., it is isomorphic to the subnet of $N(\{s_1\})$ statically reachable from the initial marking $\{s_1\}$. □

Remark 6.6. **(Extending the approach to structured actions)** In the classic definition of labeled P/T Petri nets (see, e.g., [Pet81, Rei85, DR98]), the transition labeling is given with actions taken from a set A of *unstructured* actions; hence, our assumption that $A \subseteq \mathscr{L} \cup \{\tau\}$ is in analogy with this tradition.

However, if we want to be more generous and consider *communicating* Petri nets labeled over the set $Act = \mathscr{L} \cup \overline{\mathscr{L}} \cup \{\tau\}$ of *structured* actions and co-actions, the extension of the representability theorem to this larger class of nets is not trivial. First of all, we note that the translation in Definition 6.12 is no longer accurate; consider the Petri net $N(\{s_1,s_2\}) = (\{s_1,s_2\},\{a,\bar{a}\},\{(s_1,a,\emptyset),(s_2,\bar{a},\emptyset)\},\{s_1,s_2\})$, then $\mathscr{T}_{FNC}(N(\{s_1,s_2\}))$ is $(\nu L)(C_1 \,|\, C_2)$, with $L = \{x_1^1,x_2^1,x_1^2,x_2^2\}$ and

$$C_1 \doteq a.\mathbf{0}+\mathbf{0}, \qquad C_2 \doteq \mathbf{0}+\bar{a}.\mathbf{0},$$

but now $Net(\mathscr{T}_{FNC}(N(\{s_1,s_2\})))$ contains also an additional synchronization transition $(\{C_1\theta,C_2\theta\},\tau,\emptyset)$ — where $\theta = \{L'/L\}$ — which has no counterpart in the original net $N(\{s_1,s_2\})$. A possible solution to this more general problem is outlined in Section 9.1. □

6.6 Soundness

In this section, we want to prove that the operational net semantics preserves the step semantics of Section 6.3. More precisely, we want to show that any FNC process p, in the STS semantics, is step bisimilar to $dec(p)$, in the step marking graph $SMG(Net(p))$.

Lemma 6.7. *For any $p \in \mathscr{P}_{FNC}$, any $G \in \mathscr{M}_{fin}(T_{FNC})$ and any $M \in \mathscr{M}_{fin}(Act)$, if $dec(p)[G\rangle m$ and $MSync(l(G),M)$, then there exists G' such that $dec(p)[G'\rangle m$ and $l(G') = M$.*

Proof. *By induction on the size of G. If $|G| = 1$, then only $MSync(l(G),l(G))$ holds and the thesis follows trivially by taking $G' = G$. If $|G| = n > 1$, then two subcases are in order: either $\{a,\overline{a}\} \not\subseteq l(G)$ for any $a \in \mathscr{L}$, or there exists an action a such that $\{a,\overline{a}\} \subseteq l(G)$. In the former case, only $MSync(l(G),l(G))$ holds and the thesis follows trivially. In the latter case, besides the trivial case $MSync(l(G),l(G))$, we also have to investigate the following: take two transitions t_1 and t_2 in G such that $l(t_1) = a$ and $l(t_2) = \overline{a}$, for some $a \in \mathscr{L}$. Then, take the step $\overline{G} = (G \setminus \{t_1,t_2\}) \cup \{t_1 \,|\, t_2\}$, where transition $t_1 \,|\, t_2 = ({}^\bullet t_1 \oplus {}^\bullet t_2, \tau, t_1^\bullet \oplus t_2^\bullet)$ is derived by rule (com) with t_1 and t_2 as premises.*[1] *Clearly, also $dec(p)[\overline{G}\rangle m$ and $|\overline{G}| < n$, so that induction can be applied to conclude that for any M such that $MSync(l(\overline{G}),M)$, there exists G' such that $dec(p)[G'\rangle m$ and $l(G') = M$. Note that also $MSync(l(G),M)$ holds, because $MSync(l(\overline{G}),M)$, so that the thesis follows trivially for the chosen $\overline{G}$. Since this argument can be repeated for any pair of transitions t_1 and t_2 in G such that $l(t_1) = a$ and $l(t_2) = \overline{a}$, for some $a \in \mathscr{L}$ (hence, for any $\overline{G}$ of the form above), we can conclude that the thesis holds: if $dec(p)[G\rangle m$ and $MSync(l(G),M)$, then there exists G' such that $dec(p)[G'\rangle m$ and $l(G') = M$.* □

Proposition 6.24. *For any $p \in \mathscr{P}_{FNC}$, if $p \xrightarrow{M}_s p'$, then there exists a step G, with $l(G) = M$, such that $dec(p)[G\rangle dec(p')$.*

[1] Of course, $ad({}^\bullet t_1 \oplus {}^\bullet t_2)$ holds because ${}^\bullet t_1 \oplus {}^\bullet t_2 \subseteq dec(p)$, which is admissible.

Proof. *By induction on the proof of $p \xrightarrow{M}_s p'$. The base case is axiom(Pref^s): $\mu.p \xrightarrow{\{\mu\}}_s p$. As $dec(\mu.p) = \{\mu.p\}$, by axiom (pref) of Table 6.9, the net transition $t = \{\mu.p\} \xrightarrow{\mu} dec(p)$ is derivable, and so the singleton step $dec(\mu.p)[\{t\}\rangle dec(p)$ is also derivable, as required. If the last rule used in deriving $p \xrightarrow{M}_s p'$ is (Cons^s), then $M = \{\mu\}$ for some suitable μ, and $p = C$ for some constant C, with $C \doteq q$. The premise of the rule is $q \xrightarrow{\{\mu\}}_s p'$, and by induction we know that $dec(q)[\{t'\}\rangle dec(p')$ with $l(t') = \mu$. Since q is sequential, this step is possible only if $t' = \{q\} \xrightarrow{\mu} dec(p')$ is derivable by the rules in Table 6.9; hence, by rule (cons), also the net transition $t = \{C\} \xrightarrow{\mu} dec(p')$ is derivable, and so the singleton step $dec(C)[\{t\}\rangle dec(p')$ is also derivable, as required. The two cases when rule (Sum$^s_1$) or rule (Sum$^s_2$) is the last rule used in deriving $p \xrightarrow{M}_s p'$ are similar to the above, hence omitted. If the last rule used in deriving $p \xrightarrow{M}_s p'$ is (Par$^s_1$), then $p = p_1 \,|\, p_2$, $p' = p'_1 \,|\, p_2$ and the premise transition is $p_1 \xrightarrow{M}_s p'_1$. By induction, we know that $dec(p_1)[G\rangle dec(p'_1)$ with $l(G) = M$. Then, also $dec(p_1 \,|\, p_2) = dec(p_1) \oplus dec(p_2)[G\rangle dec(p'_1) \oplus dec(p_2) = dec(p'_1 \,|\, p_2)$, as required. The case of rule (Par$^s_2$) is symmetric, hence omitted. If the last rule used in deriving $p \xrightarrow{M}_s p'$ is (Com^s), then $p = p_1 \,|\, p_2$, $p' = p'_1 \,|\, p'_2$ and the two premise transitions are $p_1 \xrightarrow{M_1}_s p'_1$, $p_2 \xrightarrow{M_2}_s p'_2$, with $MSync(M_1 \oplus M_2, M)$. By induction, we have $dec(p_1)[G_1\rangle dec(p'_1)$, with $l(G_1) = M_1$, and $dec(p_2)[G_2\rangle dec(p'_2)$, with $l(G_2) = M_2$; therefore, $dec(p_1 \,|\, p_2) = dec(p_1) \oplus dec(p_2)[G_1 \oplus G_2\rangle dec(p'_1) \oplus dec(p'_2) = dec(p'_1 \,|\, p'_2)$. By Lemma 6.7, since $MSync(l(G_1 \oplus G_2), M)$, there must exist a step G' such that $dec(p_1 \,|\, p_2)[G'\rangle dec(p'_1 \,|\, p'_2)$ and $l(G') = M$, as required. If the last rule used in deriving $p \xrightarrow{M}_s p'$ is (Res^s), then $p = (\nu a)p_1$, for some $a \in \mathscr{L}$, and the premise transition is $p_1 \xrightarrow{M}_s p'_1$, with $a, \overline{a} \notin M$ and $p' = (\nu a)p'_1$. By induction, we know that $dec(p_1)[G\rangle dec(p'_1)$ with $l(G) = M$. Since for any $t \in G$, $t\{a'/a\} = {}^\bullet t\{a'/a\} \xrightarrow{\mu\{a'/a\}} t^\bullet\{a'/a\}$ is derivable by Proposition 6.16, where $\mu\{a'/a\} = \mu$ because $a, \overline{a} \notin M$, it follows that*

$$dec((\nu a)p_1) = dec(p_1)\{a'/a\}[G\{a'/a\}\rangle dec(p'_1)\{a'/a\} = dec((\nu a)p'_1)$$

is also a step, with $l(G\{a'/a\}) = M$, as required. □

Proposition 6.25. *For any $p \in \mathscr{P}_{FNC}$ and any $G \in \mathscr{M}_{fin}(T_{FNC})$, if $dec(p)[G\rangle m$, then there exists $p' \in \mathscr{P}_{FNC}$ such that $p \xrightarrow{M}_s p'$, with $l(G) = M$ and $dec(p') = m$.*

Proof. *By induction on the definition of $dec(p)$. If $p = \mathbf{0}$, then $dec(p) = \emptyset$; this case is vacuously true, as no transition is executable from $\emptyset$. If p is any other sequential process, then $dec(p) = \{p\}$. If $\{p\}[G\rangle m$, then G is a singleton $\{t\}$, with $t = (\{p\}, \mu, dec(p'))$ for some suitable μ and p', because, by Proposition 6.14, the post-set of any derivable net transition is admissible and, by Theorem 6.4, any admissible marking is complete. The proof of the net transition t can be mimicked by the corresponding rules in Table 6.6 to produce the transition $p \xrightarrow{\{\mu\}}_s p'$, as required.*

If $p = p_1 \,|\, p_2$, then $dec(p) = dec(p_1) \oplus dec(p_2)$. The following three sub-cases are possible:

$$\frac{}{GSync(G,G)} \qquad \frac{GSync(G\oplus\{t\},G') \quad t=({}^\bullet t_1\oplus{}^\bullet t_2,\tau,t_1^\bullet\oplus t_2^\bullet) \quad l(t_1)=\alpha, l(t_2)=\overline{\alpha}}{GSync(G\oplus\{t_1,t_2\},G')}$$

Table 6.10 Step transition synchronization relation

(i) $dec(p_1)[G\rangle m_1$ and $m=m_1\oplus dec(p_2)$: in such a case, by induction, we know that there exists p_1' such that $p_1 \xrightarrow{M}_s p_1'$, with $l(G)=M$ and $dec(p_1')=m_1$; therefore, by rule (Par$_1^s$), also $p_1\,|\,p_2 \xrightarrow{M}_s p_1'\,|\,p_2$ is derivable, with $dec(p_1'\,|\,p_2)=m$, as required.

(ii) $dec(p_2)[G\rangle m_2$ and $m=dec(p_1)\oplus m_2$: this is the symmetric case of the above, hence omitted.

(iii) there exist two steps G_1 and G_2 such that $dec(p_1)[G_1\rangle m_1$, $dec(p_2)[G_2\rangle m_2$, with $l(G_1)=M_1$, $l(G_2)=M_2$, $m=m_1\oplus m_2$ and $GSync(G_1\oplus G_2,G)$, where relation $GSync(-)$ of Table 6.10 mimics, on multisets of transitions, what relation $MSync(-)$ does on multisets of labels, so that $MSync(M_1\oplus M_2,M)$ holds, too. In this case, by induction, we know that there exist p_1' and p_2' such that $p_1 \xrightarrow{M_1}_s p_1'$, $p_2 \xrightarrow{M_2}_s p_2'$, with $dec(p_1')=m_1$ and $dec(p_2')=m_2$. By rule (Com^s) of Table 6.6, also transition $p_1\,|\,p_2 \xrightarrow{M}_s p_1'\,|\,p_2'$ is derivable, with $dec(p_1'\,|\,p_2')=m$, as required.

If $p=(\nu a)p_1$, for some $a\in\mathscr{L}$, then $dec(p)=dec(p_1)\{a'/a\}$. Therefore, G is of the form $\overline{G}\{a'/a\}$, by Proposition 6.16; it follows that $dec(p_1)[\overline{G}\rangle\overline{m}$, where $m=\overline{m}\{a'/a\}$ and $l(G)=l(\overline{G})=M$, because no net transition in G is labeled with the restricted action a', and so no action in $\overline{G}$ is labeled with action a. By induction, there exists p_1' such that $p_1 \xrightarrow{M}_s p_1'$, with $dec(p_1')=\overline{m}$. By rule (Res^s), also transition $(\nu a)p_1 \xrightarrow{M}_s (\nu a)p_1'$ is derivable, with $dec((\nu a)p_1')=dec(p_1')\{a'/a\}=\overline{m}\{a'/a\}=m$, as required. □

Theorem 6.9. *For any $p\in\mathscr{P}_{FNC}$, $p\sim_{step} dec(p)$.*

Proof. *The relation $R=\{(p,dec(p))\;\big|\;p\in\mathscr{P}_{FNC}\}$ is a bisimulation between the step transition system $\mathscr{C}_p^{step}=(\mathscr{P}_p^{step},sort(p)_s,\rightarrow_s^p,p)$, originating from the step semantics of Section 6.3, and the step marking graph $SMG(Net(p))$. Given $(p,dec(p))\in R$, if $p\xrightarrow{M}_s p'$, then by Proposition 6.24 we have that $dec(p)[G\rangle dec(p')$, with $l(G)=M$ and $(p',dec(p'))\in R$; conversely, if $dec(p)[G\rangle m$, with $l(G)=M$, then by Proposition 6.25 there exists $p'\in\mathscr{P}_{FNC}$ such that $p\xrightarrow{M}_s p'$ and $dec(p')=m$, so that $(p',dec(p'))\in R$, as required.* □

6.7 Denotational Net Semantics

The extension to FNC of the denotational semantics we have defined for SFM, CFM and BPP in the previous chapters is rather complex due to the addition of the syn-

$$
\begin{aligned}
\llbracket \mathbf{0} \rrbracket_I &= (\emptyset,\emptyset,\emptyset,\emptyset)\\
\llbracket \mu.p \rrbracket_I &= (S,A,T,\{\mu.p\}) \;\; \text{given } \llbracket p \rrbracket_I = (S',A',T',dec(p)) \text{ and where}\\
&\qquad S = \{\mu.p\} \cup S',\; t = (\{\mu.p\},\mu,dec(p))\\
&\qquad T = \{t\} \cup T' \cup (\{t\} \otimes T')\\
&\qquad A = \begin{cases} \{\mu\} \cup A' & \{t\} \otimes T' = \emptyset\\ \{\mu\} \cup A' \cup \{\tau\} & \text{otherwise} \end{cases}\\
\llbracket C \rrbracket_I &= (\{C\},\emptyset,\emptyset,\{C\}) \;\; \text{if } C \in I\\
\llbracket C \rrbracket_I &= (S,A,T,\{C\}) \quad \text{if } C \notin I, \text{ given } C \doteq p \text{ and } \llbracket p \rrbracket_{I \cup \{C\}} = (S',A',T',dec(p))\\
&\qquad S = \{C\} \cup S'',\; \text{where}\\
&\qquad S'' = \begin{cases} S' & \exists t \in T' .\, t^\bullet(p) > 0 \;\vee\; p = \mathbf{0},\\ S' \setminus \{p\} & \text{otherwise} \end{cases}\\
&\qquad A = A',\; T = T'' \cup T_1 \text{ where}\\
&\qquad T'' = \begin{cases} T' & \exists t \in T' .\, t^\bullet(p) > 0 \;\vee\; p = \mathbf{0},\\ T' \setminus \{t \in T' \mid {}^\bullet t(p) > 0\} & \text{otherwise} \end{cases}\\
&\qquad T_C = \{(n \cdot C \oplus m_1, \mu, m_2) \mid (n \cdot p \oplus m_1, \mu, m_2) \in T', n \geq 1\}\\
&\qquad T_1 = \begin{cases} \{(n \cdot C \oplus m_1, \mu, m_2) \in T_C \mid m_1(p) = 0\} & \text{if } p \notin S,\\ T_C & \text{otherwise} \end{cases}
\end{aligned}
$$

Table 6.11 Denotational net semantics — part I

chronization capability to the parallel operator and to the fact that the *static* enabling of transitions is no longer equivalent to the dynamic enabling. In particular, we have to consider the case of possible self-synchronization of two instances of a sequential choice process, as illustrated in Example 6.9, which adds a lot of complexity to the definition of some operators.

The correspondence with the operational semantics is strictly preserved for restriction-free processes: the operational net $Net(t)$ is exactly the denotational net $\llbracket t \rrbracket_\emptyset$ we are going to define. On the contrary, for a restricted process $p = (\nu L)t$, we have the weaker result that $Net(p)$ is exactly the *statically reachable subnet* of $\llbracket p \rrbracket_\emptyset$.

Table 6.11 shows the denotational semantics for the empty process $\mathbf{0}$, for the action prefixing operator and for the constants. The semantics definition $\llbracket - \rrbracket_I$ is parametrized by a set I of constants that have been already found while scanning p; such a set is initially empty and it is used to avoid looping on recursive constants. The definition is syntax driven and also the places of the constructed net are syntactic objects, i.e., (extended) sequential FNC processes. The definition for $\mathbf{0}$ is as usual: the associated net is empty. On the contrary, the semantics for the action prefixing operator is more involved than for the previous calculi: we have to consider the fact that the newly added transition $t = (\{\mu.p\}, \mu, dec(p))$ may be able to synchronize with the transitions that are in set T', because we have to consider also all the possible synchronizations that are statically enabled at the set $[\![dom(\{\mu.p\})\rangle$ of the places statically reachable from $\{\mu.p\}$. This set of additional transitions is denoted by $\{t\} \otimes T'$, where the auxiliary operation $- \otimes -$ of *synchronous product* of sets of transitions is defined as follows.

Definition 6.13. (Synchronous product of two sets of transitions) Given two sets T_1 and T_2 of transitions, we define their *synchronous product* $T_1 \otimes T_2$ as the set $\{(m_1 \oplus m_2, \tau, m'_1 \oplus m'_2) \mid \exists \alpha \in \mathscr{L} \cup \overline{\mathscr{L}}.(m_1, \alpha, m'_1) \in T_1 \wedge (m_2, \overline{\alpha}, m'_2) \in T_2\}$. □

An example is useful to clarify the definition for the action prefixing operator.

Example 6.14. Let us consider the sequential FNC process $\overline{a}.a.b.\mathbf{0}$. Since $[\![\mathbf{0}]\!]_I = (\emptyset, \emptyset, \emptyset, \emptyset)$, the net for $b.\mathbf{0}$ is

$$[\![b.\mathbf{0}]\!]_\emptyset = (\{b.\mathbf{0}\}, \{b\}, \{(\{b.\mathbf{0}\}, b, \emptyset)\}, \{b.\mathbf{0}\}),$$

while the net for $a.b.\mathbf{0}$ is

$$[\![a.b.\mathbf{0}]\!]_\emptyset = (\{a.b.\mathbf{0}, b.\mathbf{0}\}, \{a, b\}, \{(\{a.b.\mathbf{0}\}, a, \{b.\mathbf{0}\}), (\{b.\mathbf{0}\}, b, \emptyset)\}, \{a.b.\mathbf{0}\}).$$

Now, the net for $\overline{a}.a.b.\mathbf{0}$ is $[\![\overline{a}.a.b.\mathbf{0}]\!]_\emptyset = (S, A, T, m_0)$, where

$$\begin{aligned}
S &= \{\overline{a}.a.b.\mathbf{0}, a.b.\mathbf{0}, b.\mathbf{0}\},\\
A &= \{a, \overline{a}, b, \tau\},\\
T &= \{t\} \cup T' \cup \{t\} \otimes T',\\
t &= (\{\overline{a}.a.b.\mathbf{0}\}, \overline{a}, \{a.b.\mathbf{0}\}),\\
T' &= \{(\{a.b.\mathbf{0}\}, a, \{b.\mathbf{0}\}), (\{b.\mathbf{0}\}, b, \emptyset)\},\\
\{t\} \otimes T' &= \{(\{\overline{a}.a.b.\mathbf{0}, a.b.\mathbf{0}\}, \tau, \{a.b.\mathbf{0}, b.\mathbf{0}\}),\\
m_0 &= \{\overline{a}.a.b.\mathbf{0}\},
\end{aligned}$$

which is depicted in Figure 6.2(a). Note that the synchronized transition is not executable from the initial marking m_0; however, it is statically enabled at $[\![dom(m_0)\rangle = S$, and even dynamically enabled at the marking $\{\overline{a}.a.b.\mathbf{0}, a.b.\mathbf{0}\}$, which is reachable from $2 \cdot m_0$. □

The denotational definition for the constant C, with $C \doteq p$ and $C \notin I$, is also a bit more involved than for the previous calculi: $[\![C]\!]_I$ is obtained by suitably manipulating the net for $[\![p]\!]_{I \cup \{C\}}$, with particular care devoted (i) to taking the place for p (if $p \neq \mathbf{0}$) — and the transitions having p in their pre-set — in the resulting net only in case there is some transition t producing at least one token in place p (definition of sets S'' and T''); and (ii) to generating the new transitions with pre-set $C \oplus m_1$ (with $|m_1| \leq 1$), or $\{C, C\}$ in correspondence with the analogous transitions with pre-set $p \oplus m_1$ or $\{p, p\}$ (definition of the set T_1).[2] In particular, the new transitions with pre-set $\{C, C\}$ are only statically enabled at the initial marking $\{C\}$; such transitions

[2] The definition of the set T_1 in Table 6.11 is apparently more general, but, since synchronization is strictly binary, one can prove that it is always the case that $1 \leq n \leq 2$, $|m_1| \leq 1$ and that if $n = 2$, then $m_1 = \emptyset$.

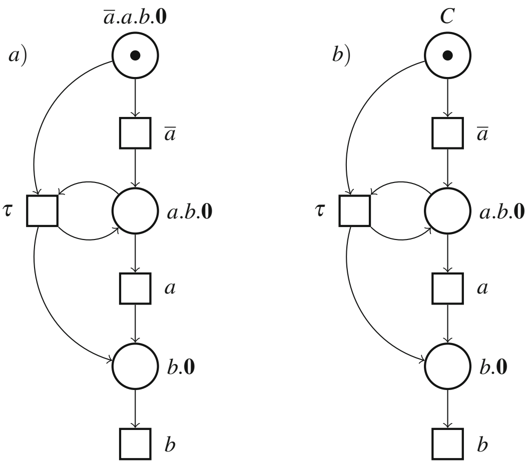


Fig. 6.2 The finite CCS nets for $\overline{a}.a.b.\mathbf{0}$ in (a), and for $C \doteq \overline{a}.a.b.\mathbf{0}$ in (b)

can be generated in case p is a process of the form $p_1 + p_2$, where different alternatives offer complementary actions, as illustrated in the following Examples 6.17 and 6.18. Finally, note that a transition in T_1 may have pre-set $\{C, p\}$, in case p can self-synchronize and p belongs to S, as illustrated in the following Example 6.19.

Example 6.15. Let us consider the constant $C \doteq \overline{a}.a.b.\mathbf{0}$. The net $[\![\overline{a}.a.b.\mathbf{0}]\!]_{\{C\}}$ is defined in Example 6.14, because the parameter $\{C\}$ is inessential. According to the semantic definition in Table 6.11, the net $[\![C]\!]_\emptyset$ is $(S, A, T, \{C\})$, where

$$\begin{aligned}
S &= \{C, a.b.\mathbf{0}, b.\mathbf{0}\},\\
A &= \{a, \overline{a}, b, \tau\},\\
T &= T_1 \cup T'',\\
T_1 &= \{(\{C\}, \overline{a}, \{a.b.\mathbf{0}\}), (\{C, a.b.\mathbf{0}\}, \tau, \{a.b.\mathbf{0}, b.\mathbf{0}\})\},\\
T'' &= \{(\{a.b.\mathbf{0}\}, a, \{b.\mathbf{0}\}), (\{b.\mathbf{0}\}, b, \emptyset)\},
\end{aligned}$$

which is depicted in Figure 6.2(b). □

Table 6.12 shows the denotational semantics for the parallel operator and for the choice operator. The case of parallel composition is not too difficult: the net for $p_1 \,|\, p_2$ is obtained by putting together the two nets of p_1 and p_2, by taking the multiset union of the respective initial markings, and by adding all the τ-labeled transitions obtained by synchronizing one transition of p_1 with one transition of p_2, labeled with complementary actions (set $T_1 \otimes T_2$).

Example 6.16. Consider processes $p_1 = a.b.\mathbf{0}$ and $p_2 = \overline{a}.\overline{b}.\mathbf{0}$. The net for p_1 is

$$[\![a.b.\mathbf{0}]\!]_\emptyset = (\{a.b.\mathbf{0}, b.\mathbf{0}\}, \{a, b\}, \{(\{a.b.\mathbf{0}\}, a, \{b.\mathbf{0}\}), (\{b.\mathbf{0}\}, b, \emptyset)\}, \{a.b.\mathbf{0}\}),$$

while the denotational net for p_2 is

$$[\![\overline{a}.\overline{b}.\mathbf{0}]\!]_\emptyset = (\{\overline{a}.\overline{b}.\mathbf{0}, \overline{b}.\mathbf{0}\}, \{\overline{a}, \overline{b}\}, \{(\{\overline{a}.\overline{b}.\mathbf{0}\}, \overline{a}, \{\overline{b}.\mathbf{0}\}), (\{\overline{b}.\mathbf{0}\}, \overline{b}, \emptyset)\}, \{\overline{a}.\overline{b}.\mathbf{0}\}).$$

The synchronous product of the two sets of transitions is the set

$$[\![p_1 \,|\, p_2]\!]_I \;=\; (S,A,T,m_0) \;\text{ given } [\![p_i]\!]_I = (S_i,A_i,T_i,m_i) \;\text{ for } i=1,2, \text{ and where}$$

$$\begin{aligned}
&S = S_1 \cup S_2,\; m_0 = m_1 \oplus m_2,\; T = T_1 \cup T_2 \cup T_1 \otimes T_2,\\
&A = \begin{cases} A_1 \cup A_2 & \text{if } T_1 \otimes T_2 = \emptyset \\ A_1 \cup A_2 \cup \{\tau\} & \text{otherwise} \end{cases}
\end{aligned}$$

$$[\![p_1 + p_2]\!]_I \;=\; (S,A,T,\{p_1+p_2\}) \;\text{ given } [\![p_i]\!]_I = (S_i,A_i,T_i,dec(p_i)) \;\text{ for } i=1,2, \text{ and where}$$

$$\begin{aligned}
&S = \{p_1+p_2\} \cup S'_1 \cup S'_2, \text{ with, for } i=1,2,\\
&S'_i = \begin{cases} S_i & \exists t \in T_i \text{ such that } t^\bullet(p_i) > 0 \,\vee\, p_i = \mathbf{0}, \\ S_i \setminus \{p_i\} & \text{otherwise} \end{cases}\\
&T = T'_1 \cup T''_1 \cup T'_2 \cup T''_2 \cup (T'_1 \cup T''_1) \otimes (T'_2 \cup T''_2), \text{ with, for } i=1,2,\\
&T'_i = \begin{cases} T_i & \exists t \in T_i\,.\,t^\bullet(p_i) > 0 \,\vee\, p_i = \mathbf{0}, \\ T_i \setminus \{t \in T_i \;\mid\; {}^\bullet t(p_i) > 0\} & \text{otherwise} \end{cases}\\
&T_i^+ = \{(n \cdot (p_1+p_2) \oplus m_1, \mu, m_2) \;\mid\; (n \cdot p_i \oplus m_1, \mu, m_2) \in T_i, n \geq 1\}\\
&T''_i = \begin{cases} \{(n \cdot (p_1+p_2) \oplus m_1, \mu, m_2) \in T_i^+ \;\mid\; m_1(p_i) = 0\} & \text{if } p_i \not\in S \\ T_i^+ & \text{otherwise} \end{cases}\\
&A = \begin{cases} A_1 \cup A_2 & \text{if } (T'_1 \cup T''_1) \otimes (T'_2 \cup T''_2) = \emptyset \\ A_1 \cup A_2 \cup \{\tau\} & \text{otherwise} \end{cases}
\end{aligned}$$

Table 6.12 Denotational net semantics — part II

$\{(\{a.b.\mathbf{0}, \overline{a}.\overline{b}.\mathbf{0}\}, \tau, \{b.\mathbf{0}, \overline{b}.\mathbf{0}\}), (\{b.\mathbf{0}, \overline{b}.\mathbf{0}\}, \tau, \emptyset)\}$.
Hence, $[\![p_1 \,|\, p_2]\!]_\emptyset = [\![a.b.\mathbf{0} \,|\, \overline{a}.\overline{b}.\mathbf{0}]\!]_\emptyset = (S,A,T,m_0)$, where

$$\begin{aligned}
S &= \{a.b.\mathbf{0}, b.\mathbf{0}, \overline{a}.\overline{b}.\mathbf{0}, \overline{b}.\mathbf{0}\},\\
A &= \{a, \overline{a}, b, \overline{b}, \tau\},\\
T &= \{(\{a.b.\mathbf{0}\}, a, \{b.\mathbf{0}\}), (\{b.\mathbf{0}\}, b, \emptyset), (\{\overline{a}.\overline{b}.\mathbf{0}\}, \overline{a}, \{\overline{b}.\mathbf{0}\}), (\{\overline{b}.\mathbf{0}\}, \overline{b}, \emptyset),\\
&\qquad (\{a.b.\mathbf{0}, \overline{a}.\overline{b}.\mathbf{0}\}, \tau, \{b.\mathbf{0}, \overline{b}.\mathbf{0}\}), (\{b.\mathbf{0}, \overline{b}.\mathbf{0}\}, \tau, \emptyset)\},\\
m_0 &= \{a.b.\mathbf{0}, \overline{a}.\overline{b}.\mathbf{0}\},
\end{aligned}$$

which is outlined in Figure 6.3(a). □

The denotational semantics of $p_1 + p_2$ is, apparently, rather involved. As for the previous calculi, in order to include only strictly necessary places and transitions, the initial place p_1 (or p_2) of the subnet $[\![p_1]\!]_I$ (or $[\![p_2]\!]_I$) is to be kept in the net for $p_1 + p_2$ only if there exists a transition reaching place p_1 (or p_2) in $[\![p_1]\!]_I$ (or $[\![p_2]\!]_I$), otherwise p_1 (or p_2) can be safely removed in the new net (definition of sets S'_i and T'_i for $i = 1,2$). Differently from the previous calculi, as we are considering all the transitions that are *statically enabled* from the places *statically reachable* from $\{p_1 + p_2\}$, we have to add also all the transitions obtained by synchronizing one transition originating from the summand p_1 (set $T'_1 \cup T''_1$) with one transition originating from the summand p_2 (set $T'_2 \cup T''_2$), provided that they are labeled with complementary actions, as if p_1 and p_2 were composed by the parallel operator. Note

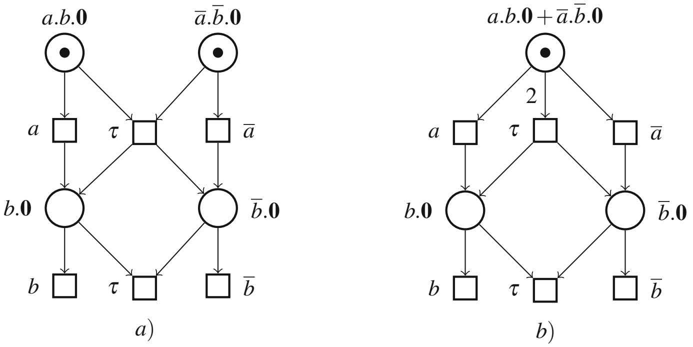

$a.b.\mathbf{0}$
$\overline{a}.\overline{b}.\mathbf{0}$
a
τ
$\overline{a}$
$b.\mathbf{0}$
$\overline{b}.\mathbf{0}$
b
τ
$\overline{b}$
a)
$a.b.\mathbf{0}+\overline{a}.\overline{b}.\mathbf{0}$
2
a
τ
$\overline{a}$
$b.\mathbf{0}$
$\overline{b}.\mathbf{0}$
b
τ
$\overline{b}$
b)

Fig. 6.3 The finite CCS nets for $a.b.\mathbf{0} \,|\, \overline{a}.\overline{b}.\mathbf{0}$ in (a), and for $a.b.\mathbf{0}+\overline{a}.\overline{b}.\mathbf{0}$ in (b)

that a transition in T_i'' may have pre-set $\{p_1+p_2, p_i\}$, when p_i can self-synchronize and $p_i \in S$. The following simple examples may help clarify the issue.

Example 6.17. Consider again processes $p_1 = a.b.\mathbf{0}$ and $p_2 = \overline{a}.\overline{b}.\mathbf{0}$, whose denotational nets $[\![p_1]\!]_\emptyset$ and $[\![p_2]\!]_\emptyset$ are described in Example 6.16. The denotational net for p_1+p_2 is $[\![a.b.\mathbf{0}+\overline{a}.\overline{b}.\mathbf{0}]\!]_\emptyset = (S,A,T,m_0)$, where

$$\begin{aligned}
S &= \{a.b.\mathbf{0}+\overline{a}.\overline{b}.\mathbf{0}, b.\mathbf{0}, \overline{b}.\mathbf{0}\},\\
A &= \{a, \overline{a}, b, \overline{b}, \tau\},\\
m_0 &= \{a.b.\mathbf{0}+\overline{a}.\overline{b}.\mathbf{0}\},\\
T &= T_1' \cup T_1'' \cup T_2' \cup T_2'' \cup (T_1' \cup T_1'') \otimes (T_2' \cup T_2''),\\
T_1' &= \{(\{b.\mathbf{0}\}, b, \emptyset)\},\\
T_1'' &= \{(\{a.b.\mathbf{0}+\overline{a}.\overline{b}.\mathbf{0}\}, a, \{b.\mathbf{0}\})\},\\
T_2' &= \{(\{\overline{b}.\mathbf{0}\}, \overline{b}, \emptyset)\},\\
T_2'' &= \{(\{a.b.\mathbf{0}+\overline{a}.\overline{b}.\mathbf{0}\}, \overline{a}, \{\overline{b}.\mathbf{0}\})\},\\
(T_1' \cup T_1'') &\otimes (T_2' \cup T_2'') =\\
&= \{(\{a.b.\mathbf{0}+\overline{a}.\overline{b}.\mathbf{0}, a.b.\mathbf{0}+\overline{a}.\overline{b}.\mathbf{0}\}, \tau, \{b.\mathbf{0}, \overline{b}.\mathbf{0}\}), (\{b.\mathbf{0}, \overline{b}.\mathbf{0}\}, \tau, \emptyset)\},
\end{aligned}$$

which is outlined in Figure 6.3(b). Note that the first synchronized transition $\{p_1+p_2, p_1+p_2\} \xrightarrow{\tau} \{b.\mathbf{0}, \overline{b}.\mathbf{0}\}$, which is a self-synchronization of two copies of the process p_1+p_2, is not dynamically enabled at the initial marking m_0, but it is statically enabled at $dom(m_0)$, and even dynamically enabled at the initial marking $2 \cdot m_0$. The other synchronized transition is enabled at the marking $\{b.\mathbf{0}, \overline{b}.\mathbf{0}\}$, reachable from $2 \cdot m_0$, hence showing the need to consider all the possible synchronized transitions between the two sets of transitions originating from p_1 and p_2. □

Example 6.18. Let us consider $C \doteq q_1 + q_2$, where $q_1 = a.b.\mathbf{0}+\overline{a}.\overline{b}.\mathbf{0}$ and $q_2 = \overline{a}.c.\mathbf{0}$. In the previous example, we have already computed the net $[\![a.b.\mathbf{0}+\overline{a}.\overline{b}.\mathbf{0}]\!]_\emptyset$, which is the same as $[\![a.b.\mathbf{0}+\overline{a}.\overline{b}.\mathbf{0}]\!]_{\{C\}}$, while the net $[\![\overline{a}.c.\mathbf{0}]\!]_{\{C\}}$ is

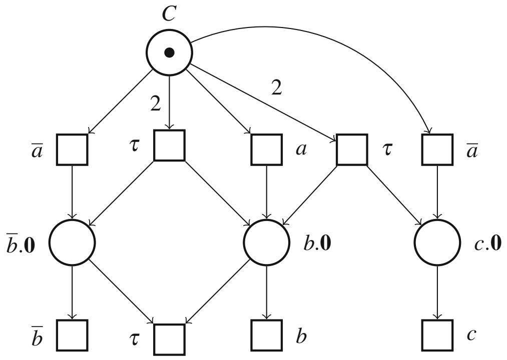


Fig. 6.4 The finite CCS net for $C \doteq (a.b.\mathbf{0} + \overline{a}.\overline{b}.\mathbf{0}) + \overline{a}.c.\mathbf{0}$

$$[\![\overline{a}.c.\mathbf{0}]\!]_{\{C\}} = (\{\overline{a}.c.\mathbf{0}, c.\mathbf{0}\}, \{\overline{a}, c\}, \{(\{\overline{a}.c.\mathbf{0}\}, \overline{a}, \{c.\mathbf{0}\}), (\{c.\mathbf{0}\}, c, \emptyset)\}, \{\overline{a}.c.\mathbf{0}\}).$$

The net $[\![(a.b.\mathbf{0} + \overline{a}.\overline{b}.\mathbf{0}) + \overline{a}.c.\mathbf{0}]\!]_{\{C\}}$ is (S, A, T, m_0), where

$$\begin{aligned}
S &= \{(a.b.\mathbf{0} + \overline{a}.\overline{b}.\mathbf{0}) + \overline{a}.c.\mathbf{0}, b.\mathbf{0}, \overline{b}.\mathbf{0}, c.\mathbf{0}\},\\
A &= \{a, \overline{a}, b, \overline{b}, c, \tau\},\\
m_0 &= \{(a.b.\mathbf{0} + \overline{a}.\overline{b}.\mathbf{0}) + \overline{a}.c.\mathbf{0}\},\\
T &= T_1' \cup T_1'' \cup T_2' \cup T_2'' \cup (T_1' \cup T_1'') \otimes (T_2' \cup T_2''),\\
T_1' &= \{(\{b.\mathbf{0}\}, b, \emptyset), (\{\overline{b}.\mathbf{0}\}, \overline{b}, \emptyset), (\{b.\mathbf{0}, \overline{b}.\mathbf{0}\}, \tau, \emptyset)\},\\
T_1'' &= \{(a.b.\mathbf{0} + \overline{a}.\overline{b}.\mathbf{0}) + \overline{a}.c.\mathbf{0}\}, a, \{b.\mathbf{0}\}), (\{(a.b.\mathbf{0} + \overline{a}.\overline{b}.\mathbf{0}) + \overline{a}.c.\mathbf{0}\}, \overline{a}, \{\overline{b}.\mathbf{0}\}),\\
&\quad (\{(a.b.\mathbf{0} + \overline{a}.\overline{b}.\mathbf{0}) + \overline{a}.c.\mathbf{0}, (a.b.\mathbf{0} + \overline{a}.\overline{b}.\mathbf{0}) + \overline{a}.c.\mathbf{0}\}, \tau, \{b.\mathbf{0}, \overline{b}.\mathbf{0}\})\},\\
T_2' &= \{(\{c.\mathbf{0}\}, c, \emptyset)\},\\
T_2'' &= \{(a.b.\mathbf{0} + \overline{a}.\overline{b}.\mathbf{0}) + \overline{a}.c.\mathbf{0}\}, \overline{a}, \{c.\mathbf{0}\})\},\\
(T_1' &\cup T_1'') \otimes (T_2' \cup T_2'') =\\
&\qquad = \{(\{(a.b.\mathbf{0} + \overline{a}.\overline{b}.\mathbf{0}) + \overline{a}.c.\mathbf{0}, (a.b.\mathbf{0} + \overline{a}.\overline{b}.\mathbf{0}) + \overline{a}.c.\mathbf{0}\}, \tau, \{b.\mathbf{0}, c.\mathbf{0}\})\}.
\end{aligned}$$

Finally, the net $[\![C]\!]_\emptyset$ is $(S', A, T', \{C\})$, where

$$\begin{aligned}
S' &= \{C, b.\mathbf{0}, \overline{b}.\mathbf{0}, c.\mathbf{0}\},\\
T' &= T_1 \cup T'',\\
T_1 &= \{(\{C\}, a, \{b.\mathbf{0}\}), (\{C\}, \overline{a}, \{\overline{b}.\mathbf{0}\}), (\{C\}, \overline{a}, \{c.\mathbf{0}\}), (\{C, C\}, \tau, \{b.\mathbf{0}, \overline{b}.\mathbf{0}\}),\\
&\quad (\{C, C\}, \tau, \{b.\mathbf{0}, c.\mathbf{0}\})\},\\
T'' &= \{(\{b.\mathbf{0}\}, b, \emptyset), (\{\overline{b}.\mathbf{0}\}, \overline{b}, \emptyset), (\{c.\mathbf{0}\}, c, \emptyset), (\{b.\mathbf{0}, \overline{b}.\mathbf{0}\}, \tau, \emptyset)\},
\end{aligned}$$

which is depicted in Figure 6.4. □

Example 6.19. Let us consider $C \doteq p$, where $p = a.A + \overline{a}.A$ and $A \doteq b.p$. The step-by-step construction is as follows. The net $[\![A]\!]_{\{A,C\}}$ is $(\{A\}, \emptyset, \emptyset, \{A\})$. Then,

$$\begin{aligned}
[\![a.A]\!]_{\{A,C\}} &= (\{a.A, A\}, \{a\}, \{(\{a.A\}, a, \{A\})\}, \{a.A\}),\\
[\![\overline{a}.A]\!]_{\{A,C\}} &= (\{\overline{a}.A, A\}, \{\overline{a}\}, \{(\{\overline{a}.A\}, \overline{a}, \{A\})\}, \{\overline{a}.A\}),
\end{aligned}$$

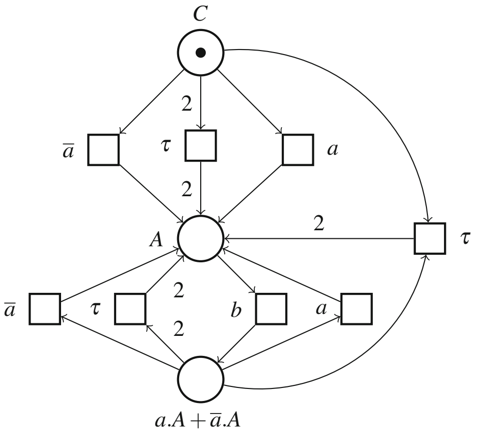


Fig. 6.5 The finite CCS net for $C \doteq a.A+\overline{a}.A$ with $A \doteq b.(a.A+\overline{a}.A)$

$$\begin{aligned}
&[\![a.A+\overline{a}.A]\!]_{\{A,C\}} = [\![p]\!]_{\{A,C\}} = (\{p,A\},\{a,\overline{a},\tau\},\\
&\qquad\qquad \{(\{p\},a,\{A\}),(\{p\},\overline{a},\{A\}),(\{p,p\},\tau,\{A,A\})\},\{p\}),\\
&[\![b.p]\!]_{\{A,C\}} = (\{b.p,p,A\},\{b,a,\overline{a},\tau\},\\
&\qquad \{(\{b.p\},b,\{p\}),(\{p\},a,\{A\}),(\{p\},\overline{a},\{A\}),(\{p,p\},\tau,\{A,A\})\},\{b.p\}),\\
&[\![A]\!]_{\{C\}} = (\{p,A\},\{b,a,\overline{a},\tau\},\{(\{A\},b,\{p\}),\\
&\qquad (\{p\},a,\{A\}),(\{p\},\overline{a},\{A\}),(\{p,p\},\tau,\{A,A\})\},\{A\}).
\end{aligned}$$

Then, by following the syntactic definition,

$$\begin{aligned}
&[\![a.A]\!]_{\{C\}} = (\{p,A,a.A\},\{b,a,\overline{a},\tau\},\{(\{a.A\},a,\{A\}),(\{A\},b,\{p\}),\\
&\qquad (\{p\},a,\{A\}),(\{p\},\overline{a},\{A\}),(\{p,p\},\tau,\{A,A\})\},\{a.A\}),\\
&[\![\overline{a}.A]\!]_{\{C\}} = (\{p,A,\overline{a}.A\},\{b,a,\overline{a},\tau\},\{(\{\overline{a}.A\},\overline{a},\{A\}),(\{A\},b,\{p\}),\\
&\qquad (\{p\},a,\{A\}),(\{p\},\overline{a},\{A\}),(\{p,p\},\tau,\{A,A\})\},\{\overline{a}.A\}).
\end{aligned}$$

$$\begin{aligned}
&[\![a.A+\overline{a}.A]\!]_{\{C\}} = [\![p]\!]_{\{C\}} = (\{p,A\},\{b,a,\overline{a},\tau\},\\
&\quad \{(\{A\},b,\{p\}),(\{p\},a,\{A\}),(\{p\},\overline{a},\{A\}),(\{p,p\},\tau,\{A,A\})\},\{p\}).
\end{aligned}$$

$$\begin{aligned}
&[\![C]\!]_{\emptyset} = (\{C,p,A\},\{b,a,\overline{a},\tau\},\{(\{A\},b,\{p\}),(\{p\},a,\{A\}),(\{p\},\overline{a},\{A\}),\\
&\qquad (\{p,p\},\tau,\{A,A\}),(\{C\},a,\{A\}),(\{C\},\overline{a},\{A\}),(\{C,p\},\tau,\{A,A\}),\\
&\qquad (\{C,C\},\tau,\{A,A\})\},\{C\}).
\end{aligned}$$

The resulting net is outlined in Figure 6.5. □

In order to prove that, for any restriction-free FNC process p, the operational net $Net(p)$ is exactly equal to the denotational net $[\![p]\!]_{\emptyset}$, we need four auxiliary lemmata, dealing with $Net(p,I)$, i.e., the operational net associated with p, assuming that the constants in I are undefined.

Lemma 6.8. *For any restriction-free process p, let $Net(p,I) = (S',A',T',dec(p))$. It follows that $Net(\mu.p,I) = (S,A,T,\{\mu.p\})$, where S,A and T are defined as in Table 6.11.*

Proof. By axiom (pref), the only transition enabled at the initial marking $\{\mu.p\}$ is $t = (\{\mu.p\},\mu,dec(p))$. Therefore, all the places in $dom(dec(p))$ are (statically) reachable from the initial marking $\{\mu.p\}$ and so all the places in S' are statically reachable as well, as $Net(p)$ is statically reduced. Therefore, $S = \{\mu.p\} \cup S'$. Now, since $Net(\mu.p,I)$ is statically reduced, T is the set of all the transitions that are statically enabled at S. Set T' contains all the transitions statically enabled at S', so it remains to compute the transitions that are statically enabled by place $\mu.p$ together with some place $s \in S'$. This is exactly the set $\{t\} \otimes T'$, whose transitions are derivable by rule (com), so that $T = \{t\} \cup T' \cup (\{t\} \otimes T')$, as required. □

Lemma 6.9. *For any sequential FNC process p and constant C such that $C \doteq p$, let $Net(p,I \cup \{C\}) = (S',A',T',dec(p))$. It follows that $Net(C,I) = (S,A,T,\{C\})$, where S,A and T are defined as in Table 6.11.*

Proof. By rule (cons), from place C we can derive only the transitions that are derivable from the place $\{p\}$ (assuming $p \neq \mathbf{0}$). These transitions are in the set T_1. Therefore, all the places statically reachable from the place p are also reachable from C, where the place p is to be taken only if it is reachable in one or more steps from p itself (definition of the set S''). Therefore, the set T of statically enabled transitions must include the following: (i) all the transitions statically enabled at S'', i.e., the set T''; moreover, (ii) all the transitions with pre-set $C \oplus m_1$ (with $|m_1| \leq 1$) or $\{C,C\}$ that are obtained by rule (cons) first, and then possibly also by rule (com) in case m_1 is not empty or the pre-set is $\{C,C\}$; these transitions are those in the set T_1. Hence, both S and T satisfy the definition in Table 6.11, as required. □

Lemma 6.10. *For any restriction-free FNC processes p_1 and p_2, let $Net(p_i,I) = (S_i,A_i,T_i,m_i)$, for $i = 1,2$. It follows that $Net(p_1 \,|\, p_2,I) = (S,A,T,m_0)$ where S,A,T and m_0 are defined as in Table 6.12.*

Proof. As $dec(p_i) = m_i$ for $i = 1,2$, it follows that $m_0 = dec(p_1 \,|\, p_2) = dec(p_1) \oplus dec(p_2) = m_1 \oplus m_2$. By induction on the static reachability relation $dom(m_1 \oplus m_2) \Longrightarrow^* V$, we shall prove that $V \subseteq S_1 \cup S_2$. This is enough, because all the places in S_1 or S_2 are statically reachable from $dom(m_1)$ or $dom(m_2)$, so that S must be $S_1 \cup S_2$. The base case of the induction is $dom(m_1 \oplus m_2) \Longrightarrow^* dom(m_1 \oplus m_2)$ and this case is trivial: $dom(m_1 \oplus m_2) \subseteq S_1 \cup S_2$ because $dom(m_1 \oplus m_2) = dom(m_1) \cup dom(m_2)$ and $dom(m_i) \subseteq S_i$ for $i = 1,2$. The inductive case is as follows: $dom(m_1 \oplus m_2) \Longrightarrow^* V_j \overset{t}{\Longrightarrow} V_{j+1}$. By induction, we can assume that $V_j \subseteq S_1 \cup S_2$, and so $V_j = V_j^1 \cup V_j^2$, with $V_j^i \subseteq S_i$ for $i = 1,2$. Note that, by Proposition 6.17, any transition $t \in T_{FNC}$ is such that either $|^\bullet t| = 1$ or $|^\bullet t| = 2$, but in the latter case $l(t) = \tau$. Therefore, if t is statically enabled at V_j, then either $dom(^\bullet t) \subseteq V_j^i$ for some $i = 1,2$ and so $dom(t^\bullet) \subseteq S_i$ because $Net(p_i)$ is statically reduced, or there exist two transitions t_1 and t_2 such that $l(t_1) = \alpha$, $l(t_2) = \overline{\alpha}$, $dom(^\bullet t_1) \subseteq V_j^1$,

$dom(^\bullet t_2) \subseteq V_j^2$ and $t = {}^\bullet t_1 \oplus {}^\bullet t_2 \xrightarrow{\tau} t_1^\bullet \oplus t_2^\bullet$ derived by rule (com); in this case, note that $dom(t_i^\bullet) \subseteq S_i$ because $Net(p_i)$ is statically reduced, so that $dom(t^\bullet) \subseteq S_1 \cup S_2$. Hence, the set $V_{j+1} = V_j \cup dom(t^\bullet)$ of places statically reachable in one step from V_j is a subset of $S_1 \cup S_2$. As a final observation, we note that if $S = S_1 \cup S_2$, necessarily $T = T_1 \cup T_2 \cup T_1 \otimes T_2$, where the transitions in $T_1 \otimes T_2$ are derived by means of rule (com), and so $A = A_1 \cup A_2$, with the possible addition of action τ. □

Lemma 6.11. *For any sequential processes p_1 and p_2, let $Net(p_i, I) = (S_i, A_i, T_i, m_i)$, for $i = 1, 2$. It follows that $Net(p_1 + p_2, I) = (S, A, T, \{p_1 + p_2\})$ where S, A and T are defined as in Table 6.12.*

Proof. From the initial marking $\{p_1 + p_2\}$, by rules (sum$_1$) and (sum$_2$), we can derive only the transitions that are derivable from the places $\{p_1\}$ or $\{p_2\}$ (assuming the summands are not the empty process $\mathbf{0}$). These transitions are included in the sets T_1'' and T_2'', respectively. Therefore, all the places reachable from the places p_1 or p_2 are also reachable from $p_1 + p_2$, where the place p_1 (or p_2) is to be taken only if it is reachable in one or more steps from p_1 itself (or p_2 itself): this corresponds to the definition of the sets S_1' and S_2'. Therefore, the set T of statically enabled transitions must include the following: (i) all the transitions statically enabled at S_1' or at S_2', i.e., the sets T_1' and T_2'; moreover, (ii) all the transitions with pre-set $p_1 + p_2 \oplus m_1$ or $\{p_1 + p_2, p_1 + p_2\}$, in correspondence with the transitions with pre-set $p_i \oplus m_1$ or $\{p_i, p_i\}$ for $i = 1, 2$, i.e., the set T_1'' and T_2''; finally, the transitions that are obtained by synchronizing one transition originating from the subnet of p_1 and one transition from the subnet of p_2 (by rule (com)), i.e., the set $(T_1' \cup T_1'') \otimes (T_2' \cup T_2'')$. It is possible to prove, by induction on the static reachability relation $\Longrightarrow^*$, that these additional transitions do not extend the set of reachable places, as done in the proof of Lemma 6.10. Hence, both S and T satisfy the definition in Table 6.12, as required. □

Now we are ready to state the correspondence theorem, stating that for any restriction-free process p, the net computed by the operational semantics is exactly the same net computed by the denotational semantics.

Theorem 6.10. *For any restriction-free, FNC process t, $Net(t) = [\![t]\!]_\emptyset$.*

Proof. By induction on the definitions of $[\![t]\!]_I$ and $Net(p, I)$, where the latter is the operational net associated with p, assuming that the constants in I are undefined. Then, the thesis follows for $I = \emptyset$.

The first base case is for $\mathbf{0}$ and the thesis is obvious, as, for any $I \subseteq \mathscr{C}ons$, $[\![\mathbf{0}]\!]_I = (\emptyset, \emptyset, \emptyset, \emptyset) = Net(\mathbf{0}, I)$, because no place/token is available and so no transition is derivable from $\emptyset$ by the operational rules in Table 6.9. The second base case is for C when $C \in I$, which corresponds to the case when C is not defined. In this case, for any $I \subseteq \mathscr{C}ons$ with $C \in I$, $[\![C]\!]_I = (\{C\}, \emptyset, \emptyset, \{C\}) = Net(C, I)$, because, being undefined, no transition is derivable from C (rule (cons) is not applicable). The inductive cases are as follows.

Action prefixing*: If $t = \mu.p$, then, by induction, we can assume that $Net(p, I) = (S', A', T', dec(p)) = [\![p]\!]_I$. The thesis $Net(\mu.p, I) = [\![\mu.p]\!]_I$ follows by Lemma 6.8.*

$$[\![(\nu a)p]\!]_I \;=\; ([\![p]\!]_I\{a'/a\})\setminus a' = (S\{a'/a\},(A\{a'/a\})\setminus a',(T\{a'/a\})\setminus a',m_0\{a'/a\})$$

given $[\![p]\!]_I = (S,A,T,m_0)$ where

$$\begin{aligned}
S\{a'/a\} &= \{s\{a'/a\} \mid s \in S\}\\
A\{a'/a\} &= \{\mu\{a'/a\} \mid \mu \in A\}\\
(A')\setminus a' &= \{\mu \in A' \mid \mu \neq a',\overline{a'}\}\\
T\{a'/a\} &= \{(m_1\{a'/a\},\mu\{a'/a\},m_2\{a'/a\}) \mid (m_1,\mu,m_2) \in T\}\\
(T')\setminus a' &= \{(m_1,\mu,m_2) \mid (m_1,\mu,m_2) \in T' \wedge \mu \neq a',\overline{a'}\}
\end{aligned}$$

Table 6.13 Denotational net semantics — part III

Constant*: In this case, $t = C$ and we assume that $C \doteq p$ and $C \notin I$. By induction, we have that $Net(p,I \cup \{C\}) = (S',A',T',dec(p)) = [\![p]\!]_{I\cup\{C\}}$, so that in p constant C is undefined. The thesis $Net(C,I) = [\![C]\!]_I$ follows by Lemma 6.9.*

Parallel*: By induction, we can assume that $Net(p_i,I) = (S_i,A_i,T_i,dec(p_i)) = [\![p_i]\!]_I$ for $i = 1,2$. The thesis $Net(p_1 \mid p_2,I) = [\![p_1 \mid p_2]\!]_I$ follows by Lemma 6.10.*

Choice*: By induction, we can assume that $Net(p_i,I) = (S_i,A_i,T_i,dec(p_i)) = [\![p_i]\!]_I$ for $i = 1,2$. The thesis $Net(p_1 + p_2,I) = [\![p_1 + p_2]\!]_I$ follows by Lemma 6.11.* □

The denotational semantics for the restriction operator is described in Table 6.13. The net $[\![(\nu a)p]\!]_I$ is essentially computed in two steps starting from the net $[\![p]\!]_I$: first, the substitution $\{a'/a\}$ is applied element-wise to all the places and transitions of the net for p; then, a *pruning* operation $-\setminus a'$ is performed to remove all the relabeled transitions that are now labeled with the restricted action a' or $\overline{a'}$. Differently from the previous cases, now the places are *extended* sequential processes.

Example 6.20. Consider the process $(\nu b)(a.b.c.\mathbf{0})$. The net for $a.b.c.\mathbf{0}$ is

$$\begin{aligned}
[\![a.b.c.\mathbf{0}]\!]_\emptyset = (&\{a.b.c.\mathbf{0},b.c.\mathbf{0},c.\mathbf{0}\},\{a,b,c\},\{(\{a.b.c.\mathbf{0}\},a,\{b.c.\mathbf{0}\}),\\
&(\{b.c.\mathbf{0}\},b,\{c.\mathbf{0}\}),(\{c.\mathbf{0}\},c,\emptyset)\},\{a.b.c.\mathbf{0}\}).
\end{aligned}$$

Now if we apply the substitution $\{b'/b\}$, we get the net

$$\begin{aligned}
&(\{a.b'.c.\mathbf{0},b'.c.\mathbf{0},c.\mathbf{0}\},\{a,b',c\},\{(\{a.b'.c.\mathbf{0}\},a,\{b'.c.\mathbf{0}\}),(\{b'.c.\mathbf{0}\},b',\{c.\mathbf{0}\}),\\
&(\{c.\mathbf{0}\},c,\emptyset)\},\{a.b'.c.\mathbf{0}\}),
\end{aligned}$$

and then, by applying the pruning operation $-\setminus b'$, we get the net

$$\begin{aligned}
[\![(\nu b)(a.b.c.\mathbf{0})]\!]_\emptyset = (&\{a.b'.c.\mathbf{0},b'.c.\mathbf{0},c.\mathbf{0}\},\{a,c\},\\
&\{(\{a.b'.c.\mathbf{0}\},a,\{b'.c.\mathbf{0}\}),(\{c.\mathbf{0}\},c,\emptyset)\},\{a.b'.c.\mathbf{0}\}),
\end{aligned}$$

which is not statically reduced, because place $c.\mathbf{0}$ is not statically reachable from the initial marking. The subnet of $[\![(\nu b)(a.b.c.\mathbf{0})]\!]_\emptyset$ which is statically reachable from the initial marking $\{a.b'.c.\mathbf{0}\}$ is

$$(\{a.b'.c.\mathbf{0}, b'.c.\mathbf{0}\}, \{a\}, \{(\{a.b'.c.\mathbf{0}\}, a, \{b'.c.\mathbf{0}\})\}, \{a.b'.c.\mathbf{0}\}),$$

which is exactly $Net((\nu b)(a.b.c.\mathbf{0}))$, i.e., the statically reachable subnet of the denotational semantics net is the operational semantics net. □

The observation at the end of the previous example holds in general: the statically reachable subnet of the denotational semantics net is the operational semantics net for any restricted FNC process. In order to prove this result, we need an auxiliary lemma, where $Net_s(N(m_0))$ denotes the subnet of N statically reachable from the initial marking m_0 (Definition 3.9).

Lemma 6.12. *For any FNC process p, let $Net(p) = (S, A, T, m_0)$. It follows that $Net((\nu a)p) = Net_s((Net(p)\{a'/a\}) \setminus a')$, where $(Net(p)\{a'/a\}) \setminus a' = (S\{a'/a\}, (A\{a'/a\}) \setminus a', (T\{a'/a\}) \setminus a', m_0\{a'/a\})$, and the operations of substitution and pruning are defined in Table 6.13.*

Proof. *By induction on the static reachability relation $\Longrightarrow^*$, we prove that the set $S_{(\nu a)p}$ of the statically reachable places of $Net((\nu a)p)$ is the subset of $S\{a'/a\}$ of the places statically reachable from $m_0\{a'/a\}$ in $(Net(p)\{a'/a\}) \setminus a'$. And vice versa, the places statically reachable from $m_0\{a'/a\}$ in $(Net(p)\{a'/a\}) \setminus a'$ constitute exactly the set $S_{(\nu a)p}$.*

Since $m_0 = dec(p)$, also $m_0\{a'/a\} = dec(p)\{a'/a\} = dec((\nu a)p)$, so that the initial markings of the two nets are the same. Hence, the base case of the induction is true: $dom(dec((\nu a)p)) \Longrightarrow^* dom(dec((\nu a)p)) = dom(m_0\{a'/a\}) \subseteq S\{a'/a\}$. Now assume that there exist a set of places V_j and a transition t in $Net((\nu a)p)$ such that $dom(dec((\nu a)p)) \Longrightarrow^* V_j \overset{t}{\Longrightarrow} V_{j+1}$. By induction, we can assume that also $dom(m_0\{a'/a\}) \Longrightarrow^* V_j$, so that V_j must be of the form $V_j'\{a'/a\}$ for a suitable $V_j' \subseteq S$. Hence, if transition t is statically enabled at $V_j = V_j'\{a'/a\}$, then, by Proposition 6.16, there exists a transition t' statically enabled at V_j', where $t = t'\{a'/a\}$ and t' is a transition of $Net(p)$; moreover, $l(t') \neq a, \overline{a}$. Hence, also in the net $(Net(p)\{a'/a\}) \setminus a'$ we have that $V_j \overset{t}{\Longrightarrow} V_{j+1}$, as required. Conversely, if a transition t' of $Net(p)$ is statically enabled at V_j', then $t = t'\{a'/a\}$ is a transition of $Net((\nu a)p)$ only if $l(t) \neq a', \overline{a'}$; in this case, t is statically enabled at $V_j = V_j'\{a'/a\}$, as required.

Summing up, any place statically reachable from $dom(dec((\nu a)p))$ in $Net((\nu a)p)$ is also a place statically reachable from $dom(m_0\{a'/a\})$ in $(Net(p)\{a'/a\}) \setminus a'$, and vice versa. □

Theorem 6.11. *For any FNC process p, $Net(p) = Net_s([\![p]\!]_\emptyset)$, i.e., $Net(p)$ is exactly the statically reachable subnet of $[\![p]\!]_\emptyset$.*

Proof. *An FNC process p is either a restriction-free process t, or there exists an action a and an FNC process p' such that $p = (\nu a)p'$. In the former case, $Net(t) =$*

$[\![t]\!]_\emptyset$ by Theorem 6.10, so that the thesis follows trivially, because $[\![t]\!]_\emptyset$ is statically reduced: $Net_s([\![t]\!]_\emptyset) = [\![t]\!]_\emptyset$.

In the latter case, we can assume, by induction, that $Net(p') = Net_s([\![p']\!]_\emptyset)$. If $Net(p') = (S, A, T, m_0)$, then $Net((\nu a)p') = Net_s((Net(p')\{a'/a\}) \setminus a')$, by Lemma 6.12. By the definition in Table 6.13, $[\![(\nu a)p']\!]_\emptyset = ([\![p']\!]_\emptyset\{a'/a\}) \setminus a'$. Moreover, it holds trivially that $Net_s(N(m_0)) = Net_s(Net_s(N(m_0)))$ for any net $N(m_0)$. Therefore,

$$\begin{aligned} Net_s([\![(\nu a)p']\!]_\emptyset) &= Net_s(([\![p']\!]_\emptyset\{a'/a\}) \setminus a') \\ &= Net_s((Net_s([\![p']\!]_\emptyset)\{a'/a\}) \setminus a') \\ &= Net_s((Net(p')\{a'/a\}) \setminus a') \\ &= Net((\nu a)p'). \end{aligned}$$

□

6.8 RCS

RCS is the calculus whose terms are generated from actions in $Act = \mathscr{L} \cup \overline{\mathscr{L}} \cup \{\tau\}$ and constants, as described by the following abstract syntax:

$$\begin{array}{lllll} s & ::= \mathbf{0} \mid & \mu.q & \mid s+s & \\ q & ::= s \mid & C & & \textit{sequential terms} \\ t & ::= q \mid & t \,|\, t & & \textit{restriction-free terms} \\ p & ::= t & \mid (\nu a)p & & \textit{general terms} \end{array}$$

where, as usual, we assume that $Const(p)$ is finite and any constant C is equipped with a definition in syntactic category s. The only difference with respect to FNC is that the action prefixing operator $\mu.-$ is applied to processes of category q, instead of category t. In other words, RCS is obtained by adding to CFM the external restriction operator (and the synchronization capability to its parallel operator). So, the class of RCS processes is included in the class of FNC processes; it includes the CFM processes, while it is incomparable with the class of BPP processes. Sometimes the syntax of RCS processes is more succinctly given as

$$\begin{aligned} s &::= \Sigma_{j \in J} \mu_j.s_j \mid C \\ q &::= s \mid q \,|\, q \\ p &::= q \mid (\nu a)p \end{aligned}$$

which more clearly expresses the fact that an RCS process is obtained as the parallel composition (with external restriction) of a finite number of (communicating) SFM processes. RCS is essentially *regular* CCS [GV15].

With respect to the LTS semantics, RCS is as expressive as SFM: for any RCS process p, the LTS $\mathscr{C}_p = (\mathscr{P}_p$, sort(p), $\rightarrow_p, p)$ reachable from p is finite-state. This fact can be proved by structural induction. The base case of the induction is guaranteed by Corollary 4.2, as it ensures that the LTS reachable from p, $\mathscr{C}_p = (\mathscr{P}_p, sort(p), \rightarrow_p, p)$, is finite-state for any SFM process p. Now, by induction, we can assume that the cardinality of the set of states reachable from a restriction-free RCS process p_i is k_i, and moreover that $sort(p_i)$ is finite, for $i = 1,2$; by inspecting the SOS rules (Par$_1$), (Par$_2$) and (Com), it is an easy exercise to observe that the cardinality of the set of states reachable from the RCS process $p_1 \mid p_2$ is $k_1 \times k_2$, and that $sort(p_1 \mid p_2) \subseteq sort(p_1) \cup sort(p_2) \cup \{\tau\}$. Finally, we can assume that the cardinality of the set of states reachable from the (possibly restricted) RCS process p is k, and moreover that $sort(p)$ is finite; by inspecting the SOS rules (Res), it is clear that the set of states reachable from the RCS process $(\nu a)p$ is less than, or equal to, k, and that $sort((\nu a)p) \subseteq sort(p)$. Therefore, we have the following fact.

Proposition 6.26. (Finite-state LTS) *For any RCS process p, the LTS reachable from p, $\mathscr{C}_p = (\mathscr{P}_p, sort(p), \rightarrow_p, p)$, is finite-state.* □

However, with respect to the net semantics, RCS is more expressive than CFM, as it represents, besides all the concurrent FSMs, also many finite CCS nets. For instance, the net in Figure 6.3(a) is a finite CCS net originating from the RCS process $a.b.\mathbf{0} \mid \overline{a}.\overline{b}.\mathbf{0}$.

Example 6.21. Let us consider the RCS process $p = (\nu a)(P \mid P \mid C)$, where constants P and C are defined as follows: $P \doteq prod.P'$, $P' \doteq a.P$, $C \doteq \overline{a}.C'$ and $C' \doteq cons.C$. It is an easy exercise to check that $Net(p)$ is the finite CCS net in Figure 3.2(a). □

Now we want to prove that, for any RCS process p, $Net(p)$ is a *bounded*, finite CCS net. By Corollary 6.3, any FNC process p is such that $Net(p)$ is a finite CCS net; hence, since RCS is a subcalculus of FNC, $Net(p)$ is a finite CCS net for any RCS process p, too. So, it remains to prove that such a net is bounded. By syntactic definition, the basic constituents of any RCS process $p = (\nu L)(p_1 \mid \ldots \mid p_n)$ are SFM processes $p_1, \ldots, p_n$. By Proposition 4.6, for any $t \in T_{SFM}$, we have that $|{}^\bullet t| = 1$ and $|t^\bullet| \leq 1$; hence, the net transitions that p_i may generate have a singleton pre-set and a post-set which is a singleton at most. By rule (com), two net transitions t_1 and t_2 can be synchronized, producing a transition t with pre-set ${}^\bullet t = {}^\bullet t_1 \oplus {}^\bullet t_2$ and post-set $t^\bullet = t_1^\bullet \oplus t_2^\bullet$, so that $|{}^\bullet t| = 2$ and $|t^\bullet| \leq 2$; since $l(t) = \tau$, t cannot be used as a premise for rule (com), so that no transition generable by the rules may have a pre-set larger than two and a corresponding post-set larger than two. Hence, for any m and any t, if $m[t\rangle m'$, then $|m'| \leq |m|$, because t does not produce more tokens than it consumes. Therefore, the following theorem holds.

Theorem 6.12. *For any process p of RCS, $Net(p)$ is a k-bounded, finite CCS net, where $k = |dec(p)|$.* □

However, RCS is not able to express all the possible bounded, finite CCS nets. For instance, consider the simple net

$$N(\{s_1\}) = (\{s_1, s_2, s_3\}, \{a, b, c\}, \{(\{s_1\}, a, \{s_2, s_3\}), (\{s_2\}, b, \emptyset), (\{s_3\}, c, \emptyset)\}, \{s_1\}),$$

depicted in Figure 3.6(a). Clearly, no RCS process can model such a net, because the transition $(\{s_1\}, a, \{s_2, s_3\})$ produces more tokens than it consumes.

Chapter 7
Adding Multi-party Communication: FNM

Abstract FNC is enhanced with a simple operator for atomicity, called strong prefixing, so that action sequences can be used as labels for transitions. Multi-party communication is implemented by means of atomic sequences of binary, CCS-like synchronizations; hence, the parallel operator has a richer synchronization discipline, dealing also with action sequences. The resulting calculus is a subset of Multi-CCS, called *Finite-Net* Multi-CCS (FNM, for short), which is expressive enough to model the class of finite P/T nets.

7.1 Preliminaries

This chapter is the natural continuation of the previous one; it relies upon many concepts and ideas developed there, which here are simply extended to the new operator of strong prefixing; hence, for easier and better understanding, it is advisable to first read Chapter 6.

7.1.1 Syntax and Informal Semantics

As for FNC, let $\mathscr{L}$ be a finite set of *names*, ranged over by $a, b, c, \ldots$, also called the *input actions*. Let $\overline{\mathscr{L}}$ be the set of *co-names*, ranged over by $\overline{a}, \overline{b}, \overline{c}, \ldots$, also called the *output actions*. The set $\mathscr{L} \cup \overline{\mathscr{L}}$, ranged over by $\alpha, \beta, \ldots$, is the set of *observable actions*. By $\overline{\alpha}$ we mean the complement of α, assuming that $\overline{\overline{\alpha}} = \alpha$. Let $Act = \mathscr{L} \cup \overline{\mathscr{L}} \cup \{\tau\}$, such that $\tau \notin \mathscr{L} \cup \overline{\mathscr{L}}$, be the finite set of *actions* (or *labels*), ranged over by μ. Action τ denotes an invisible, internal activity.

As for the previous calculi, let $\mathscr{C}ons$ be a countable set of process *constants*, disjoint from Act, ranged over by $A, B, C, \ldots$, possibly indexed.

R. Gorrieri, *Process Algebras for Petri Nets*, Monographs in Theoretical Computer Science. An EATCS Series, DOI 10.1007/978-3-319-55559-1_7

Finite-Net Multi-CCS (FNM for short) is the calculus whose terms are generated from actions and constants as described by the following abstract syntax:

$$\begin{array}{lll} s ::= \mathbf{0} \mid \mu.t \mid \underline{a}.s \mid s+s & & \\ q ::= s \mid C & & \textit{sequential terms} \\ t ::= q \mid t|t & & \textit{restriction-free terms} \\ p ::= t \mid (\nu a)p & & \textit{general terms} \end{array}$$

where, as usual, we assume that a constant C is defined by a process in syntactic category s. The only new operator of the calculus is *strong prefixing*: $\underline{a}.s$ is a strongly prefixed process, where the strong prefix a is the first input action of a transaction that continues with the *sequential* process s (provided that s can complete the transaction). Its operational SOS rule is

$$\text{(S-Pref)}\ \frac{s \xrightarrow{\sigma} t}{\underline{a}.s \xrightarrow{a \diamond \sigma} t} \quad \text{where} \quad a \diamond \sigma = \begin{cases} a & \text{if } \sigma = \tau, \\ a\sigma & \text{otherwise,} \end{cases}$$

where σ is either a single action, or an atomic sequence of *visible* actions, composed of inputs only, but possibly ending with an output; more precisely, σ ranges over the set of *labels* $\mathscr{A} = \{\tau\} \cup \mathscr{L}^* \cdot (\mathscr{L} \cup \overline{\mathscr{L}})$, and so $a \diamond \sigma$ ranges over $\mathscr{L}^+ \cdot (\mathscr{L} \cup \overline{\mathscr{L}})$. In the following, the sequence σ is called an *atomic sequence* if $\sigma \in \mathscr{L}^+ \cdot (\mathscr{L} \cup \overline{\mathscr{L}})$ and so $|\sigma| \geq 2$.

Rule (S-Pref) allows for the creation of transitions labeled with an atomic sequence. For instance, $\underline{a}.(b.\mathbf{0} + \overline{c}.\mathbf{0})$ can perform two transitions reaching state $\mathbf{0}$: one labeled with the atomic sequence ab, the other one with $a\overline{c}$, as illustrated by the following proof trees:

$$\text{(S-Pref)}\ \dfrac{\text{(Sum}_1)\ \dfrac{\text{(Pref)}\ \dfrac{}{b.\mathbf{0} \xrightarrow{b} \mathbf{0}}}{b.\mathbf{0} + \overline{c}.\mathbf{0} \xrightarrow{b} \mathbf{0}}}{\underline{a}.(b.\mathbf{0} + \overline{c}.\mathbf{0}) \xrightarrow{ab} \mathbf{0}} \qquad \text{(S-Pref)}\ \dfrac{\text{(Sum}_2)\ \dfrac{\text{(Pref)}\ \dfrac{}{\overline{c}.\mathbf{0} \xrightarrow{\overline{c}} \mathbf{0}}}{b.\mathbf{0} + \overline{c}.\mathbf{0} \xrightarrow{\overline{c}} \mathbf{0}}}{\underline{a}.(b.\mathbf{0} + \overline{c}.\mathbf{0}) \xrightarrow{a\overline{c}} \mathbf{0}}$$

In order for $\underline{a}.s$ to make a move, it is necessary that s be able to perform a transition, i.e., the rest of the transaction. Hence, if $s \xrightarrow{\sigma} t$ then $\underline{a}.s \xrightarrow{a \diamond \sigma} t$. Note that $\underline{a}.\mathbf{0}$ cannot execute any action, as $\mathbf{0}$ is terminated. If a transition is labeled with $\sigma = a_1 \dots a_{n-1}\alpha_n$, then all the actions $a_1 \dots a_{n-1}$ are inputs due to strong prefixes, while α_n is due to a normal prefix (or α_n is a strong input prefix followed by a normal prefix τ).

A consequence of the fact that transitions may be labeled by atomic sequences is the need for a new SOS rule for communication, which extends the FNC rule (Com). The new rule is

$$\text{(S-Com)}\ \frac{p \xrightarrow{\sigma_1} p' \qquad q \xrightarrow{\sigma_2} q'}{p\,|\,q \xrightarrow{\sigma} p'\,|\,q'}\ \mathit{Sync}(\sigma_1, \sigma_2, \sigma),$$

which has a side condition on the possible synchronizability of σ_1 and σ_2, whose result is σ. Relation $Sync(\sigma_1, \sigma_2, \sigma)$ holds if at least one of the two sequences is a single output action, say $\sigma_1 = \overline{a}$, and the other one is either the complementary input action a or an atomic sequence starting with a. Hence, it is not possible to synchronize two atomic sequences. This means that, usually, a multi-party synchronization can take place only among one *leader*, i.e., the process performing the atomic sequence of inputs, and as many other components (the *servants*), as the length of the atomic sequence, where each servant executes one output action. This is strictly the case for so-called *well-formed processes* (see Section 7.1.3).

An FNM term p is an FNM *process* if it is fully defined and the set of constants $Const(p)$ used by p is finite, where function $Const(-)$, defined for the FNC operators in Section 6.1, is extended to strong prefixing as follows: $Const(\underline{a}.s) = Const(s)$.

The set of FNM processes is denoted by $\mathscr{P}_{FNM}$, and the set of its sequential processes, i.e., of the processes in syntactic category q, by $\mathscr{P}_{FNM}^{seq}$.

We extend the definitions of free names $F(p,I)$ and free outputs $G(p,I)$ (see Definition 6.1) to strong prefixing as follows:

$$F(\underline{a}.s,I) = F(s,I) \cup \{a\},$$
$$G(\underline{a}.s,I) = G(s,I).$$

Of course, for any FNM process p, the sets $fn(p)$ (free names of p), $bn(p)$ (bound names of p) and $fo(p)$ (free outputs of p) are finite. A process p is *closed* if the set $fn(p)$ is empty. Process p is *output-closed* if the set $fo(p)$ is empty.

7.1.2 Extended Processes and Sequential Subterms

As for FNC, we assume we have the set $\mathscr{G} = \mathscr{L} \cup \mathscr{L}'$, ranged over by γ, as the set of input actions and their restricted counterparts; and the set $\overline{\mathscr{G}} = \overline{\mathscr{L}} \cup \overline{\mathscr{L}'}$ as the set of output actions and their restricted counterparts, ranged over by $\overline{\gamma}$. The set of all actions $Act_\gamma = \mathscr{G} \cup \overline{\mathscr{G}} \cup \{\tau\}$, ranged over by μ (with abuse of notation), is used to build the set of *extended terms*, whose syntax is as follows:

$$\begin{array}{llll} s & ::= \mathbf{0} \mid \mu.t \mid \underline{\gamma}.s \mid s+s & & \mu \in Act_\gamma, \gamma \in \mathscr{G} \\ q & ::= s \mid C & & \textit{sequential extended terms} \\ t & ::= q \mid t|t & & \textit{restriction-free extended terms} \\ p & ::= t \mid (\nu a)p & & \textit{general extended terms} \end{array}$$

where the prefixes are taken from the set Act_γ, the strong prefixes from the set $\mathscr{G}$ and the bound actions from the set $\mathscr{L}$. An extended FNM term $p = (\nu L)t$ is an *extended process* if it is fully defined, $Const(p)$ is finite and t is *admissible*, i.e.,

$$\forall a \in \mathscr{L}, \{a, a'\} \not\subseteq fn(t),$$

where the function $fn(-)$ is defined on extended terms in the obvious way. By the notation $ad(t)$ we indicate that t is admissible.

By $\mathscr{P}_{FNM}^{\gamma}$ we denote the set of all extended FNM processes. By $\mathscr{P}_{FNM}^{\gamma,par}$ we denote the set of all restriction-free, extended FNM processes, i.e., those extended processes in syntactic category t. By $\mathscr{P}_{FNM}^{\gamma,seq}$ we denote the set of all sequential, extended FNM processes, i.e., those extended processes in syntactic category q.

Syntactic substitution of a restricted action a' for all the occurrences of the visible action a in a process p, denoted by $p\{a'/a\}$, was formally defined for extended FNC processes in Section 6.1.2; it is defined for strong prefixing as follows:

$$\begin{aligned} (\underline{a}.s)\{a'/a\} &= \underline{a'}.(s\{a'/a\}), \\ (\underline{\gamma}.s)\{a'/a\} &= \underline{\gamma}.(s\{a'/a\}) \qquad \text{if } \gamma \neq a. \end{aligned}$$

Of course, for any extended FNM process p and for any $a \in \mathscr{L}$, $p\{a'/a\}$ is an extended FNM process. Moreover, for any extended, restriction-free FNM process t such that $L' = fn(t) \cap \mathscr{L}'$, we have that $(t\{L/L'\})\{L'/L\} = t$, by admissibility of t, where $\{L/L'\}$ is the inverse substitution (see Remark 6.2).

Function $i : \mathscr{P}_{FNM} \to \mathscr{P}_{FNM}^{\gamma,par}$, defined as $i((\nu L)t) = t\{L'/L\}$, maps FNM processes to restriction-free, extended FNM processes. Function i is the identity over restriction-free processes: $i(t) = t$. In general, i is not injective; however, it is possible to prove that i is surjective. Given any restriction-free, extended FNM process t, let $L' = fn(t) \cap \mathscr{L}'$; since t is an extended process, t is admissible: there is no $a \in \mathscr{L}$ such that $a \in fn(t)$ and $a' \in fn(t)$; hence, for any $a' \in L'$, we have $a \notin fn(t)$. Therefore, the FNM process $p = (\nu L)(t\{L/L'\})$ is such that $i(p) = i((\nu L)(t\{L/L'\})) = i(t\{L/L'\})\{L'/L\} = (t\{L/L'\})\{L'/L\} = t$.

We will see that the net semantics for a FNM process p is actually given to its corresponding restriction-free, extended FNM process $i(p)$, because it holds that $dec(p) = dec(i(p))$ (see Proposition 7.18).

We are interested in singling out the set of sequential subterms of an FNM process p. We will prove that, for any p, the set of its sequential subterms $sub(p)$ is finite and each of its sequential subterms is a sequential, extended FNM process. The definition of the auxiliary function $sub(p,I)$, outlined in Table 6.4, is to be extended to strong prefixing as follows: $sub(\underline{a}.s,I) = \{\underline{a}.s\} \cup sub(s,I)$. Then, $sub(p)$ is simply $sub(p,\emptyset)$.

Lemma 7.1. *For any extended, restriction-free FNM process t and for any $L \subseteq \mathscr{L}$, $sub(t)\{L'/L\} = sub(t\{L'/L\})$.*

Proof. By induction on the definition of $sub(t,\emptyset)$. The cases corresponding to the FNC operators were discussed in Lemma 6.1. For the inductive case of strong prefixing, i.e., when $t = \underline{\gamma}.s$, we have that $t\{L'/L\} = \underline{\gamma}.(s\{L'/L\})$, if $\gamma \neq a$ for all a in L. Hence, $sub(t\{L'/L\},I\{L'/L\}) = sub(\underline{\gamma}.(s\{L'/L\}),I\{L'/L\}) = \{\underline{\gamma}.(s\{L'/L\})\} \cup sub(s\{L'/L\},I\{L'/L\})$. By induction, $sub(s,I)\{L'/L\} = sub(s\{L'/L\},I\{L'/L\})$; so, the thesis holds:

$$sub((\underline{\gamma}.s)\{L'/L\}, I\{L'/L\}) = \{\underline{\gamma}.(s\{L'/L\})\} \cup sub(s\{L'/L\}, I\{L'/L\}) =$$
$$= \{\underline{\gamma}.s\}\{L'/L\} \cup sub(s,I)\{L'/L\} = sub(\underline{\gamma}.s,I)\{L'/L\}.$$

The case when $\gamma = a$ for some $a \in L$ is similar, hence omitted. □

Proposition 7.1. *For any FNM process p, $sub(p) = sub(i(p))$.*

Proof. If p is restriction-free, then $i(p) = p$, and so the thesis follows trivially. If $p = (\nu L)t$, then $sub(p) = sub(t)\{L'/L\}$. Moreover, $i(p) = t\{L'/L\}$. So, the thesis follows by Lemma 7.1: $sub(i(p)) = sub(t\{L'/L\}) = sub(t)\{L'/L\} = sub(p)$. □

Theorem 7.1. *For any FNM process p, the set of its sequential subterms $sub(p)$ is finite; moreover, $sub(p) \subseteq \mathscr{P}_{FNM}^{\gamma,seq}$.*

Proof. By induction on the definition of $sub(p,\emptyset)$. The proof is the same as for the analogous Theorem 6.1 for FNC. The additional inductive case for strong prefixing is easy: If $p = \underline{a}.p'$, then $sub(\underline{a}.p',I) = \{\underline{a}.p'\} \cup sub(p',I)$. By induction, we can assume that $sub(p',I)$ is finite and a subset of $\mathscr{P}_{FNM}^{\gamma,seq}$. Hence, $sub(\underline{a}.p',I)$ is finite and a subset of $\mathscr{P}_{FNM}^{\gamma,seq}$ because also $\underline{a}.p'$ is a sequential process. □

Proposition 7.2. *For any FNM processes p and q, the following hold:*

(i) *if p is sequential, then $p \in sub(p)$, and*
(ii) *if $p \in sub(q)$, then $sub(p) \subseteq sub(q)$.*

Proof. The proof of (i) follows directly from the definition of $sub(-)$. The proof of (ii) follows by the observation that the definition of $sub(q)$ recursively calls itself on all of its subterms. □

7.1.3 Well-Formed Processes

A *well-formed* process is a process satisfying a simple syntactic condition, ensuring that its executable atomic sequences are composed of input actions only, so that the synchronization of atomic sequences is impossible, even indirectly. As a matter of fact, even if the strong prefixes are all input actions by syntactic definition, the last action of the sequence, being a normal prefix, may be an output action, which can be used for synchronization with another atomic sequence. For instance, the non-well-formed process $\underline{a}.\overline{b}.\mathbf{0}\,|\,(\overline{a}.\mathbf{0}\,|\,\underline{b}.c.\mathbf{0})$ can perform the atomic sequences $a\overline{b}$ and bc, and also the single action $\overline{a}$; an indirect synchronization between these two atomic sequences is possible by first synchronizing $a\overline{b}$ with $\overline{a}$, yielding action $\overline{b}$, which is then synchronized with bc, yielding c as the result of this three-way synchronization. Well-formedness of a process p is a sufficient condition to ensure that p is unable to synchronize two of its concurrent atomic sequences, even indirectly; this property will be crucial in guaranteeing that the net associated with a well-formed process p is finite, because (indirect) synchronizability of atomic sequences may cause the generation of infinitely many statically enabled transitions, as Example 7.8 will show.

$$\frac{}{\mathit{wf}(\mathbf{0},I)} \qquad \frac{\mathit{wf}(p,I)}{\mathit{wf}(\mu.p,I)} \qquad \frac{\mathit{wf}(p,I) \quad \emptyset \neq \mathit{In}(p) \subseteq \mathscr{L}^{+} \cup \{\tau\}}{\mathit{wf}(\underline{a}.p,I)}$$

$$\frac{\mathit{wf}(p,I)}{\mathit{wf}((\nu a)p,I)} \qquad \frac{C \in I}{\mathit{wf}(C,I)} \qquad \frac{\mathit{wf}(p,I \cup \{C\}) \quad C \doteq p \quad C \notin I}{\mathit{wf}(C,I)}$$

$$\frac{\mathit{wf}(p_1,I) \quad \mathit{wf}(p_2,I)}{\mathit{wf}(p_1 + p_2,I)} \qquad \frac{\mathit{wf}(p_1,I) \quad \mathit{wf}(p_2,I)}{\mathit{wf}(p_1 \,|\, p_2,I)}$$

Table 7.1 Well-formedness predicate

For any FNM process p in syntactic category s, $\mathit{In}(p) \subseteq \mathscr{A}$ is the set of *initials* of p, defined inductively as follows:

$$\begin{aligned} \mathit{In}(\mathbf{0}) &= \emptyset, & \mathit{In}(p_1 + p_2) &= \mathit{In}(p_1) \cup \mathit{In}(p_2), \\ \mathit{In}(\mu.p) &= \{\mu\}, & \mathit{In}(\underline{a}.p) &= a \diamond \mathit{In}(p), \end{aligned}$$

where $a \diamond \mathit{In}(p) = \{a \diamond \sigma \mid \sigma \in \mathit{In}(p)\}$.

Now we can define the well-formedness predicate; a process p is well formed if predicate $\mathit{wf}(p)$ holds, and $\mathit{wf}(p)$ holds if the auxiliary relation (with the same name, with abuse of notation) $\mathit{wf}(p,\emptyset)$ holds; the auxiliary relation $\mathit{wf}(p,I)$, where the second parameter is a set of constants, is defined as the least relation induced by the axioms and rules of Table 7.1. There are two base cases: when $p = \mathbf{0}$ and when $p = C$ with $C \in I$. The rule for C, when $C \notin I$, shows how the set I is used to avoid cycling on the possibly recursive, constant C; in fact, we have to check that $\mathit{wf}(p, I \cup \{C\})$ holds, so that possible occurrences of C inside its body p will be considered well formed by the base case above. The assumption that any process uses finitely many constants ensures that the well-formedness predicate is well defined. The other rules are essentially by induction on the structure of p. The only non-trivial one is that for strong prefixing: $\mathit{wf}(\underline{a}.p,I)$ holds not only if $\mathit{wf}(p,I)$ holds, but also if the nonempty set of its initials does not contain output actions or atomic sequences ending with an output action. As an example, $\underline{a}.\overline{b}.\mathbf{0}$ is not well formed, because, even if $\mathit{wf}(\overline{b}.\mathbf{0},\emptyset)$ holds, we have that the additional condition $\mathit{In}(\overline{b}.\mathbf{0}) \subseteq \mathscr{L}^{+} \cup \{\tau\}$ is not satisfied. Analogously, $\underline{a}.\mathbf{0}$ is not well-formed because, even if $\mathit{wf}(\mathbf{0})$ holds, we have that the additional condition $\mathit{In}(\mathbf{0}) \neq \emptyset$ is not satisfied; this additional condition is not essential for the correctness of the net semantics, but it is useful in discarding irrelevant processes that cannot perform any transition. Summing up, $\underline{a}.p$ is well formed if p is well formed, and the set $\mathit{In}(p)$ of initials of p is not empty and contains sequences of inputs (or τ), only.

The class of well-formed processes can also be characterized by means of an abstract grammar, more restrictive than that of Section 7.1.1, described in Table 7.2, where any process p in syntactic category s is such that $\emptyset \neq \mathit{In}(p) \subseteq \mathscr{L}^{+} \cup \{\tau\}$, so that $\underline{a}.p$ is well formed by construction. Note that syntactic category q takes care of

s	$::=$	$a.t$	$\mid\ \tau.t$	$\mid\ \underline{a}.s$	$\mid\ s+s$	
q	$::=$	s	$\mid\ \mathbf{0}$	$\mid\ \overline{a}.t$	$\mid\ q+q$	
r	$::=$	q	$\mid\ C$			*sequential terms*
t	$::=$	r	$\mid\ t \vert t$			*restriction-free terms*
p	$::=$	t	$\mid\ (\nu a)p$			*general terms*

Table 7.2 Syntax for well-formed FNM

the generation of processes that start with output actions or that use a general form of choice (i.e., the summands in $q_1 + q_2$ can start with inputs as well as with outputs). However, this syntax is a bit awkward and so we prefer to stick to the simpler syntax of Section 7.1.1, equipped with the well-formedness predicate above.

Remark 7.1. **(FNM and Multi-CCS)** FNM is a subcalculus of Multi-CCS, which is fully described in [GV15]. Multi-CCS allows for a slightly more generous strong prefixing operator, admitting also output actions as strong prefixes, as well as a more flexible well-formedness predicate. However, the simpler theory we are presenting here is powerful enough for the aims of this chapter, i.e., the representability of all finite P/T nets. □

7.2 Operational LTS Semantics

The FNM labeled transition system $\mathscr{C}_{FNM}$ is the triple $(\mathscr{P}_{FNM}, \mathscr{A}, \longrightarrow)$, where the states are the processes in $\mathscr{P}_{FNM}$, the set of labels is $\mathscr{A} = \{\tau\} \cup \mathscr{L}^* \cdot (\mathscr{L} \cup \overline{\mathscr{L}})$ (ranged over by σ and composed of the invisible action τ and sequences of input actions, possibly ending with an output), and $\longrightarrow \subseteq \mathscr{P}_{FNM} \times \mathscr{A} \times \mathscr{P}_{FNM}$ is the least transition relation generated by the axiom and rules in Table 7.3.

We briefly comment on the rules that are less standard. As discussed in Section 7.1.1, rule (S-pref) allows for the creation of transitions labeled by atomic sequences. In order for $\underline{a}.p$ to make a move, it is necessary that p be able to perform a transition, i.e., the rest of the transaction. Hence, if $p \xrightarrow{\sigma} p'$ then $\underline{a}.p \xrightarrow{a \diamond \sigma} p'$, where $a \diamond \sigma = a$ if $\sigma = \tau$, $a \diamond \sigma = a\sigma$ otherwise. Note that $\underline{a}.\mathbf{0}$ cannot execute any action, as $\mathbf{0}$ is terminated, and this is the reason why we have considered terms of this kind as not well formed. Rule (S-Com) has a side condition on the possible synchronizability of σ_1 and σ_2. Relation $Sync(\sigma_1, \sigma_2, \sigma)$, defined by the axioms of Table 7.4, holds if at least one of the two sequences is a single output action, say $\sigma_1 = \overline{a}$, and the other one is either the complementary action a or an atomic sequence starting with a. Note that it is not possible to synchronize two atomic sequences. This means that, usually, a multi-party synchronization can take place only among one *leader*, i.e., the process performing the atomic sequence of inputs, and as many other com-

$$
\begin{array}{llll}
\text{(Pref)} & \dfrac{}{\mu.p \xrightarrow{\mu} p} & \text{(Cons)} & \dfrac{p \xrightarrow{\sigma} p'}{C \xrightarrow{\sigma} p'} \qquad C \doteq p \\[2ex]
\text{(Sum}_1\text{)} & \dfrac{p \xrightarrow{\sigma} p'}{p+q \xrightarrow{\sigma} p'} & \text{(Sum}_2\text{)} & \dfrac{q \xrightarrow{\sigma} q'}{p+q \xrightarrow{\sigma} q'} \\[2ex]
\text{(Par)} & \dfrac{p \xrightarrow{\sigma} p'}{p \,|\, q \xrightarrow{\sigma} p' \,|\, q} & \text{(Cong)} & \dfrac{p \equiv p' \xrightarrow{\sigma} q' \equiv q}{p \xrightarrow{\sigma} q} \\[2ex]
\text{(S-Pref)} & \dfrac{p \xrightarrow{\sigma} p'}{\underline{a}.p \xrightarrow{a \diamond \sigma} p'} & & a \diamond \sigma = \begin{cases} a & \text{if } \sigma = \tau, \\ a\sigma & \text{otherwise} \end{cases} \\[2ex]
\text{(S-Res)} & \dfrac{p \xrightarrow{\sigma} p'}{(\nu a)p \xrightarrow{\sigma} (\nu a)p'} & & a \notin n(\sigma) \\[2ex]
\text{(S-Com)} & \dfrac{p \xrightarrow{\sigma_1} p' \qquad q \xrightarrow{\sigma_2} q'}{p \,|\, q \xrightarrow{\sigma} p' \,|\, q'} & & Sync(\sigma_1, \sigma_2, \sigma)
\end{array}
$$

Table 7.3 LTS operational rules for FNM

$$
\dfrac{}{Sync(\alpha, \overline{\alpha}, \tau)} \qquad \dfrac{\sigma \neq \varepsilon}{Sync(a\sigma, \overline{a}, \sigma)} \qquad \dfrac{\sigma \neq \varepsilon}{Sync(\overline{a}, a\sigma, \sigma)}
$$

Table 7.4 Synchronization relation *Sync*

E1	$(p \,|\, q) \,|\, r = p \,|\, (q \,|\, r)$
E2	$p \,|\, q = q \,|\, p$

Table 7.5 Axioms generating the structural congruence $\equiv$

ponents (the *servants*) as the length of the sequence, where each servant executes one output action. This is strictly the case for well-formed processes. However, more elaborate forms of synchronization are possible, as illustrated in Example 7.3, for non-well-formed processes. Rule (S-Res) is slightly more general than the corresponding one for FNC, as it requires that no action in σ can be a or $\overline{a}$. By $n(\sigma)$ we denote the set of all actions occurring in σ. Formally: $n(\tau) = \{\tau\}$, $n(a) = \{a\}$, $n(\overline{a}) = \{a\}$, $n(a\sigma) = \{a\} \cup n(\sigma)$. For instance, $n(aca\overline{b}) = \{a, b, c\}$.

Example 7.1. **(Multi-party synchronization)** Consider $(\underline{a}.b.p \,|\, \overline{a}.q) \,|\, \overline{b}.r$. The ternary synchronization $(\underline{a}.b.p \,|\, \overline{a}.q) \,|\, \overline{b}.r \xrightarrow{\tau} (p \,|\, q) \,|\, r$ can take place, as proved in Table 7.6, where the subprocess $\underline{a}.b.p$ acts as the leader.

This idea can be generalized to a multi-party synchronization with n participants as follows. Consider the leader process

$$
p_0 = \underline{a_1}.\underline{a_2}.\dots.\underline{a_{n-2}}.a_{n-1}.p_0'
$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{b.p \xrightarrow{b} p}}{\underline{a}.b.p \xrightarrow{ab} p} \qquad \dfrac{}{\overline{a}.q \xrightarrow{\overline{a}} q}}{\underline{a}.b.p \,|\, \overline{a}.q \xrightarrow{b} p \,|\, q} \qquad \dfrac{}{\overline{b}.r \xrightarrow{\overline{b}} r}}{(\underline{a}.b.p \,|\, \overline{a}.q) \,|\, \overline{b}.r \xrightarrow{\tau} (p \,|\, q) \,|\, r}$$

Table 7.6 Multi-party synchronization among three processes

and the servant processes $\quad p_i = \overline{a_i}.p'_i, \quad$ for $i = 1, \dots, n-1$ with $n \geq 3$.
Then it is not difficult to realize that the transition

$$(\dots((p_0 \,|\, p_1) \,|\, p_2) \,|\, \dots) \,|\, p_{n-1} \xrightarrow{\tau} (\dots((p'_0 \,|\, p'_1) \,|\, p'_2) \,|\, \dots) \,|\, p'_{n-1}$$

is derivable, i.e., the n processes have synchronized in one atomic transaction. □

There is one further rule, called (Cong), which makes use of the structural congruence $\equiv$, induced by the two axioms in Table 7.5. Axioms **E1** and **E2** are for associativity and commutativity, respectively, of the parallel operator. This explains why there is no rule symmetric to (Par): from a transition $q \xrightarrow{\sigma} q'$, by rule (Par), we can derive the transition $q \,|\, p \xrightarrow{\sigma} q' \,|\, p$; then by rule (Cong), also $p \,|\, q \xrightarrow{\sigma} p \,|\, q'$ is derivable; hence, the symmetric rule of (Par) is derivable by (Cong). Rule (Cong) enlarges the set of transitions derivable from a given process p, as the following example shows. The intuition is that, given a process p, a transition is derivable from p if it is derivable from any p' obtained as a rearrangement in any order (or association) of all of its sequential subprocesses.

Example 7.2. **(Associativity and commutativity)** In Example 7.1, we showed that $(\underline{a}.b.p \,|\, \overline{a}.q) \,|\, \overline{b}.r \xrightarrow{\tau} (p \,|\, q) \,|\, r$ is derivable without using rule (Cong). However, if we consider the very similar process $\underline{a}.b.p \,|\, (\overline{a}.q \,|\, \overline{b}.r)$, then we can see that $\underline{a}.b.p$ is able to synchronize with both $\overline{a}.q$ and $\overline{b}.r$ only by using rule (Cong), as follows:

$$\frac{\underline{a}.b.p \,|\, (\overline{a}.q \,|\, \overline{b}.r) \equiv (\underline{a}.b.p \,|\, \overline{a}.q) \,|\, \overline{b}.r \xrightarrow{\tau} (p \,|\, q) \,|\, r \equiv p \,|\, (q \,|\, r)}{\underline{a}.b.p \,|\, (\overline{a}.q \,|\, \overline{b}.r) \xrightarrow{\tau} p \,|\, (q \,|\, r)}$$

If we consider the slightly different variant term $(\underline{a}.b.p \,|\, \overline{b}.r) \,|\, \overline{a}.q$, we easily see that, without rule (Cong), no ternary synchronization is possible, because $Sync(ab, \overline{b}, a)$ does not hold. This example shows that, by using the axioms **E1** and **E2**, it is possible to reorder the servant subcomponents (in this example, subprocesses $\overline{a}.q$ and $\overline{b}.r$) in such a way that the actions they offer are in the order expected by the leader process (in this example, $\underline{a}.b.p$). □

Remark 7.2. **(Commutativity of parallel composition and relation** ***Sync*****)** Thanks to the commutativity axiom **E2** of the structural congruence $\equiv$, the definition of relation *Sync* can be simplified to make it not symmetric in the first two arguments:

the axiom is simply $Sync(a,\overline{a},\tau)$, and the only rule is $Sync(a\sigma,\overline{a},\sigma)$ if $\sigma \neq \varepsilon$. A simplification of this sort will be used in the next chapter for the language NPL. □

Proposition 7.3. *For any FNM process p, if $p \xrightarrow{\sigma} p'$, then $fn(p') \subseteq fn(p)$, and if $\sigma \neq \tau$, then $n(\sigma) \subseteq fn(p)$.*

Proof. *By induction on the proof of transition $p \xrightarrow{\sigma} p'$.* □

It is also possible to prove that for any FNM process p, the set $sort(p)$ is finite. In fact, by iterating Proposition 7.3, we know that any $\sigma \in sort(p)$ (i.e., any σ such that $p \longrightarrow^* p' \xrightarrow{\sigma}$) is such that $n(\sigma) \subseteq fn(p) \cup \{\tau\}$. Since $fn(p)$ is finite for any FNM process p, the thesis follows if we prove that there is an upper bound on the length of any performable σ. Of course, the longest σ cannot be longer than the longest sequence of strong prefixings (plus one for the final normal prefix) syntactically occurring in p (and in all the bodies of the finitely many constants p may use). Therefore, $sort(p)$ is a finite set.

Proposition 7.4. *For any well-formed FNM process p, if $p \xrightarrow{\sigma} p'$, then p' is well formed.*

Proof. *By induction on the proof of transition $p \xrightarrow{\sigma} p'$.* □

Proposition 7.5. *For any FNM process p in syntactic category s and for any $\sigma \in \mathscr{A}$, $p \xrightarrow{\sigma} p'$ if and only if $\sigma \in In(p)$.*

Proof. *By induction on the proof of $p \xrightarrow{\sigma} p'$ and on the definition of $In(p)$.* □

In the last part of this section, we want to prove the theorem justifying the well-formedness predicate of Table 7.1: if a process is well formed, it is not possible to synchronize two atomic sequences, even indirectly. As a matter of fact, two atomic sequences cannot be synchronized directly, as relation $Sync(\sigma_1, \sigma_2, \sigma)$ requires that at least one of σ_1 and σ_2 be a single output action. However, indirectly, two atomic sequences can be synchronized, as discussed in the following example.

Example 7.3. Consider the process $p = (\underline{a}.c.\mathbf{0} \,|\, \overline{a}.\mathbf{0}) \,|\, (\overline{b}.\mathbf{0} \,|\, \underline{b}.\overline{c}.\mathbf{0})$, which is not well formed because of the subterm $\underline{b}.\overline{c}.\mathbf{0}$. Such a process can do a four-way synchronization $(\underline{a}.c.\mathbf{0} \,|\, \overline{a}.\mathbf{0}) \,|\, (\overline{b}.\mathbf{0} \,|\, \underline{b}.\overline{c}.\mathbf{0}) \xrightarrow{\tau} (\mathbf{0} \,|\, \mathbf{0}) \,|\, (\mathbf{0} \,|\, \mathbf{0})$, as proved in Table 7.7. This transition shows a different form of synchronization, where there is no leader, but rather there are two sub-transactions (each one with a sub-leader) synchronizing at the end. Hence, the two atomic sequences ac and $b\overline{c}$ may synchronize indirectly. □

To prove this theorem, some auxiliary results are needed.

Lemma 7.2. *For any FNM processes p and q in syntactic category s, if $p \equiv q$, then $In(p) = In(q)$.*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{c.\mathbf{0} \xrightarrow{c} \mathbf{0}}}{\underline{a}.c.\mathbf{0} \xrightarrow{ac} \mathbf{0}} \qquad \dfrac{}{\overline{a}.\mathbf{0} \xrightarrow{\overline{a}} \mathbf{0}}}{\underline{a}.c.\mathbf{0} \,|\, \overline{a}.\mathbf{0} \xrightarrow{c} \mathbf{0} \,|\, \mathbf{0}} \qquad \dfrac{\dfrac{}{\overline{b}.\mathbf{0} \xrightarrow{\overline{b}} \mathbf{0}} \qquad \dfrac{\dfrac{}{\overline{c}.\mathbf{0} \xrightarrow{\overline{c}} \mathbf{0}}}{\underline{b}.\overline{c}.\mathbf{0} \xrightarrow{b\overline{c}} \mathbf{0}}}{\overline{b}.\mathbf{0} \,|\, \underline{b}.\overline{c}.\mathbf{0} \xrightarrow{\overline{c}} \mathbf{0} \,|\, \mathbf{0}}}{(\underline{a}.c.\mathbf{0} \,|\, \overline{a}.\mathbf{0}) \,|\, (\overline{b}.\mathbf{0} \,|\, \underline{b}.\overline{c}.\mathbf{0}) \xrightarrow{\tau} (\mathbf{0} \,|\, \mathbf{0}) \,|\, (\mathbf{0} \,|\, \mathbf{0})}$$

Table 7.7 Hierarchical multi-party synchronization among four processes

Proof. *By induction on the syntactic definition of p, assuming that $p \equiv q$, we prove that $In(p) = In(q)$. We proceed by case analysis on the shape of p. As $p \equiv q$, if $p = \mathbf{0}$, then $q = \mathbf{0}$, and the thesis follows trivially. If $p = \mu.p_1$, then $q = \mu.q_1$, with $p_1 \equiv q_1$; in this case, $In(p) = \{\mu\} = In(q)$, as required. If $p = \underline{a}.p_1$, then $q = \underline{a}.q_1$, with $p_1 \equiv q_1$; by induction, we can assume that $In(p_1) = In(q_1)$; hence, $In(p) = a \diamond In(p_1) = a \diamond In(q_1) = In(q)$, as required. If $p = p_1 + p_2$, then $q = q_1 + q_2$ with $p_i \equiv q_i$, for $i = 1,2$. By induction, $In(p_i) = In(q_i)$ for $i = 1,2$, and so $In(p) = In(p_1) \cup In(p_2) = In(q_1) \cup In(q_2) = In(q)$, as required.* □

Proposition 7.6. *If $wf(p)$ and $p \equiv q$, then $wf(q)$.*

Proof. *By induction on the proof of $p \equiv q$. First, we have to prove the thesis for the two axioms of Table 7.5.*

For axiom **E1**, *$wf(p_1 \,|\, (p_2 \,|\, p_3))$ holds if $wf(p_1)$, $wf(p_2 \,|\, p_3)$, and $wf(p_2 \,|\, p_3)$ holds if $wf(p_2)$, $wf(p_3)$. Hence, $wf(p_1 \,|\, p_2)$ holds as $wf(p_1)$ and $wf(p_2)$ hold, and, finally, also $wf((p_1 \,|\, p_2) \,|\, p_3)$ holds, because $wf(p_1 \,|\, p_2)$ and $wf(p_3)$ hold. The case of axiom* **E2** *is trivial.*

Then, one has to prove the thesis for all the rules of equational deduction (reflexivity, symmetry, transitivity, substitutivity and instantiation; see, e.g., [GV15], Section 4.3). The only nontrivial case is about substitutivity; we discuss two cases only. For strong prefixing, we assume we have $p = \underline{a}.p_1$ and $q = \underline{a}.q_1$, with $p_1 \equiv q_1$ and $wf(p)$; since $wf(p)$ holds, also $wf(p_1)$ holds and $\emptyset \neq In(p_1) \subseteq \mathscr{L}^+ \cup \{\tau\}$. By induction, we have that $wf(q_1)$; since p_1 and q_1 are in syntactic category s and $p_1 \equiv q_1$, by Lemma 7.2 we have that $In(p_1) = In(q_1)$, and so also $\emptyset \neq In(q_1) \subseteq \mathscr{L}^+ \cup \{\tau\}$; therefore, $wf(\underline{a}.q_1)$ holds, as required. For parallel composition, we assume we have $p = p_1 \,|\, r$ and $q = p_2 \,|\, r$, with $p_1 \equiv p_2$ and $wf(p)$; therefore, $wf(p_1)$ and $wf(r)$ hold. By induction, we have that $wf(p_2)$ holds. Therefore, $wf(p_2 \,|\, r)$ holds, as required. □

Proposition 7.7. *For any well-formed FNM process p, if $p \xrightarrow{\sigma} p'$, then $\sigma \in \{\tau\} \cup \overline{\mathscr{L}} \cup \mathscr{L}^+$. Moreover, if $\sigma \in \overline{\mathscr{L}}$, then the proof tree for $p \xrightarrow{\sigma} p'$ does not contain any transition with a label different from σ itself.*

Proof. *By induction on the proof of transition $p \xrightarrow{\sigma} p'$. We proceed by case analysis. The base case is when $p = \mu.p'$; in such a case, $\mu.p' \xrightarrow{\mu} p'$ is derivable, so that $\mu \in \{\tau\} \cup \overline{\mathscr{L}} \cup \mathscr{L}^+$ and, if $\mu = \overline{a}$, then the proof tree of the transition does not contain any transition with a label different from σ itself.*

If $p = p_1 + p_2$, then $p \xrightarrow{\sigma} p'$ is due to either rule (Sum$_1$) or (Sum$_2$); w.l.o.g., assume that (Sum$_1$) was used; in this case, the premise transition must be $p_1 \xrightarrow{\sigma} p'$; since wf$(p)$ holds only if also wf(p_1) holds, induction can be used to conclude that all the requirements of the proposition are satisfied for σ.

The cases when $p = C$ or when $p = (\nu a)p_1$ are analogous to the above, hence omitted.

If $p = \underline{a}.p_1$, then $p \xrightarrow{\sigma} p'$ is due to rule (S-Pref), with premise $p_1 \xrightarrow{\sigma'} p'$ such that $a \diamond \sigma' = \sigma$. By Proposition 7.5, $\sigma' \in In(p_1)$; wf(p) holds only if also wf(p_1) holds and, moreover, $In(p_1) \subseteq \{\tau\} \cup \mathscr{L}^+$; therefore, $\sigma \in \mathscr{L}^+$, and so the requirements of the proposition are satisfied for σ.

If $p = p_1 \,|\, p_2$, then $p \xrightarrow{\sigma} p'$ is due either to rule (Par) or to rule (S-Com). In the former case, the premise transition is $p_1 \xrightarrow{\sigma} p_1'$ and $p' = p_1' \,|\, p_2$. As wf(p) holds only if also wf(p_1) holds, induction can be used to conclude that all the requirements of the proposition are satisfied for σ. In the latter case, the two premise transitions are $p_1 \xrightarrow{\sigma_1} p_1'$, $p_2 \xrightarrow{\sigma_2} p_2'$, with Sync$(\sigma_1, \sigma_2, \sigma)$ and $p' = p_1' \,|\, p_2'$. As wf(p) holds only if also wf(p_1) and wf(p_2) hold, induction can be used to conclude that σ_1 and σ_2 satisfy all the requirements of the proposition. In particular, by definition of Sync, either σ_1 or σ_2 is $\overline{a}$ for some $a \in \mathscr{L}$, and the other sequence is the complementary action a or an atomic sequence starting with a; w.l.o.g., assume $\sigma_1 = \overline{a}$ and $\sigma_2 = a$, if $\sigma = \tau$, or $\sigma_2 = a\sigma$. In the former case, the requirements of the proposition are trivially satisfied for $\sigma = \tau$; in the latter case, $\sigma_2 \in \mathscr{L}^+$, so that also $\sigma \in \mathscr{L}^+$, and so also in this subcase all the requirements of the proposition are satisifed for σ.

If the last rule used in deriving $p \xrightarrow{\sigma} p'$ is (Cong), then there exist q and q' such that $q \equiv p$, $q' \equiv p'$ and $q \xrightarrow{\sigma} q'$. By Proposition 7.6, also wf(q) holds, and so, by induction, σ satisfies all the requirements of the proposition. □

In the proposition above, the requirement that if $\sigma \in \overline{\mathscr{L}}$ then the proof tree for $p \xrightarrow{\sigma} p'$ does not contain any transition with a label different from σ itself explains that no atomic sequence was performed in order to derive $p \xrightarrow{\sigma} p'$. Therefore, the following theorem follows easily.

Theorem 7.2. (Well-formedness implies no synchronization of atomic sequences) *If wf(p) and $p \xrightarrow{\sigma} p'$, then in the proof of such a transition it is not possible that two atomic sequences are synchronized, even indirectly.*

Proof. *By induction on the proof of $p \xrightarrow{\sigma} p'$. All the cases are trivial — if the thesis holds for the premise transition, then it holds also for the conclusion — except when rule (S-Com) is used. In this case, assume we have $p = p_1 \,|\, p_2$, $p_1 \xrightarrow{\sigma_1} p_1'$, $p_2 \xrightarrow{\sigma_2} p_2'$, Sync$(\sigma_1, \sigma_2, \sigma)$ and $p' = p_1' \,|\, p_2'$. Since wf$(p_1 \,|\, p_2)$, then also wf(p_1) and wf(p_2). Hence, by induction, we can assume that the thesis holds for the two premise transitions. So, it remains to prove that, since wf$(p_1 \,|\, p_2)$ holds, no indirect synchronization of two atomic sequences is performed in the last proof step of transition $p_1 \,|\, p_2 \xrightarrow{\sigma} p_1' \,|\, p_2'$. By definition of Sync, at least one of σ_1 and σ_2 is a single output action. W.l.o.g., assume $\sigma_2 = \overline{a}$. Since wf(p_2) holds, by Proposition 7.7, transition*

$p_2 \xrightarrow{\sigma_2} p_2'$ cannot be due to the execution of an atomic sequence, so that the conclusion $p \xrightarrow{\sigma} p'$ does not synchronize two atomic sequences. □

Remark 7.3. The operational LTS rules of Table 7.3 can be used to give an operational semantics also to extended FNM processes. There is only one caution: of the transitions derivable from the rules, we should take only those that are labeled over $\mathscr{A}$, thus discarding those transitions labeled with restricted actions or sequences containing at least one restricted action. However, derivable transitions labeled with (sequences containing) complementary restricted actions give rise, by rule (S-Com), to a synchronized transition, which is to be taken, if labeled τ or with a shorter sequence containing no restricted actions. This issue will be clarified better in Section 7.4.2, about FNM net transitions. □

7.2.1 Expressiveness

From a syntactic point of view, FNC is a proper subcalculus of FNM. From a semantic point of view, we can prove, following the same steps as the proof in Section 6.2.1 of [GV15], that FNM is a *conservative extension* of FNC: for any FNC process p, the rooted LTS for p generated by means of the operational rules in Table 6.5 is bisimilar (but, in general, not isomorphic[1]) to the rooted LTS for p generated by means of the operational rules in Table 7.3.

Just looking at the LTS interleaving semantics, we have that FNM is more expressive than FNC. Let us consider the simple FNM process $p = \underline{a}.b.\mathbf{0}$, whose set of traces is $Tr(p) = \{\varepsilon, ab\}$. It is not difficult to realize that no FNC process q is trace equivalent to p; in fact, the only natural candidate FNC process is $q = a.b.\mathbf{0}$, whose set of traces is $Tr(q) = \{\varepsilon, a, ab\} \neq Tr(p)$. Note that $Tr(q)$ is prefix-closed, and this property holds for the set of traces $Tr(r)$ of any FNC process r; on the contrary, $Tr(p)$ is not prefix-closed. Moreover, when considering the net semantics we are going to define for FNM, we will see that well-formed FNM is able to represent all finite P/T nets, while FNC is able to represent all finite CCS nets, only.

However, if we are interested in weak completed traces only, then the result is surprisingly different: for any well-formed, output-closed FNM process p, we can find an output-closed FNC process q such that $WCtr(p) = WCtr(q)$.[2]

Definition 7.1. (FNM language) A language $L \subseteq \mathscr{L}^*$ is an *FNM language* if there exists a well-formed, output-closed FNM process p such that the set of its weak completed traces is L, i.e., $WCTr(p) = L$. □

[1] For instance, $a.(b.\mathbf{0} \,|\, c.\mathbf{0})$ has one unique transition labeled a, reaching $b.\mathbf{0} \,|\, c.\mathbf{0}$, in the LTS for FNC, while it has one additional transition in the LTS for FNM, namely $a.(b.\mathbf{0} \,|\, c.\mathbf{0}) \xrightarrow{a} c.\mathbf{0} \,|\, b.\mathbf{0}$, due to rule (Cong).

[2] The fact that p and q have the same set of weak completed traces does not mean that p and q are weak completed trace equivalent, as they may be not weak trace equivalent.

$\llbracket (\nu L_1)t \rrbracket = (\nu L_2)(\llbracket t \rrbracket^p \mid Sem)$	$L_2 = L_1 \cup \{p, v\}$	$Sem \doteq p.v.Sem$
$\llbracket \mathbf{0} \rrbracket^p = \mathbf{0}$	$\llbracket C \rrbracket^p = C'$	where $C' \doteq \llbracket q \rrbracket^p$ if $C \doteq q$
$\llbracket a.q \rrbracket^p = \overline{p}.a.\overline{v}.(\llbracket q \rrbracket^p)$	$\llbracket \overline{a}.q \rrbracket^p = \overline{a}.(\llbracket q \rrbracket^p)$	$\llbracket \tau.q \rrbracket^p = \tau.(\llbracket q \rrbracket^p)$
$\llbracket \underline{a}.q \rrbracket^p = \overline{p}.a.(\llbracket q \rrbracket^v)$	$\llbracket q_1 + q_2 \rrbracket^p = \llbracket q_1 \rrbracket^p + \llbracket q_2 \rrbracket^p$	$\llbracket q_1 \mid q_2 \rrbracket^p = \llbracket q_1 \rrbracket^p \mid \llbracket q_2 \rrbracket^p$
$\llbracket \mu.q \rrbracket^v = \mu.\overline{v}.(\llbracket q \rrbracket^p)$	$\llbracket \underline{a}.q \rrbracket^v = a.(\llbracket q \rrbracket^v)$	$\llbracket q_1 + q_2 \rrbracket^v = \llbracket q_1 \rrbracket^v + \llbracket q_2 \rrbracket^v$

Table 7.8 Encoding well-formed, output-closed FNM into output-closed FNC

Theorem 7.3. (FNM languages = FNC languages) *The class of FNM languages coincides with the class of FNC languages.* □

In order to prove this theorem, we first observe that, since FNM conservatively extends FNC (up to $\sim$), the class of FNC languages is a subclass of the class of FNM languages. To prove the reverse inclusion, we propose an encoding from well-formed, output-closed FNM processes to output-closed FNC processes, formally defined in Table 7.8, which preserves the weak completed traces. Given a well-formed, output-closed FNM process $(\nu L_1)t$, where t is a restriction-free process, its encoding $\llbracket (\nu L_1)t \rrbracket$ is the output-closed FNC process $(\nu L_2)(\llbracket t \rrbracket^p \mid Sem)$, where L_2 and Sem are defined in Table 7.8, which simulates the execution of t by forcing mutual exclusion of its input actions or input atomic sequences, by means of the semaphore Sem. In fact, $\llbracket t \rrbracket^p$ is the FNC term where each input action, as well as each input atomic sequence, is preceded by the semaphore request $\overline{p}$ and followed by the semaphore release $\overline{v}$. The effect is that atomic sequences cannot be interleaved with concurrently executable input actions or sequences.

As an example, consider the FNM process $q = \underline{a}.b.\mathbf{0} \mid c.\mathbf{0}$, whose set of completed traces is $WCtr(q) = \{abc, cab\}$; its encoding $\llbracket q \rrbracket = (\nu L)((\overline{p}.a.b.\overline{v}.\mathbf{0} \mid \overline{p}.c.\overline{v}.\mathbf{0}) \mid Sem)$ (where $L = \{p, v\}$) can perform the same weak completed traces; note that the weak trace acb is not a weak completed trace for $\llbracket q \rrbracket$ because of the mutual exclusion mechanism.

The output actions are not guarded by $\overline{p}$, because they must be freely executable: since the process is output-closed, such actions can only be used for synchronization with the currently active input action or sequence. For instance, the well-formed, output-closed FNM process $r = (\nu a)((\underline{a}.b.\mathbf{0} \mid \underline{a}.c.\mathbf{0}) \mid \overline{a}.\mathbf{0})$, whose set of weak completed traces is $WCtr(r) = \{b, c\}$, is mapped to the FNC process

$$\llbracket r \rrbracket = (\nu L)(((\overline{p}.a.b.\overline{v}.\mathbf{0} \mid \overline{p}.a.c.\overline{v}.\mathbf{0}) \mid \overline{a}.\mathbf{0}) \mid Sem),$$

where $L = \{p, v, a\}$, exhibiting the same set $\{b, c\}$ of weak completed traces.

Note that if the FNM process is not output-closed, then the encoding may be incorrect, as the output actions can be interleaved with the atomic sequences. For instance, $q = \underline{a}.b.\mathbf{0} \mid \overline{a}.\mathbf{0}$ is such that $WCtr(q) = \{ab\overline{a}, \overline{a}ab, b\}$; however, its encoding $\llbracket q \rrbracket = (\nu L)((\overline{p}.a.b.\overline{v}.\mathbf{0} \mid \overline{a}.\mathbf{0}) \mid Sem)$ (where $L = \{p, v\}$) can also perform the completed trace $a\overline{a}b$. Even when the FNM process is not well formed, the encoding may

be incorrect. For instance, the non-well-formed process $r = (\nu a)((\underline{a}.\overline{b}.\mathbf{0} \,|\, \overline{a}.\mathbf{0}) \,|\, b.c.\mathbf{0})$ is such that $WCtr(r) = \{\overline{b}bc, bc\overline{b}, b\overline{b}c, c\}$; however, its encoding
$[\![r]\!] = (\nu L)(((\overline{p}.a.\overline{b}.\overline{v}.\mathbf{0} \,|\, \overline{a}.\mathbf{0}) \,|\, \overline{p}.b.\overline{v}.\overline{p}.c.\overline{v}.\mathbf{0}) \,|\, Sem)$
where $L = \{p, v, a\}$, cannot perform the completed trace c, because $[\![r]\!]$, after silently reaching the state $(\nu L)(((\overline{b}.\overline{v}.\mathbf{0} \,|\, \mathbf{0}) \,|\, \overline{p}.b.\overline{v}.\overline{p}.c.\overline{v}.\mathbf{0}) \,|\, v.Sem)$, cannot perform the synchronization on b because $\overline{b}$ precedes the release action $\overline{v}$, so that its corresponding input action b is unable to perform the preliminary trigger action $\overline{p}$. Finally, the encoding may be undefined for some non-well-formed processes; for instance, the encoding of $\underline{a}.\mathbf{0}$ is not properly defined, as the case $[\![\mathbf{0}]\!]^v$ is not specified.

7.2.2 Congruence Problem

A bisimulation over $\mathscr{C}_{FNM} = (\mathscr{P}_{FNM}, \mathscr{A}, \longrightarrow)$ is a relation $R \subseteq \mathscr{P}_{FNM} \times \mathscr{P}_{FNM}$ such that if $(q_1, q_2) \in R$ then for all $\sigma \in \mathscr{A}$

- $\forall q_1'$ such that $q_1 \xrightarrow{\sigma} q_1'$, $\exists q_2'$ such that $q_2 \xrightarrow{\sigma} q_2'$ and $(q_1', q_2') \in R$,
- $\forall q_2'$ such that $q_2 \xrightarrow{\sigma} q_2'$, $\exists q_1'$ such that $q_1 \xrightarrow{\sigma} q_1'$ and $(q_1', q_2') \in R$.

Two FNM processes p and q are *interleaving bisimilar*, denoted $p \sim q$, if there exists a bisimulation $R \subseteq \mathscr{P}_{FNM} \times \mathscr{P}_{FNM}$ such that $(p, q) \in R$.

Proposition 7.8. *Let $p, q \in \mathscr{P}_{FNM}$ be two FNM processes. If $p \equiv q$, then $p \sim q$.*

Proof. It is enough to check that relation $R = \{(p,q) \;\big|\; p \equiv q\}$ is a bisimulation. If $(p,q) \in R$ and $p \xrightarrow{\sigma} p'$, then by rule (Cong) also $q \xrightarrow{\sigma} p'$ and $(p', p') \in R$. Symmetrically, if q moves first. □

Interleaving bisimilarity is a congruence for almost all the operators of FNM, in particular for strong prefixing. The congruence proofs are standard and similar to the corresponding proofs in the previous chapters, the only difference being that the proposed relations for action prefixing and choice are not bisimulations, rather bisimulations up to $\equiv$ (Definition 2.14); hence, these relations are omitted.

Proposition 7.9. *Given two FNM processes p and q in syntactic category s, if $p \sim q$, then the following hold:*
(i) $\underline{a}.p \sim \underline{a}.q$ for all $a \in \mathscr{L}$,
(ii) $p + r \sim q + r$ for all r in syntactic category s.

Proof. We prove only (i). Let R be a bisimulation such that $(p,q) \in R$. It is easy to check that relation $R' = \{(\underline{a}.p, \underline{a}.q)\} \cup R$ is a bisimulation. □

Proposition 7.10. *Given two restriction-free FNM processes p and q, if $p \sim q$, then $\mu.p \sim \mu.q$ for all $\mu \in Act$.* □

Proposition 7.11. *Given two FNM processes p and q, if $p \sim q$, then $(\nu a)p \sim (\nu a)q$ for all $a \in \mathscr{L}$.* □

$$\text{(Cong)}\ \dfrac{\text{(S-Com)}\ \dfrac{\text{(Pref)}\ \dfrac{}{\overline{a}.\mathbf{0} \xrightarrow{\overline{a}} \mathbf{0}} \qquad \text{(S-Com)}\ \dfrac{\text{(Pref)}\ \dfrac{}{\overline{a}.\mathbf{0} \xrightarrow{\overline{a}} \mathbf{0}} \qquad \text{(S-Pref)}\ \dfrac{\text{(S-Pref)}\ \dfrac{\text{(Pref)}\ \dfrac{}{c.\mathbf{0} \xrightarrow{c} \mathbf{0}}}{\underline{a}.c.\mathbf{0} \xrightarrow{ac} \mathbf{0}}}{\underline{a}.\underline{a}.c.\mathbf{0} \xrightarrow{aac} \mathbf{0}}}{\overline{a}.\mathbf{0}\,|\,\underline{a}.\underline{a}.c.\mathbf{0} \xrightarrow{ac} \mathbf{0}\,|\,\mathbf{0}}}{(\overline{a}.\mathbf{0}\,|\,\overline{a}.\mathbf{0})\,|\,\underline{a}.\underline{a}.c.\mathbf{0} \equiv \overline{a}.\mathbf{0}\,|\,(\overline{a}.\mathbf{0}\,|\,\underline{a}.\underline{a}.c.\mathbf{0}) \xrightarrow{c} \mathbf{0}\,|\,(\mathbf{0}\,|\,\mathbf{0}) \equiv (\mathbf{0}\,|\,\mathbf{0})\,|\,\mathbf{0}}}{(\overline{a}.\mathbf{0}\,|\,\overline{a}.\mathbf{0})\,|\,\underline{a}.\underline{a}.c.\mathbf{0} \xrightarrow{c} (\mathbf{0}\,|\,\mathbf{0})\,|\,\mathbf{0}}$$

Table 7.9 Proof of a ternary synchronization

Unfortunately, $\sim$ is not a congruence for parallel composition, as the following example shows.

Example 7.4. **(No congruence for parallel composition)** Consider the FNC processes $r = \overline{a}.\overline{a}.\mathbf{0}$ and $t = \overline{a}.\mathbf{0}\,|\,\overline{a}.\mathbf{0}$. Clearly, r is bisimilar to t, $r \sim t$. However, if we consider the FNM context $\mathscr{C}[-] = -\,|\,\underline{a}.\underline{a}.c.\mathbf{0}$, we get that $\mathscr{C}[r] \not\sim \mathscr{C}[t]$, because the latter can execute c, as shown in Table 7.9, while $\mathscr{C}[r]$ cannot. The reason for this difference is that the process $\underline{a}.\underline{a}.c.\mathbf{0}$ can react with a number of concurrently active components equal to the length of the trace it can perform. Hence, a congruence semantics for the operator of parallel composition needs to distinguish between r and t on the basis of their different degrees of parallelism. In other words, the interleaving semantics is to be replaced by a *truly concurrent* semantics, as illustrated in the following section. □

7.3 Step Semantics

The step operational semantics for FNM is given by the step transition system $\mathscr{C}^{step}_{FNM} = (\mathscr{P}_{FNM}, \mathscr{B}, \longrightarrow_s)$, where $\mathscr{B} = \mathscr{M}_{fin}(\mathscr{A})$ — i.e., the set of all the finite multisets over $\mathscr{A}$ — is the set of labels (ranged over by M, possibly indexed), and $\longrightarrow_s \subseteq \mathscr{P}_{FNM} \times \mathscr{B} \times \mathscr{P}_{FNM}$ is the least transition relation generated by the rules in Table 7.10.

Axiom (Pref^s) states that $\mu.p$ can perform the singleton step $\{\mu\}$, reaching p. Rule (S-Pref^s) assumes that the transition in the premise is sequential, i.e., composed of one single action or sequence; this is because p is a sequential process, and so it cannot execute multiple actions or sequences at the same time. Analogously, since the $+$ operator composes only sequential processes, it is assumed in the premise that the label is a singleton; similarly for rule (Cons^s). Rule (Res^s) requires that M contains no occurrences of action a or $\overline{a}$ in any $\sigma \in M$; we denote by $n(M)$ the set $\bigcup_{\sigma \in M} n(\sigma)$. The highlight of this semantics is rule (S-Com^s): it allows for the generation of multisets as labels, by using an additional auxiliary relation

$$(\mathrm{Pref}^s)\ \frac{}{\mu.p \xrightarrow{\{\mu\}}_s p} \qquad (\mathrm{Cons}^s)\ \frac{p \xrightarrow{\{\sigma\}}_s p'}{C \xrightarrow{\{\sigma\}}_s p'} \quad C \doteq p$$

$$(\text{S-Pref}^s)\ \frac{p \xrightarrow{\{\sigma\}}_s p'}{\underline{a}.p \xrightarrow{\{a\diamond\sigma\}}_s p'} \qquad (\mathrm{Res}^s)\ \frac{p \xrightarrow{M}_s p'}{(\nu a)p \xrightarrow{M}_s (\nu a)p'} \ a \notin n(M)$$

$$(\mathrm{Sum}_1^s)\ \frac{p \xrightarrow{\{\sigma\}}_s p'}{p+q \xrightarrow{\{\sigma\}}_s p'} \qquad (\mathrm{Sum}_2^s)\ \frac{q \xrightarrow{\{\sigma\}}_s q'}{p+q \xrightarrow{\{\sigma\}}_s q'}$$

$$(\mathrm{Par}_1^s)\ \frac{p \xrightarrow{M}_s p'}{p\,|\,q \xrightarrow{M}_s p'\,|\,q} \qquad (\mathrm{Par}_2^s)\ \frac{q \xrightarrow{M}_s q'}{p\,|\,q \xrightarrow{M}_s p\,|\,q'}$$

$$(\text{S-Com}^s)\ \frac{p \xrightarrow{M_1}_s p' \qquad q \xrightarrow{M_2}_s q'}{p\,|\,q \xrightarrow{M}_s p'\,|\,q'} \ MSync(M_1 \oplus M_2, M)$$

Table 7.10 Step operational rules for FNM

$$\frac{}{MSync(M,M)} \qquad \frac{Sync(\sigma_1,\sigma_2,\sigma) \quad MSync(M \oplus \{\sigma\}, M')}{MSync(M \oplus \{\sigma_1,\sigma_2\}, M')}$$

Table 7.11 Step synchronization relation

MSync, defined in Table 7.11, where $\oplus$ denotes multiset union. The intuition behind the definitions of rule (S-Com^s) and *MSync* is that, whenever two parallel processes p and q perform steps M_1 and M_2, respectively, then we can put all the sequences together — yielding $M_1 \oplus M_2$ — and see whether $MSync(M_1 \oplus M_2, \overline{M})$ holds. The resulting multiset $\overline{M}$ may be just $M_1 \oplus M_2$ (hence no synchronization takes place), according to axiom $MSync(M,M)$, or the multiset M' we obtain from the application of the rule: select σ_1 and σ_2 from $M_1 \oplus M_2$, synchronize them to produce σ, then recursively apply *MSync* to $\{\sigma\} \oplus (M_1 \oplus M_2) \ominus \{\sigma_1, \sigma_2\}$ to obtain M'. This procedure of synchronizing sequences may go on until pairs of synchronizable sequences can be found, but may also stop at any moment due to the axiom $MSync(M,M)$.

It is possible to prove that $\mathscr{C}_{FNM}^{step}$ is *fully concurrent*, i.e., for any $p \in \mathscr{P}_{FNM}$, if $p \xrightarrow{M_1 \oplus M_2}_s p'$, with $M_1 \neq \emptyset \neq M_2$, then there exist q_1 and q_2 such that $p \xrightarrow{M_1}_s q_1 \xrightarrow{M_2}_s p'$ and $p \xrightarrow{M_2}_s q_2 \xrightarrow{M_1}_s p'$.

An example of step bisimilar processes is given by the two processes $a.\mathbf{0}\,|\,b.\mathbf{0}$ and $(\nu c)((a.\mathbf{0} + \overline{c}.\mathbf{0})\,|\,(b.\mathbf{0} + \underline{c}.a.b.\mathbf{0}))$.[3] It is not difficult to check that they are step bisimulation equivalent. On the one hand, $a.\mathbf{0}\,|\,b.\mathbf{0}$ can do, besides the obvious sequential transitions $a.\mathbf{0}\,|\,b.\mathbf{0} \xrightarrow{\{a\}}_s \mathbf{0}\,|\,b.\mathbf{0} \xrightarrow{\{b\}}_s \mathbf{0}\,|\,\mathbf{0}$ and $a.\mathbf{0}\,|\,b.\mathbf{0} \xrightarrow{\{b\}}_s a.\mathbf{0}\,|\,\mathbf{0} \xrightarrow{\{a\}}_s \mathbf{0}\,|\,\mathbf{0}$, also the parallel transition proven below.

[3] The FNM process $(\nu c)((a.\mathbf{0} + \overline{c}.\mathbf{0})\,|\,(b.\mathbf{0} + \underline{c}.a.b.\mathbf{0}))$ can be equivalently expressed in a CCS-like calculus with *unguarded* sum as $(a.\mathbf{0}\,|\,b.\mathbf{0}) + a.b.\mathbf{0}$.

$$(\text{Res}^s)\,\dfrac{(\text{S-Com}^s)\,\dfrac{(\text{Sum}^s_1)\,\dfrac{(\text{Pref}^s)\,\dfrac{}{a.\mathbf{0}\xrightarrow{\{a\}}_s\mathbf{0}}}{a.\mathbf{0}+\overline{c}.\mathbf{0}\xrightarrow{\{a\}}_s\mathbf{0}} \qquad (\text{Sum}^s_1)\,\dfrac{(\text{Pref}^s)\,\dfrac{}{b.\mathbf{0}\xrightarrow{\{b\}}_s\mathbf{0}}}{b.\mathbf{0}+\underline{c}.a.b.\mathbf{0}\xrightarrow{\{b\}}_s\mathbf{0}}}{(a.\mathbf{0}+\overline{c}.\mathbf{0})\,|\,(b.\mathbf{0}+\underline{c}.a.b.\mathbf{0})\xrightarrow{\{a,b\}}_s\mathbf{0}\,|\,\mathbf{0}}\ \mathit{MSync}(\{a,b\},\{a,b\})}{(\nu c)((a.\mathbf{0}+\overline{c}.\mathbf{0})\,|\,(b.\mathbf{0}+\underline{c}.a.b.\mathbf{0}))\xrightarrow{\{a,b\}}_s(\nu c)(\mathbf{0}\,|\,\mathbf{0})}$$

Table 7.12 Proof of a parallel step transition

$$(\text{S-Com}^s)\,\dfrac{(\text{Pref}^s)\,\dfrac{}{a.\mathbf{0}\xrightarrow{\{a\}}_s\mathbf{0}} \qquad (\text{Pref}^s)\,\dfrac{}{b.\mathbf{0}\xrightarrow{\{b\}}_s\mathbf{0}}}{a.\mathbf{0}\,|\,b.\mathbf{0}\xrightarrow{\{a,b\}}_s\mathbf{0}\,|\,\mathbf{0}}\ \mathit{MSync}(\{a,b\},\{a,b\})$$

On the other hand, $(\nu c)((a.\mathbf{0}+\overline{c}.\mathbf{0})\,|\,(b.\mathbf{0}+\underline{c}.a.b.\mathbf{0}))$ can do the obvious sequential transitions

$$(\nu c)((a.\mathbf{0}+\overline{c}.\mathbf{0})\,|\,(b.\mathbf{0}+\underline{c}.a.b.\mathbf{0}))\xrightarrow{\{a\}}_s(\nu c)(\mathbf{0}\,|\,(b.\mathbf{0}+\underline{c}.a.b.\mathbf{0}))\xrightarrow{\{b\}}_s(\nu c)(\mathbf{0}\,|\,\mathbf{0}),$$
$$(\nu c)((a.\mathbf{0}+\overline{c}.\mathbf{0})\,|\,(b.\mathbf{0}+\underline{c}.a.b.\mathbf{0}))\xrightarrow{\{b\}}_s(\nu c)((a.\mathbf{0}+\overline{c}.\mathbf{0})\,|\,\mathbf{0})\xrightarrow{\{a\}}_s(\nu c)(\mathbf{0}\,|\,\mathbf{0}),$$

and the sequential transitions triggered by the bound strong prefix c

$$(\nu c)((a.\mathbf{0}+\overline{c}.\mathbf{0})\,|\,(b.\mathbf{0}+\underline{c}.a.b.\mathbf{0}))\xrightarrow{\{a\}}_s(\nu c)(\mathbf{0}\,|\,b.\mathbf{0})\xrightarrow{\{b\}}_s(\nu c)(\mathbf{0}\,|\,\mathbf{0}),$$

and also the parallel step $(\nu c)((a.\mathbf{0}+\overline{c}.\mathbf{0})\,|\,(b.\mathbf{0}+\underline{c}.a.b.\mathbf{0}))\xrightarrow{\{a,b\}}_s(\nu c)(\mathbf{0}\,|\,\mathbf{0})$, as proven in Table 7.12.

This example shows that a process can be saturated with additional subterms expressing some linearizations of its step behavior, without changing its step equivalence class. Therefore, also $(\nu c)((a.\mathbf{0}+\overline{c}.\mathbf{0})\,|\,((b.\mathbf{0}+\underline{c}.a.b.\mathbf{0})+\underline{c}.b.a.\mathbf{0}))$ is step bisimilar to $a.\mathbf{0}\,|\,b.\mathbf{0}$.

It is interesting to observe that the step operational rules in Table 7.10 do not make use of the structural congruence $\equiv$. The same operational effect of rule (Cong) is here ensured by relation *MSync*, which allows for multiple synchronization of concurrently active subprocesses.

Example 7.5. **(Why is structural congruence not needed?)** Continuing Example 7.2, consider process $Q=\underline{a}.b.p\,|\,(\overline{b}.r\,|\,\overline{a}.q)$. We have already noticed that in the interleaving transition system $Q\not\xrightarrow{\tau}p\,|\,(r\,|\,q)$ if rule (Cong) is not available. However, such a process can do a ternary synchronization step as follows:

$$\dfrac{\dfrac{\dfrac{}{b.p\xrightarrow{\{b\}}_s p}}{\underline{a}.b.p\xrightarrow{\{ab\}}_s p}\qquad\dfrac{\dfrac{}{\overline{b}.r\xrightarrow{\{\overline{b}\}}_s r}\quad\dfrac{}{\overline{a}.q\xrightarrow{\{\overline{a}\}}_s q}}{\overline{b}.r\,|\,\overline{a}.q\xrightarrow{\{\overline{a},\overline{b}\}}_s r\,|\,q}\ \mathit{MSync}(\{\overline{a},\overline{b}\},\{\overline{a},\overline{b}\})}{\underline{a}.b.p\,|\,(\overline{b}.r\,|\,\overline{a}.q)\xrightarrow{\{\tau\}}_s p\,|\,(r\,|\,q)}\ \mathit{MSync}(\{ab,\overline{a},\overline{b}\},\{\tau\})$$

where the proof for $MSync(\{ab, \overline{a}, \overline{b}\}, \{\tau\})$ is

$$\dfrac{\dfrac{}{Sync(ab, \overline{a}, b)} \qquad \dfrac{\dfrac{}{Sync(b, \overline{b}, \tau)} \qquad \dfrac{}{MSync(\{\tau\}, \{\tau\})}}{MSync(\{b, \overline{b}\}, \{\tau\})}}{MSync(\{ab, \overline{a}, \overline{b}\}, \{\tau\})}$$

Note that the proof of $MSync(\{ab, \overline{b}, \overline{a}\}, \{\tau\})$ gives a precise algorithm to rearrange the three sequential subprocesses of Q to obtain a process Q' in such a way that no instance of rule (Cong) is needed in deriving the interleaving ternary synchronization; first, the subprocesses originating ab and $\overline{a}$ are to be contiguous: $\underline{a}.b.p \,|\, \overline{a}.q$ would produce b. Then, we compose this system with the subprocess performing $\overline{b}$, yielding $Q' = (\underline{a}.b.p \,|\, \overline{a}.q) \,|\, \overline{b}.r$. Indeed, $Q' \xrightarrow{\tau} (p \,|\, r) \,|\, q$ and its proof makes no use of rule (Cong). □

7.3.1 Step Bisimilarity Implies Interleaving Bisimilarity

By *step bisimilarity*, denoted $\sim_{step}$, we mean ordinary bisimulation equivalence over the step transition system of FNM. This section is devoted to proving that step bisimilarity $\sim_{step}$ is more discriminating than interleaving bisimilarity $\sim$. The proof is a bit technical and needs some auxiliary lemmata. First, we list some properties of relation *MSync*.

Lemma 7.3. (Additivity) *For all multisets $M_1, M_2, N \in \mathscr{B}$, we have $MSync(M_1, M_2)$ if and only if $MSync(M_1 \oplus N, M_2 \oplus N)$.*

Proof. Take a proof tree for $MSync(M_1, M_2)$, which will end with an axiom of the form $MSync(M_2, M_2)$; then, replace this axiom with $MSync(M_2 \oplus N, M_2 \oplus N)$ and update the proof tree accordingly; the resulting proof tree proves $MSync(M_1 \oplus N, M_2 \oplus N)$. For instance, the proof tree

$$\dfrac{\dfrac{}{Sync(\sigma_1, \sigma_2, \sigma')} \qquad \dfrac{\dfrac{}{Sync(\sigma', \sigma_3, \sigma)} \qquad \dfrac{}{MSync(M_1' \oplus \{\sigma\}, M_1' \oplus \{\sigma\})}}{MSync(M_1' \oplus \{\sigma', \sigma_3\}, M_1' \oplus \{\sigma\})}}{MSync(M_1' \oplus \{\sigma_1, \sigma_2, \sigma_3\}, M_1' \oplus \{\sigma\})}$$

is transformed into the proof tree

$$\dfrac{\dfrac{}{Sync(\sigma_1, \sigma_2, \sigma')} \qquad \dfrac{\dfrac{}{Sync(\sigma', \sigma_3, \sigma)} \qquad \dfrac{}{MSync(M_1' \oplus \{\sigma\} \oplus N, M_1' \oplus \{\sigma\} \oplus N)}}{MSync(M_1' \oplus \{\sigma', \sigma_3\} \oplus N, M_1' \oplus \{\sigma\} \oplus N)}}{MSync(M_1' \oplus \{\sigma_1, \sigma_2, \sigma_3\} \oplus N, M_1' \oplus \{\sigma\} \oplus N)}$$

Conversely, the proof tree for $MSync(M_1 \oplus N, M_2 \oplus N)$ *ends with an axiom* $MSync(M_2 \oplus N, M_2 \oplus N)$*; by replacing this axiom with* $MSync(M_2, M_2)$ *and by updating the proof tree accordingly, we get a proof tree for* $MSync(M_1, M_2)$*.* □

Lemma 7.4. (Transitivity) *For all multisets* $M_1, M_2, M_3 \in \mathscr{B}$*, if* $MSync(M_1, M_2)$ *and* $MSync(M_2, M_3)$*, then* $MSync(M_1, M_3)$*.*

Proof. The proof tree for $MSync(M_1, M_2)$ *has a final axiom* $MSync(M_2, M_2)$*; replace that axiom with the proof tree for* $MSync(M_2, M_3)$*, to create a new proof tree where all the occurrence of* M_2 *(as second argument of MSync) are replaced with* M_3 *in the original proof tree: this new proof tree proves* $MSync(M_1, M_3)$*.* □

Lemma 7.5. (Associativity) *For all* $M_1, M_2, M_3 \in \mathscr{B}$ *such that there exist* M', M *with* $MSync(M_1 \oplus M_2, M')$ *and* $MSync(M' \oplus M_3, M)$*, there exists* M'' *such that* $MSync(M_2 \oplus M_3, M'')$ *and* $MSync(M_1 \oplus M'', M)$*. Also,* $MSync(M_1 \oplus M_2 \oplus M_3, M)$*.*

Proof. As $MSync(M_1 \oplus M_2, M')$*, by additivity Lemma 7.3 we have* $MSync(M_1 \oplus M_2 \oplus M_3, M' \oplus M_3)$*. By transitivity Lemma 7.4, we have* $MSync(M_1 \oplus M_2 \oplus M_3, M)$*. By choosing* $M'' = M_2 \oplus M_3$*, as* $MSync(M_2 \oplus M_3, M_2 \oplus M_3)$*, by additivity Lemma 7.3 we have* $MSync(M_1 \oplus M_2 \oplus M_3, M_1 \oplus M_2 \oplus M_3)$*, and by transitivity Lemma 7.4, we have* $MSync(M_1 \oplus M'', M)$ *as well.* □

Proposition 7.12. *For all restriction-free processes* $p, q, r \in \mathscr{P}_{FNM}$*, if* $p \,|\, (q \,|\, r) \xrightarrow{M}_s s$*, then there exists* t *such that* $(p \,|\, q) \,|\, r \xrightarrow{M}_s t$*, with* $s \equiv t$*, and vice versa.*

Proof. By induction on the proof of $p \,|\, (q \,|\, r) \xrightarrow{M}_s s$*. We have three cases:* (i) $p \xrightarrow{M}_s p_1$ *and* $s = p_1 \,|\, (q \,|\, r)$*; or* (ii) $(q \,|\, r) \xrightarrow{M}_s s_1$ *and* $s = p \,|\, s_1$*; or* (iii) $p \xrightarrow{M_1}_s p_1$*,* $(q \,|\, r) \xrightarrow{M_2}_s s_1$*,* $MSync(M_1 \oplus M_2, M)$ *and* $s = p_1 \,|\, s_1$*.*

In the first case, by rule (Par_1^s*),* $p \,|\, q \xrightarrow{M}_s p_1 \,|\, q$*, and so* $(p \,|\, q) \,|\, r \xrightarrow{M}_s (p_1 \,|\, q) \,|\, r$*, with* $p_1 \,|\, (q \,|\, r) \equiv (p_1 \,|\, q) \,|\, r$*, as required.*

In the second case, we have three subcases: (a) $q \xrightarrow{M}_s q_1$ *and* $s_1 = q_1 \,|\, r$*; or* (b) $r \xrightarrow{M}_s r_1$ *and* $s_1 = q \,|\, r_1$*; or* (c) $q \xrightarrow{M_1}_s q_1$*,* $r \xrightarrow{M_2}_s r_1$*,* $MSync(M_1 \oplus M_2, M)$ *and* $s_1 = q_1 \,|\, r_1$*. In the first subcase, by rule (*Par_2^s*),* $p \,|\, q \xrightarrow{M}_s p \,|\, q_1$*, and so, by rule (*$Par_1^s$*),* $(p \,|\, q) \,|\, r \xrightarrow{M}_s (p \,|\, q_1) \,|\, r$*, with* $p \,|\, (q_1 \,|\, r) \equiv (p \,|\, q_1) \,|\, r$*. The second subcase is symmetric, hence omitted. In the third subcase, by (*Par_2^s*),* $p \,|\, q \xrightarrow{M_1}_s p \,|\, q_1$*, and so, by (S-Com*^{s}*),* $(p \,|\, q) \,|\, r \xrightarrow{M}_s (p \,|\, q_1) \,|\, r_1$*, with* $p \,|\, (q_1 \,|\, r_1) \equiv (p \,|\, q_1) \,|\, r_1$*.*

In the third case, we have three subcases: (a) $q \xrightarrow{M_2}_s q_1$ *and* $s_1 = q_1 \,|\, r$*; or* (b) $r \xrightarrow{M_2}_s r_1$ *and* $s_1 = q \,|\, r_1$*; or* (c) $q \xrightarrow{M_1'}_s q_1$*,* $r \xrightarrow{M_2'}_s r_1$*,* $MSync(M_1' \oplus M_2', M_2)$ *and* $s_1 = q_1 \,|\, r_1$*. The first two subcases are similar to the third subcase of the previous case, and so omitted. In the third subcase, by rule (S-Com*^{s}*),* $p \,|\, q \xrightarrow{M_1 \oplus M_1'}_s p_1 \,|\, q_1$*, and so* $(p \,|\, q) \,|\, r \xrightarrow{M}_s (p_1 \,|\, q_1) \,|\, r_1$ *— because* $MSync(M_1 \oplus M_1' \oplus M_2', M)$ *by the associativity Lemma 7.5 — with* $p_1 \,|\, (q_1 \,|\, r_1) \equiv (p_1 \,|\, q_1) \,|\, r_1$*.*

The symmetric cases where $(p \,|\, q) \,|\, r \xrightarrow{M}_s t$ *moves first are analogous, hence omitted.* □

Theorem 7.4. *Let $p,q \in \mathscr{P}_{FNM}$ be FNM processes such that $p \equiv q$. If $p \xrightarrow{M}_s p'$, then there exists q' such that $q \xrightarrow{M}_s q'$ with $p' \equiv q'$.*

Proof. *By induction on the proof of $p \equiv q$. One has first to show that for each axiom $p = q$ in Table 7.5, generating the structural congruence $\equiv$, we have the thesis. For* **E1** *(associativity), we can resort to Propositions 7.12, while the case of axiom* **E2** *(commutativity) is obvious. Then, one has to prove the thesis for all the rules of equational deduction (reflexivity, symmetry, transitivity, substitutivity and instantiation; see, e.g., [GV15]). The only nontrivial case is about substitutivity, in particular the case of parallel composition: $p = p_1 \,|\, p_2$, $q = q_1 \,|\, q_2$ with $p_1 \equiv q_1$ and $p_2 \equiv q_2$. If $p \xrightarrow{M}_s p'$, then three cases are possible: $p_1 \xrightarrow{M}_s p'_1$ and $p' = p'_1 \,|\, p_2$, or $p_2 \xrightarrow{M}_s p'_2$ and $p' = p_1 \,|\, p'_2$, or $p_1 \xrightarrow{M_1}_s p'_1$, $p_2 \xrightarrow{M_2}_s p'_2$, $MSync(M_1 \oplus M_2, M)$ and $p' = p'_1 \,|\, p'_2$. In the first case, by induction, there exists q'_1 such that $q_1 \xrightarrow{M}_s q'_1$ with $p'_1 \equiv q'_1$, so that, by rule (Par$^s_1$), $q = q_1 \,|\, q_2 \xrightarrow{M}_s q'_1 \,|\, q_2 = q' \equiv p'$. The second case is symmetric, hence omitted. In the third case, by induction, $q_1 \xrightarrow{M_1}_s q'_1$ with $p'_1 \equiv q'_1$ and $q_2 \xrightarrow{M_2}_s q'_2$ with $p'_2 \equiv q'_2$, so that, by rule (S-Com^s), $q = q_1 \,|\, q_2 \xrightarrow{M}_s q'_1 \,|\, q'_2 = q' \equiv p'$.* □

Now we want to prove that all the step transitions labeled with a singleton are also interleaving transitions.

Proposition 7.13. *For any $p \in \mathscr{P}_{FNM}$, if $p \xrightarrow{\{\sigma\}}_s q$, then $p \xrightarrow{\sigma} q$.*

Proof. *(Sketch) The proof is by induction on the proof of $p \xrightarrow{\{\sigma\}}_s q$. We proceed by case analysis. If $p = \mu.p'$, then $p \xrightarrow{\{\mu\}}_s p'$ by (Pref^s), and also $p \xrightarrow{\mu} p'$ by (Pref).*

If $p = \underline{a}.p'$, then $p \xrightarrow{\{a \diamond \sigma\}}_s q$ is derivable by rule (S-Pref^s) only if $p' \xrightarrow{\{\sigma\}}_s q$; induction can be applied to conclude that $p' \xrightarrow{\sigma} q$; hence, by rule (S-Pref), it follows that $p \xrightarrow{a \diamond \sigma} q$.

If $p = p_1 + p_2$, then $p \xrightarrow{\{\sigma\}}_s q$ is derivable only if either $p_1 \xrightarrow{\{\sigma\}}_s q$ (rule (Sum$^s_1$)) or $p_2 \xrightarrow{\{\sigma\}}_s q$ (rule (Sum$^s_2$)). Hence, by induction, we have that either $p_1 \xrightarrow{\sigma} q$ or $p_2 \xrightarrow{\sigma} q$. In any case, transition $p \xrightarrow{\sigma} q$ is derivable by (Sum$_1$) or (Sum$_2$).

If $p = p_1 \,|\, p_2$, transition $p \xrightarrow{\{\sigma\}}_s q$ is derivable only in one of the following three cases: $p_1 \xrightarrow{\{\sigma\}}_s p'_1$ and $q = p'_1 \,|\, p_2$ (rule (Par$^s_1$)); or $p_2 \xrightarrow{\{\sigma\}}_s p'_2$ and $q = p_1 \,|\, p'_2$ (rule (Par$^s_2$)); or $p_1 \xrightarrow{M_1}_s p'_1$, $p_2 \xrightarrow{M_2}_s p'_2$ and $MSync(M_1 \oplus M_2, \{\sigma\})$ (rule (S-Com^s)). In the first case, by induction, we have $p_1 \xrightarrow{\sigma} p'_1$ and so, by rule (Par$_1$), also $p \xrightarrow{\sigma} q$. The second case is analogous, hence omitted. The third case is the most difficult one. As both p_1 and p_2 are restriction-free, for each $\sigma^k_j \in M_k$, there is a subprocess p^k_j of p_k that performs it, for $k = 1,2$. That is, from $p_1 \xrightarrow{M_1}_s p'_1$ and $p_2 \xrightarrow{M_2}_s p'_2$ we can extract two multisets T_1 and T_2 of transitions of the form $p^k_j \xrightarrow{\{\sigma^k_j\}}_s q^k_j$ that, by induction, have their counterpart of the form $p^k_j \xrightarrow{\sigma^k_j} q^k_j$. These interleaving transitions can be *rearranged suitably to build a proof tree for $p' \xrightarrow{\sigma} q'$, with $p \equiv p'$ and $q \equiv q'$, by using the actual proof of relation $MSync(M_1 \oplus M_2, \{\sigma\})$ which tells in what order the subcomponents p_j^k are to be arranged by means of the structural congruence, as illustrated in Example 7.5; then, $p \xrightarrow{\sigma} q$ follows by rule (Cong).*

If $p = (\nu a)p'$, transition $p \xrightarrow{\{\sigma\}}_s q$ is derivable only if, by rule (Res^s), $p' \xrightarrow{\{\sigma\}}_s q'$ is derivable, with σ not containing any occurrence of a or $\overline{a}$, and $q = (\nu a)q'$. By induction, we have $p' \xrightarrow{\sigma} q'$, and so by rule (Res), also $p \xrightarrow{\sigma} q$ is derivable.

If $p = C$, with $C \doteq r$, transition $C \xrightarrow{\{\sigma\}}_s q$ is derivable only if, by rule (Cons^s), $r \xrightarrow{\{\sigma\}}_s q$ is derivable. By induction, $r \xrightarrow{\sigma} q$ is derivable, and so, by rule (Cons), also $C \xrightarrow{\sigma} q$. □

In the reverse direction, one can prove the following fact.

Proposition 7.14. *Let $p \in \mathscr{P}_{FNM}$ be an FNM process. If $p \xrightarrow{\sigma} q$, then there exists q' such that $p \xrightarrow{\{\sigma\}}_s q'$ with $q' \equiv q$.*

Proof. *The proof is by induction on the proof of $p \xrightarrow{\sigma} q$. All the cases are trivial, except when rule (Cong) is used*

$$\frac{p \equiv p' \xrightarrow{\sigma} q' \equiv q}{p \xrightarrow{\sigma} q}$$

If $p' \xrightarrow{\sigma} q'$, then, by induction, we can assume that $p' \xrightarrow{\{\sigma\}}_s q''$ with $q'' \equiv q'$. By Theorem 7.4, also transition $p \xrightarrow{\{\sigma\}}_s q'''$ is derivable with $q''' \equiv q''$. So the thesis follows as, by transitivity, also $q''' \equiv q$. □

Theorem 7.5. (Step bisimilarity implies interleaving bisimilarity) *Let p, q be FNM processes. If $p \sim_{step} q$ then $p \sim q$.*

Proof. *Let R be a step bisimulation such that $(p,q) \in R$. Then, it is easy to prove that R is an interleaving bisimulation* up to $\equiv$ *(Definition 2.14).Consider a generic pair $(p,q) \in R$ and assume that $p \xrightarrow{\sigma} p'$. By Proposition 7.14, if $p \xrightarrow{\sigma} p'$, then there exists p'' such that $p \xrightarrow{\{\sigma\}}_s p''$ with $p' \equiv p''$. Since $(p,q) \in R$ and R is a step bisimulation, then also $q \xrightarrow{\{\sigma\}}_s q'$ is derivable with $(p'', q') \in R$. By Proposition 7.13, also $q \xrightarrow{\sigma} q'$ is derivable. Summing up, to move $p \xrightarrow{\sigma} p'$, q replies with $q \xrightarrow{\sigma} q'$, so that $p' \equiv p'' R q' \equiv q'$, as required by the definition of (interleaving) bisimulation up to $\equiv$. The case when q moves first is symmetric, hence omitted.* □

Of course, the reverse implication of the theorem above is false; for instance, $a.\mathbf{0} \,|\, b.\mathbf{0} \sim a.b.\mathbf{0} + b.a.\mathbf{0}$, but the two are not step bisimilar, as only the former can perform a step transition labeled $\{a, b\}$.

7.3.2 Step Bisimilarity Is a Congruence

Proposition 7.15. (Congruence for prefixing and parallel composition) *Let p and q be restriction-free FNM processes. If $p \sim_{step} q$, then*

(i) $\mu.p \sim_{step} \mu.q$, *for all* $\mu \in Act$,

(ii) $p \,|\, r \sim_{step} q \,|\, r$, *for any restriction-free* $r \in \mathscr{P}_{FNM}$.

Proof. *Assume R is a step bisimulation containing the pair (p,q). For case (i), relation $R_1 = R \cup \{(\mu.p, \mu.q)\}$ is a step bisimulation. For case (ii), relation $R_2 = \{(p' \,|\, r', q' \,|\, r') \mid r' \in \mathscr{P}_{FNM}, (p',q') \in R\}$ is a step bisimulation.* □

Proposition 7.16. (Congruence for strong prefixing and choice) *Let p and q be FNM processes in syntactic category s. If $p \sim_{step} q$, then*

(i) $\underline{a}.p \sim_{step} \underline{a}.q$, *for all* $a \in \mathscr{L}$,

(ii) $p + r \sim_{step} q + r$, *for any process r in syntactic category s.*

Proof. *Assume R is a step bisimulation containing the pair (p,q).*

Case (i) can be proven by considering relation $R_3 = R \cup \{(\underline{a}.p, \underline{a}.q)\}$, which is a step bisimulation. In fact, transition $\underline{a}.p \xrightarrow{\{a \diamond \sigma\}}_s p'$ is derivable, by rule (S-Pref), only if $p \xrightarrow{\{\sigma\}}_s p'$. As $(p,q) \in R$, also $q \xrightarrow{\{\sigma\}}_s q'$ with $(p',q') \in R$. Hence, also $\underline{a}.q \xrightarrow{\{a \diamond \sigma\}}_s q'$ with $(p',q') \in R_3$, as required.

Case (ii) can be proven by showing that the relation $R_4 = \{(p+r, q+r) \mid r \text{ in syntactic category } s\} \cup R \cup \{(r,r) \mid r \text{ is a restriction-free process}\}$ is a step bisimulation. □

Proposition 7.17. (Congruence for restriction) *Let p and q be FNM processes. If $p \sim_{step} q$, then $(\nu a)p \sim_{step} (\nu a)q$ for all $a \in \mathscr{L}$.*

Proof. *Assume R is a step bisimulation containing the pair (p,q). Relation $R_5 = \{((\nu a)p', (\nu a)q') \mid (p',q') \in R\}$ is the required step bisimulation.*

Summing up, we have that step bisimilarity is a congruence over all the FNM operators. This result gives evidence that to give a satisfactory, compositional account of FNM, one needs a *non-interleaving* model of concurrency, such as the step transition system. The advantages of the step semantics are essentially

- a simpler structural operational semantics, which makes no use of the structural congruence $\equiv$, and
- a more adequate behavioral semantics, namely step bisimilarity, which is finer than interleaving bisimilarity and is a congruence for all the operators of the language.

Proposition 7.15(ii) and Theorem 7.5 ensure that for any pair of restriction-free, FNM processes p and q, if $p \sim_{step} q$ then $p \,|\, r \sim q \,|\, r$, for any restriction-free process r. One may wonder whether the reverse holds: if for all restriction-free r, we have that $p \,|\, r \sim q \,|\, r$, can we conclude that $p \sim_{step} q$? If this is the case, we can say that step equivalence is the *coarsest congruence* contained in interleaving bisimulation for FNM. The answer to this question is negative, as the following example shows.

Example 7.6. Take processes $p = \tau.\tau.\mathbf{0}$ and $q = \tau.\mathbf{0} \,|\, \tau.\mathbf{0}$. It is not difficult to see that $p \,|\, r \sim q \,|\, r$, for all processes r; however, $p \not\sim_{step} q$ as only q can perform the step $\{\tau, \tau\}$. □

The problem of finding the coarsest congruence contained in $\sim$ is open.

7.4 Operational Net Semantics

In this section, we describe a technique for building a P/T net for the whole of FNM, starting from a description of its places and its net transitions. The resulting net $N_{FNM} = (S_{FNM}, \mathscr{A}, T_{FNM})$ is such that, for any $p \in \mathscr{P}_{FNM}$, the net system $N_{FNM}(dec(p))$ statically reachable from the initial marking $dec(p)$ is a statically reduced, finite P/T net; such a net system is denoted by $Net(p)$.

7.4.1 Places and Markings

The infinite set of FNM places, ranged over by s, is $S_{FNM} = \mathscr{P}^{\gamma,seq}_{FNM} \setminus \{\mathbf{0}\}$, i.e., the set of all sequential, *extended* FNM processes, except $\mathbf{0}$.

Function $dec : \mathscr{P}^{\gamma}_{FNM} \to \mathscr{M}_{fin}(S_{FNM})$, which defines the decomposition of extended processes into markings, was outlined in Table 6.8 for the FNC operators. It is extended to the strong prefixing operator as follows: $dec(\underline{\gamma}.p, I) = \{\underline{\gamma}.p\}$, where $\gamma \in \mathscr{G} = \mathscr{L} \cup \mathscr{L}'$. Hence, the decomposition of the sequential (extended) process $\underline{\gamma}.p$ produces one place with name $\underline{\gamma}.p$, analogously to what happens for any other sequential process.

Lemma 7.6. *For any restriction-free, extended FNM process t, $dec(t)\{L'/L\} = dec(t\{L'/L\})$.*

Proof. *By induction on the definition of $dec(t)$, as done for the analogous Lemma 6.2.* □

Proposition 7.18. *For any restriction-free $t \in \mathscr{P}_{FNM}$, $dec((\nu L)t) = dec(i((\nu L)t)) = dec(t\{L'/L\})$.*

Proof. *By definition, $dec((\nu L)t) = dec(t)\{L'/L\}$. Then, by Lemma 7.6 $dec(t)\{L'/L\} = dec(t\{L'/L\})$.* □

This means that we can restrict our attention to extended, restriction-free processes built over the set of visible and restricted actions Act_γ, as a restricted process $(\nu L)t$ in $\mathscr{P}_{FNM}$ is mapped via dec to the same marking associated with $i((\nu L)t) = t\{L'/L\}$ in $\mathscr{P}^{\gamma,par}_{FNM}$.

Example 7.7. Consider the (non-well-formed) FNM process $p = (\nu a)p'$, where $p' = (a.\mathbf{0} \,|\, (\underline{a}.\overline{a}.\mathbf{0} \,|\, \overline{a}.\mathbf{0}))$. Then,

$$\begin{aligned} dec(p) &= dec(p')\{a'/a\} = dec(a.\mathbf{0} \,|\, (\underline{a}.\overline{a}.\mathbf{0} \,|\, \overline{a}.\mathbf{0}))\{a'/a\} \\ &= (dec(a.\mathbf{0}) \oplus dec(\underline{a}.\overline{a}.\mathbf{0} \,|\, \overline{a}.\mathbf{0}))\{a'/a\} \\ &= (dec(a.\mathbf{0}) \oplus dec(\underline{a}.\overline{a}.\mathbf{0}) \oplus dec(\overline{a}.\mathbf{0}))\{a'/a\} \\ &= (a.\mathbf{0} \oplus \underline{a}.\overline{a}.\mathbf{0} \oplus \overline{a}.\mathbf{0})\{a'/a\} = a'.\mathbf{0} \oplus \underline{a'}.\overline{a'}.\mathbf{0} \oplus \overline{a'}.\mathbf{0} \\ &= dec(a'.\mathbf{0} \,|\, \underline{a'}.\overline{a'}.\mathbf{0} \,|\, \overline{a'}.\mathbf{0}) \\ &= dec(i(p)) \end{aligned}$$

where a' is the restricted name in $\mathscr{L}'$ corresponding to the bound name a in $\mathscr{L}$. □

It is easy to see that the decomposition function *dec* is well defined.

Proposition 7.19. *For any $p \in \mathscr{P}_{FNM}^{\gamma}$, $dec(p)$ is a finite multiset of places.*

Proof. *By induction on the definition of $dec(p)$.* □

Of course, function *dec* is not injective; however, one can prove that function *dec* is surjective over *admissible* markings, where we recall that $m \in \mathscr{M}_{fin}(S_{FNM})$ is admissible, denoted by $ad(m)$, if for all $a \in \mathscr{L}$, $\{a, a'\} \not\subseteq fn(m)$ (see Definition 6.9). Note that, for any two markings m_1 and m_2, if $fn(m_2) \subseteq fn(m_1)$ and $ad(m_1)$ holds, then also $ad(m_2)$ holds; this fact will often be used in the following proofs. We also recall that a marking $m \in \mathscr{M}_{fin}(S_{FNM})$ is *complete* if there is a FNM process $p \in \mathscr{P}_{FNM}$ such that $dec(p) = m$.

Lemma 7.7. *For any $t \in \mathscr{P}_{FNM}^{\gamma,par}$, $fn(t) = fn(dec(t))$.*

Proof. *By induction on the definition of $dec(t)$.* □

Theorem 7.6. *A marking $m \in \mathscr{M}_{fin}(S_{FNM})$ is admissible iff it is complete.*

Proof. *Similar to that for the analogous Theorem 6.4, by using Proposition 7.18 and Lemma 7.7.* □

Hence, this theorem states not only that function *dec* maps FNM processes to admissible markings over S_{FNM}, but also that *dec* is surjective over this set.

We now list some useful properties of places and markings. First, as done for FNC, we extend the definition of sequential subterms of a process p to a set of places S, as follows: $sub(\emptyset) = \emptyset$ and $sub(S) = \bigcup_{s \in S} sub(s)$. The goal is to prove that the sequential subterms of $dom(dec(p))$ are the sequential subterms of p. This property will be useful in proving (Theorem 7.10) that each place statically reachable from $dom(dec(p))$ is a sequential subterm of p, so that, since $sub(p)$ is finite for any p (Theorem 7.1), the set of all the places statically reachable from $dom(dec(p))$ is finite, too. The proofs of the following propositions are very similar to the analogous propositions of Section 6.4.1, where the reader can find more details.

Proposition 7.20. *For any finite set of places S_1 and S_2, if $S_1 \subseteq sub(S_2)$, then $sub(S_1) \subseteq sub(S_2)$.*

Proof. *By induction on the cardinality of* S_1. □

Proposition 7.21. *For any set of places* S, $S \subseteq sub(S)$.

Proof. *For any sequential process* s, $s \in sub(s)$ *holds; hence, the thesis follows.* □

Proposition 7.22. *For any* $t \in \mathscr{P}_{FNM}^{\gamma,par}$, $sub(dom(dec(t))) \subseteq sub(t)$.

Proof. *By induction on the definition of* $dec(t)$. □

Corollary 7.1. *For any* $p \in \mathscr{P}_{FNM}$, $sub(dom(dec(p))) \subseteq sub(p)$.

Proof. *If* p *is restriction-free, then the thesis follows by Proposition 7.22. If* $p = (\nu L)t$, *then* $dec(p) = dec(t\{L'/L\})$ *by Proposition 7.18. By Proposition 7.1,* $sub(p) = sub(t\{L'/L\})$. *By Proposition 7.22,* $sub(dom(dec(t\{L'/L\}))) \subseteq sub(t\{L'/L\})$. *So,* $sub(dom(dec(p))) = sub(dom(dec(t\{L'/L\}))) \subseteq sub(t\{L'/L\}) = sub(p)$. □

Now we want to extend the definition of well-formed processes to sets of places. To this aim, we define the notion of a *well-behaved* set of places, which will be useful in Section 7.4.3 for proving that the set of transitions statically enabled at a well-behaved set S is finite (Theorem 7.9).

Definition 7.2. (Well-behaved) A set of places $S \subseteq S_{FNM}$ is *well behaved* if for all $s \in S$ we have that $wf(s)$ holds. □

Note that the empty marking $m = \emptyset$ is well behaved, because the universal condition on its constituents is vacuously true. Note also that, by definition, it follows that the union of two well-behaved sets of places is well behaved and a subset of a well-behaved set is well behaved. Now we prove that an FNM process p is well formed if and only if its associated marking $dec(p)$ is such that $dom(dec(p))$ is well behaved.

Theorem 7.7. *For any* $p \in \mathscr{P}_{FNM}$, $wf(p)$ *holds if and only if* $dom(dec(p))$ *is well behaved.*

Proof. *By induction on the definition of* $dec(p)$. *We proceed by case analysis. If* $p = \mathbf{0}$, *then* $wf(p)$ *holds and* $dec(p) = \emptyset$ *is well behaved, as required. If* p *is any other sequential process, then* $dec(p) = \{p\}$; *therefore,* $wf(p)$ *holds if and only if* $\{p\}$ *is well behaved, as required.*

If $p = p_1 \mid p_2$, *then* $wf(p)$ *holds only if* $wf(p_1)$ *and* $wf(p_2)$ *hold. By induction, we have that* $dom(dec(p_i))$ *is well behaved, for* $i = 1,2$; *so,* $dom(dec(p_1 \mid p_2)) = dom(dec(p_1)) \cup dom(dec(p_2))$ *is well behaved, too, because the union of two well-behaved sets is well behaved. Conversely, if* $dom(dec(p_1 \mid p_2))$ *is well behaved, then also* $dom(dec(p_i))$ *is well behaved, for* $i = 1,2$; *hence, by induction,* $wf(p_1)$ *and* $wf(p_2)$ *hold, and so also* $wf(p)$ *holds.*

If $p = (\nu a)p'$, *then* $wf(p)$ *holds only if* $wf(p')$ *holds. By induction, we have that* $dom(dec(p'))$ *is well behaved. It is easy to see that* $dom(dec(p'))\{a'/a\}$ *is well behaved too, as the substitution replaces action* a *with a new name* a' *not in use. Hence, as* $dom(dec(p'))\{a'/a\} = dom(dec(p')\{a'/a\}) = dom(dec(p))$, *also* $dom(dec(p))$ *is well behaved. Conversely, if* $dom(dec(p)) = dom(dec(p'))\{a'/a\}$ *is well behaved, then also* $dom(dec(p'))$ *is well behaved; by induction,* $wf(p')$ *holds and so* $wf(p)$ *holds too.* □

$$\text{(pref)}\ \frac{}{\{\mu.p\}\xrightarrow{\mu} dec(p)} \qquad \text{(cons)}\ \frac{dec(p)\xrightarrow{\sigma} m}{\{C\}\xrightarrow{\sigma} m}\ C \doteq p$$

$$\text{(sum}_1\text{)}\ \frac{dec(p)\xrightarrow{\sigma} m}{\{p+q\}\xrightarrow{\sigma} m} \qquad \text{(s-pref)}\ \frac{dec(p)\xrightarrow{\sigma} m}{\{\underline{\gamma}.p\}\xrightarrow{\gamma\diamond\sigma} m}$$

$$\text{(s-com)}\ \frac{m_1\xrightarrow{\sigma_1} m_1' \quad m_2\xrightarrow{\sigma_2} m_2'}{m_1\oplus m_2\xrightarrow{\sigma} m_1'\oplus m_2'}\ ad(m_1\oplus m_2)\wedge Sync(\sigma_1,\sigma_2,\sigma)$$

Table 7.13 Rules for net transitions (symmetric rule (sum$_2$) omitted)

$$\text{(s-com)}\ \dfrac{\text{(s-com)}\ \dfrac{\text{(s-pref)}\ \dfrac{\text{(pref)}\ \dfrac{}{\{b'.p\}\xrightarrow{b'} dec(p)}}{\{\underline{a}.b'.p\}\xrightarrow{ab'} dec(p)} \qquad \text{(pref)}\ \dfrac{}{\{\bar{a}.q\}\xrightarrow{\bar{a}} dec(q)}}{\{\underline{a}.b'.p,\bar{a}.q\}\xrightarrow{b'} dec(p)\oplus dec(q)} \qquad \text{(pref)}\ \dfrac{}{\{\overline{b'}.r\}\xrightarrow{\overline{b'}} dec(r)}}{\{\underline{a}.b'.p,\bar{a}.q,\overline{b'}.r\}\xrightarrow{\tau} dec(p)\oplus dec(q)\oplus dec(r)}$$

Table 7.14 The proof of a net transition

7.4.2 *Net Transitions*

Let $\mathscr{A}^\gamma = \{\tau\}\cup\mathscr{G}^*\cdot(\mathscr{G}\cup\overline{\mathscr{G}})$, ranged over by σ with abuse of notation, be the set of labels; hence, a label σ can be the invisible action τ, or a (possibly empty) sequence of inputs (or restricted inputs) followed by an input or an output (or its restricted counterpart). Let $\rightarrow\subseteq\mathscr{M}_{fin}(S_{FNM})\times\mathscr{A}^\gamma\times\mathscr{M}_{fin}(S_{FNM})$ be the least set of transitions generated by the axiom and rules in Table 7.13.

Let us comment on the new rules of Table 7.13, where the symmetric rule (sum$_2$) is omitted. In rule (s-pref), γ may be any input action a or any *restricted* action a'. This rule requires that the premise transition $dec(p)\xrightarrow{\sigma} m$ be derivable by the rules; since p is in syntactic category s, $dec(p)=\emptyset$ if $p=\mathbf{0}$, or $dec(p)=\{p\}$. Since no transition is derivable from the empty marking, the conclusion $\{\underline{\gamma}.p\}\xrightarrow{\gamma\diamond\sigma} m$ is actually derivable only from a premise of the form $\{p\}\xrightarrow{\sigma} m$. Rule (s-com) requires that the transition pre-set be admissible in order to avoid producing synchronized transitions that have no counterpart in the LTS semantics (Proposition 7.27). Rule (s-com) explains how a synchronization takes place: it is required that m_1 and m_2 perform synchronizable sequences σ_1 and σ_2, producing σ; here we assume that relation *Sync* has been extended also to restricted actions in the obvious way, i.e., a restricted output action $\overline{a'}$ can be synchronized only with its complementary restricted input action a' or with an atomic sequence beginning with a'. As an example, the net transition $\{\underline{a}.b'.p,\bar{a}.q,\overline{b'}.r\}\xrightarrow{\tau} dec(p)\oplus dec(q)\oplus dec(r)$ is derivable by the rules, as shown in Table 7.14.

Transitions with labels containing restricted actions must not be taken in the resulting net, as we accept only transitions labeled over $\mathscr{A} = \{\tau\} \cup \mathscr{L}^* \cdot (\mathscr{L} \cup \overline{\mathscr{L}})$. However, they are useful in producing acceptable transitions, as two complementary restricted actions can synchronize, producing a τ-labeled transition or shortening the synchronized sequence. For instance, in the example above, the derivable transition $\{b'.p\} \xrightarrow{b'} dec(p)$ is not an acceptable transition because its label is not in $\mathscr{A}$, while $\{\underline{a}.b'.p, \overline{a}.q, \overline{b'}.r\} \xrightarrow{\tau} dec(p) \oplus dec(q) \oplus dec(r)$ is so. Hence, the P/T net for FNM is the triple $N_{FNM} = (S_{FNM}, \mathscr{A}, T_{FNM})$, where the set

$$T_{FNM} = \{(m_1, \sigma, m_2) \mid m_1 \xrightarrow{\sigma} m_2 \text{ is derivable by the rules and } \sigma \in \mathscr{A}\}$$

is obtained by filtering out those transitions derivable by the rules whose label σ contains some restricted name a' or $\overline{a'}$.

7.4.3 Properties of Net Transitions

Some useful properties of net transitions are listed here. First, given a transition $t = (m_1, \sigma, m_2)$, derivable by the rules in Table 7.13, we show that the subterms of the marking m_2 are already present in the set of subterms of the marking m_1.

Proposition 7.23. *Let $t = m_1 \xrightarrow{\sigma} m_2$ be a transition derivable by the rules in Table 7.13. Then, $sub(dom(m_2)) \subseteq sub(dom(m_1))$.*

Proof. *By induction on the proof of t. The proof is very similar to the analogous Proposition 6.13; here we consider only the inductive case of the new rule (s-pref), whose premise transition is $\{p\} \xrightarrow{\sigma} m_2$ and whose initial marking is $m_1 = \{\underline{\gamma}.p\}$. By induction, we have that $sub(dom(m_2)) \subseteq sub(dom(\{p\})) = sub(p)$. By definition, $sub(\{\underline{\gamma}.p\}) = sub(\underline{\gamma}.p)$ and $sub(\underline{\gamma}.p) = \{\underline{\gamma}.p\} \cup sub(p)$; hence, the thesis follows by transitivity.* □

Now we want to prove that if we start from an admissible set of places, then we can statically reach only admissible sets of places.

Lemma 7.8. *Let $t = m_1 \xrightarrow{\sigma} m_2$ be a transition derivable by the rules in Table 7.13. Then, $fn(m_2) \subseteq fn(m_1)$.*

Proof. *By induction on the proof of t. The proof if very similar to the analogous Lemma 6.4; here we consider only the inductive case of the new rule (s-pref), whose premise transition is $\{p\} \xrightarrow{\sigma} m_2$ and whose initial marking is $m_1 = \{\underline{\gamma}.p\}$. By induction, we have that $fn(m_2) \subseteq fn(\{p\})$. By definition, $fn(\{p\}) = fn(p)$, $fn(\{\underline{\gamma}.p\}) = fn(\underline{\gamma}.p)$ and $fn(\underline{\gamma}.p) = \{\gamma\} \cup fn(p)$; hence, the thesis — $fn(m_2) \subseteq fn(\{\underline{\gamma}.p\})$ — follows by transitivity.* □

Proposition 7.24. *Let $t = m_1 \xrightarrow{\sigma} m_2$ be a transition derivable by the rules in Table 7.13. Then, m_1 and m_2 are admissible.*

Proof. *By induction on the proof of t, we can easily conclude that m_1 is admissible; this proof is similar to the analogous Proposition 6.14, hence omitted. Moreover, by Lemma 7.8, we have that $fn(m_2) \subseteq fn(m_1)$, and so m_2 is admissible, too.* □

Proposition 7.25. *Let $t = m_1 \xrightarrow{\sigma} m_2$ be a transition derivable by the rules in Table 7.13. Let m be an admissible marking such that $m[t\rangle m'$. Then, m' is admissible.*

Proof. *If t is enabled at m, then ${}^\bullet t \subseteq m$; hence, $m = m_0 \oplus {}^\bullet t$, for some suitable m_0. By the definition of transition firing, $m' = m \ominus {}^\bullet t \oplus t^\bullet = m_0 \oplus t^\bullet$. By Lemma 7.8, $fn(t^\bullet) \subseteq fn({}^\bullet t)$; consequently, $fn(m') = fn(m_0) \cup fn(t^\bullet) \subseteq fn(m_0) \cup fn({}^\bullet t) = fn(m)$. Therefore, also m' is admissible.* □

As a set of places can be seen as a marking, we can generalize the results above about admissibility to statically reachable sets of places.

Theorem 7.8. *If S_1 is admissible and $S_1 \Longrightarrow^* S_k$, then S_k is admissible.*

Proof. *By induction on the static reachability relation $\Longrightarrow^*$, as done for the analogous Theorem 6.5, by using in places Lemma 7.8.* □

Now we want to prove that the net N_{FNM} contains only transitions that have a counterpart in the LTS semantics for FNM, as outlined in Section 7.2; this depends crucially on the admissibility condition over rule (s-com). This proposition exploits Lemma 7.9, which states a correspondence between net transitions derivable by the rules in Table 7.13 — which can be labeled over $\mathscr{A}^\gamma$ — and LTS transitions derivable by the rules in Table 7.3 from extended, restriction-free processes (see Remark 7.3). Some auxiliary results are needed.

Proposition 7.26. *If $t = (m_1, \sigma, m_2)$ is a transition derivable by the rules in Table 7.13 and $L \subseteq \mathscr{L}$, then transition $t\{L'/L\} = (m_1\{L'/L\}, \sigma\{L'/L\}, m_2\{L'/L\})$ is derivable as well, where $\mu\{L'/L\} = \mu$ if $\mu, \overline{\mu} \notin L$, while $\mu\{L'/L\} = \mu'$ otherwise, and $(a\sigma)\{L'/L\} = a\{L'/L\}\sigma\{L'/L\}$. And vice versa, if $t\{L'/L\}$ is derivable, then also t is derivable.*

Proof. *By induction on the proof of t. The proof is similar to the analogous Proposition 6.16, hence we consider only the cases of the new rules. If the last rule is (s-pref), then $(\{\underline{\gamma}.p\}, \sigma, m_2)$ is derived by the premise transition $(\{p\}, \sigma', m_2)$, with $\sigma = \gamma \diamond \sigma'$. We can assume, by induction, that $(\{p\}\{L'/L\}, \sigma'\{L'/L\}, m_2\{L'/L\})$ is derivable; by Lemma 7.6, $\{p\}\{L'/L\} = \{p\{L'/L\}\}$, and so the last transition is actually $(\{p\{L'/L\}\}, \sigma'\{L'/L\}, m_2\{L'/L\})$. Now assume that $\gamma \notin L$, so that $(\underline{\gamma}.p)\{L'/L\} = \underline{\gamma}.p\{L'/L\}$. Hence, by (s-pref), also $(\{(\underline{\gamma}.p)\{L'/L\}\}, \gamma \diamond (\sigma'\{L'/L\}), m_2\{L'/L\})$ is derivable, as required, as $\sigma\{L'/L\} = (\gamma \diamond \sigma')\{L'/L\} = \gamma \diamond (\sigma'\{L'/L\})$. If $\gamma \in L$, then $(\underline{\gamma}.p)\{L'/L\} = \underline{\gamma'}.p\{L'/L\}$, and so by (s-pref) also $(\{(\underline{\gamma}.p)\{L'/L\}\}, \gamma' \diamond (\sigma'\{L'/L\}), m_2\{L'/L\})$ is derivable, as required, as $\sigma\{L'/L\} = (\gamma \diamond \sigma')\{L'/L\} = \gamma' \diamond (\sigma'\{L'/L\})$.*

For rule (s-com), transition $(m_1 \oplus m_2, \sigma, m_1' \oplus m_2')$ is due to the premise transitions (m_1, σ_1, m_1') and (m_2, σ_2, m_2'), with $Sync(\sigma_1, \sigma_2, \sigma)$. By induction, we can assume that $(m_1\{L'/L\}, \sigma_1\{L'/L\}, m_1'\{L'/L\})$ and $(m_2\{L'/L\}, \sigma_2\{L'/L\}, m_2'\{L'/L\})$

are derivable; note that $Sync(\sigma_1\{L'/L\}, \sigma_2\{L'/L\}, \sigma\{L'/L\})$ *holds, and so, by (s-com), also* $(m_1\{L'/L\} \oplus m_2\{L'/L\}, \sigma\{L'/L\}, m'_1\{L'/L\} \oplus m'_2\{L'/L\})$ *is derivable, where* $m_1\{L'/L\} \oplus m_2\{L'/L\} = (m_1 \oplus m_2)\{L'/L\}$ *and* $m'_1\{L'/L\} \oplus m'_2\{L'/L\} = (m'_1 \oplus m'_2)\{L'/L\}$*, as required.*

For the vice versa — if $t\{L'/L\}$ *is derivable, then also* t *is derivable — observe that this is similar to the above: if* $t\{L'/L\}$ *is derivable, then* $(t\{L'/L\})\{L/L'\} = t$ *is derivable, where an inverse substitution* $\{L/L'\}$ *is used instead.* □

Lemma 7.9. *For any* $t = (m_1, \sigma, m_2)$ *derivable by the rules in Table 7.13, there exist two extended, restriction-free FNM processes* p_1 *and* p_2 *such that* $p_1 \xrightarrow{\sigma} p_2$ *is derivable by the rules in Table 7.3, with* $dec(p_1) = m_1$ *and* $dec(p_2) = m_2$.

Proof. By induction on the proof of t*, we show a proof for* $p_1 \xrightarrow{\sigma} p_2$*, for suitable extended, restriction-free processes* p_1 *and* p_2 *such that* $dec(p_1) = m_1$ *and* $dec(p_2) = m_2$*. The proof is similar to the analogous Lemma 6.5; hence, we discuss only the cases of the new rules. If the last rule used to derive* t *is (s-pref), then* $t = \{\underline{\gamma}.p\} \xrightarrow{\sigma} m_2$*, with a premise* $t' = \{p\} \xrightarrow{\sigma'} m_2$ *such that* $\sigma = \gamma \diamond \sigma'$*. By induction, we know that* $p \xrightarrow{\sigma'} p_2$ *is derivable, with* p_2 *an extended, restriction-free process such that* $dec(p_2) = m_2$*; hence, by rule (S-Pref), also transition* $\underline{\gamma}.p \xrightarrow{\sigma} p_2$ *is derivable, as required. If the last rule used to derive* t *is (s-com), then there exist two transitions* $t_1 = m'_1 \xrightarrow{\sigma_1} m'_2$ *and* $t_2 = m''_1 \xrightarrow{\sigma_2} m''_2$ *such that* $t = m'_1 \oplus m''_1 \xrightarrow{\sigma} m'_2 \oplus m''_2$*,* $ad(m'_1 \oplus m''_1)$ *and* $Sync(\sigma_1, \sigma_2, \sigma)$*. By induction, there exist transition* $q_1 \xrightarrow{\sigma_1} q_2$ *— with* q_1 *and* q_2 *extended, restriction-free processes such that* $dec(q_1) = m'_1$ *and* $dec(q_2) = m'_2$ *— and transition* $r_1 \xrightarrow{\sigma_2} r_2$ *— with* r_1 *and* r_2 *extended, restriction-free processes such that* $dec(r_1) = m''_1$ *and* $dec(r_2) = m''_2$*. Therefore, by rule (S-Com), also transition* $q_1 \,|\, r_1 \xrightarrow{\sigma} q_2 \,|\, r_2$ *is derivable. Note that, by Lemma 7.7,* $fn(q_1 \,|\, r_1) = fn(dec(q_1 \,|\, r_1)) = fn(m'_1 \oplus m''_1)$*; therefore,* $q_1 \,|\, r_1$ *is an extended, restriction-free process because it is admissible, since the marking* $m'_1 \oplus m''_1$ *is assumed admissible by rule (s-com). Similarly, also* $q_2 \,|\, r_2$ *is admissible, because* $fn(q_2 \,|\, r_2) = fn(dec(q_2 \,|\, r_2)) = fn(m'_2 \oplus m''_2)$ *and the marking* $m'_2 \oplus m''_2$ *is admissible by Proposition 7.24.* □

Proposition 7.27. *For any* $t \in T_{FNM}$*, where* $t = (m_1, \sigma, m_2)$ *with* $\sigma \in \mathscr{A}$*, there exist two FNM processes* p_1 *and* p_2 *such that* $p_1 \xrightarrow{\sigma} p_2$*,* $dec(p_1) = m_1$ *and* $dec(p_2) = m_2$.

Proof. By Lemma 7.9, transition $q_1 \xrightarrow{\sigma} q_2$ *is derivable by the rules in Table 7.3, for some suitable extended, restriction-free processes* q_1 *and* q_2 *such that* $dec(q_1) = m_1$ *and* $dec(q_2) = m_2$*. Let* $L' = fn(m_1) \cap \mathscr{L}'$ *and* $L = \{a \mid a' \in L'\}$*. If* $L' = \emptyset$*, then* q_1 *is actually an FNM process and the required transition is exactly* $q_1 \xrightarrow{\sigma} q_2$*, with* q_2 *also an FNM process by Proposition 7.3. On the contrary, if* $L' \neq \emptyset$*, then* $a \notin fn(m_1)$ *for all* $a \in L$*, because* m_1 *is admissible, so that* $a \notin n(\sigma)$*. Moreover, by Proposition 7.26, also* $t\{L/L'\}$ *is derivable, and the same proof of transition* $q_1 \xrightarrow{\sigma} q_2$ *can be used to prove that transition* $q_1\{L/L'\} \xrightarrow{\sigma\{L/L'\}} q_2\{L/L'\}$ *is derivable by the rules in*

Table 7.3; note that $\sigma\{L/L'\} = \sigma$, as $\sigma \in \mathscr{A}$; moreover, $q_1\{L/L'\}$ and $q_2\{L/L'\}$ are FNM processes, as, for $i = 1,2$, $fn(q_i) = fn(dec(q_i)) = fn(m_i)$ by Lemma 7.7, and so all the restricted actions a' occurring in q_i are turned into their corresponding action a in $q_i\{L/L'\}$. Finally, by rule (Res), also $(\nu L)(q_1\{L/L'\}) \xrightarrow{\sigma} (\nu L)(q_2\{L/L'\})$ is derivable, where, for $i = 1,2$, $(\nu L)(q_i\{L/L'\})$ is an FNM process. Note that $dec((\nu L)(q_i\{L/L'\})) = m_i$, as $dec((\nu L)(q_i\{L/L'\})) = dec(q_i\{L/L'\})\{L'/L\}$ by definition of function dec, and $dec(q_i\{L/L'\})\{L'/L\} = dec((q_i\{L/L'\})\{L'/L\})$ by Lemma 7.6, and $dec((q_i\{L/L'\})\{L'/L\}) = dec(q_i) = m_i$ by Proposition 6.3 (extended to FNM). □

Now we want to prove that if we start from a well-behaved set of places, then we can statically reach only well-behaved sets of places.

Proposition 7.28. *If $t = m_1 \xrightarrow{\sigma} m_2$ is derivable by the rules and $dom(m_1)$ is well behaved, then $dom(m_2)$ is well behaved.*

Proof. By induction on the proof of t. For axiom (pref), $m_1 = \{\mu.p\}$, $\sigma = \mu$ and $m_2 = dec(p)$. Since $dom(m_1)$ is well behaved, we have that $wf(\mu.p)$ holds; and this holds only if $wf(p)$ holds, too. The thesis then follows by Theorem 7.7. For rule (sum$_1$), $m_1 = \{p+q\}$ and the premise transition is $\{p\} \xrightarrow{\sigma} m_2$. Since $dom(m_1)$ is well behaved, we have that $wf(p+q)$ holds; and this holds only if $wf(p)$ holds, too, so that $dom(\{p\})$ is well behaved, too; therefore, induction can be applied to the premise transition to conclude that $dom(m_2)$ is well behaved, as required. The cases of rules (sum$_2$), (cons) and (s-pref) are similar, hence omitted. For rule (s-com), $m_1 = m_1' \oplus m_1''$, $m_2 = m_2' \oplus m_2''$ and the premise transitions are $m_1' \xrightarrow{\sigma_1} m_2'$ and $m_1'' \xrightarrow{\sigma_2} m_2''$, with $ad(m_1' \oplus m_1'')$ and $Sync(\sigma_1, \sigma_2, \sigma)$. Since $dom(m_1)$ is well behaved, we have that $dom(m_1')$ and $dom(m_1'')$ are well behaved, too, so that, by induction, we can conclude that $dom(m_2')$ and $dom(m_2'')$ are well behaved; as a consequence, also $dom(m_2)$ is well behaved, as required. □

Corollary 7.2. *If S_1 is well behaved and $S_1 \Longrightarrow^* S_k$, then S_k is well behaved.*

Proof. By induction on the static reachability relation $\Longrightarrow^*$. The base case is $S_1 \Longrightarrow^* S_1$ and it is trivial. The inductive case is $S_1 \Longrightarrow^* S_{k-1} \overset{t}{\Longrightarrow} S_k$, for some $t = m_1 \xrightarrow{\sigma} m_2$ in T_{FNM}. By induction we can assume that S_{k-1} is well behaved. Since t is statically enabled at S_{k-1}, we have that $dom(m_1) \subseteq S_{k-1}$, so that also $dom(m_1)$ is well behaved. By Proposition 7.28, also $dom(m_2)$ is well behaved, so that $S_k = S_{k-1} \cup dom(m_2)$ is well behaved, as required. □

Now we want to prove that when a transition $m \xrightarrow{\sigma} m'$ has a well-behaved pre-set, then σ can be the invisible action τ, or a single output $\overline{\gamma} \in \overline{\mathscr{G}}$, or a sequence of (possibly restricted) inputs in $\mathscr{G}^+$. Moreover, if $\sigma \in \overline{\mathscr{G}}$, then the pre-set m_1 is actually a singleton.

The following lemma makes use of function $In(-)$, defined over sequential processes in syntactic category s in Section 7.1.3, and extended to process constants as follows: $In(C) = In(p)$ if $C \doteq p$, since p is in syntactic category s.

Lemma 7.10. *For any $s \in S_{FNM}$ and for any $\sigma \in \mathscr{A}^{\gamma}$, transition $t = \{s\} \xrightarrow{\sigma} m$ is derivable if and only if $\sigma \in In(s)$.*

Proof. *By induction on the proof of t and on the definition of $In(s)$.* □

Proposition 7.29. *If $t = m_1 \xrightarrow{\sigma} m_2$ is derivable by the rules and m_1 is well behaved, then $\sigma \in \{\tau\} \cup \overline{\mathscr{G}} \cup \mathscr{G}^+$. Moreover, if $\sigma \in \overline{\mathscr{G}}$, then $|^\bullet t| = 1$ and the proof tree for t does not contain any transition with a label different from σ itself.*

Proof. *By induction on the proof of t. We proceed by case analysis. If $m_1 = \{\mu.p'\}$, by axiom (pref), $\{\mu.p'\} \xrightarrow{\mu} dec(p')$ is derivable, so that $\mu \in \{\tau\} \cup \overline{\mathscr{G}} \cup \mathscr{G}^+$ and $|m_1| = 1$; if $\mu \in \overline{\mathscr{G}}$, then the proof tree of the transition does not contain any transition with a label different from μ itself.*

If $m_1 = \{p_1 + p_2\}$, then $m_1 \xrightarrow{\sigma} m_2$ is due to either rule (sum$_1$) or (sum$_2$); w.l.o.g., assume that (sum$_1$) is used; in this case, the premise transition must be $\{p_1\} \xrightarrow{\sigma} m_2$; since $dom(m_1)$ is well behaved, wf$(p_1 + p_2)$ holds, and this holds only if also wf(p_1) holds; therefore, induction can be used to conclude that all the requirements of the proposition are satisfied for σ; in particular, if $\sigma \in \overline{\mathscr{G}}$, then $|m_1| = 1$ and the proof tree of the transition does not contain any transition with a label different from σ itself.

The case when $m_1 = \{C\}$ is analogous to the above, hence omitted.

If $m_1 = \{\gamma.p_1\}$, then $m_1 \xrightarrow{\sigma} m_2$ is due to rule (s-pref), with premise $\{p_1\} \xrightarrow{\sigma'} m_2$ such that $\gamma \diamond \sigma' = \sigma$. By Lemma 7.10, $\sigma' \in In(p_1)$; since $dom(m_1)$ is well behaved, wf$(\gamma.p_1)$ holds, and this holds only if also wf(p_1) holds and, moreover, $In(p_1) \subseteq \{\tau\} \cup \mathscr{G}^+$; therefore, $\sigma \in \mathscr{G}^+$, and so the requirements of the proposition are satisfied for σ.

If m_1 is not a singleton, then $m_1 \xrightarrow{\sigma} m_2$ is due to rule (s-com); the two premise transitions are $m'_1 \xrightarrow{\sigma_1} m'_2$ and $m''_1 \xrightarrow{\sigma_2} m''_2$, with Sync$(\sigma_1, \sigma_2, \sigma)$, $m_1 = m'_1 \oplus m''_1$ and $m_2 = m'_2 \oplus m''_2$. As $dom(m_1)$ is well behaved, also $dom(m'_1)$ and $dom(m''_1)$ are well behaved; so, induction can be applied to conclude that σ_1 and σ_2 satisfy all the requirements of the proposition. In particular, by definition of Sync, either σ_1 or σ_2 is $\overline{\gamma}$ for some $\gamma \in \mathscr{G}$, and the other one is γ or an atomic sequence starting with γ; w.l.o.g., assume $\sigma_1 = \overline{\gamma}$ (and so, by induction, $|m'_1| = 1$) and $\sigma_2 = \gamma$, if $\sigma = \tau$, or $\sigma_2 = \gamma\sigma$. In the former case, the requirements of the proposition are trivially satisfied for $\sigma = \tau$; in the latter case, $\sigma_2 \in \mathscr{L}^+$, so that also $\sigma \in \mathscr{L}^+$, and so all the requirements of the proposition are satisfied for σ. □

Corollary 7.3. *If transition $t = (m_1, \overline{\gamma}, m_2)$ is derivable by the rules of Table 7.13 from a well-behaved $dom(m_1)$, then m_1 is a singleton. If transition $t = (m_1, \sigma, m_2)$ is derivable by rule (s-com) from a well-behaved $dom(m_1)$, then m_1 is not a singleton and $\sigma \in \{\tau\} \cup \mathscr{G}^+$.* □

Remark 7.4. By the proof of Proposition 7.29 and by the corollary above, it is clear that any transition $t = m_1 \xrightarrow{\sigma} m_2$ derivable from a *well-behaved* set of places $dom(m_1)$ is such that in the proof tree for t, whenever rule (s-com) is used with premise transitions t_1 and t_2, at least one of the two, say t_1 w.l.o.g., is such that ${}^\bullet t_1$ is a singleton and $l(t_1)$ is a single (possibly restricted) output action in $\overline{\mathscr{G}}$. That is, any derivable transition t from a well-behaved set of places $dom(m_1)$ is such that one sequential process $s \in dom(m_1)$ acts as the *leader* of the synchronization by performing one (possibly restricted) input (if the synchronization is binary) or an atomic sequence of (possibly restricted) inputs (if the synchronization is multi-party), while the other sequential components each contribute with a single (possibly restricted) output action, acting as *servants*. □

We now want to prove that for any finite, well-behaved set of places $S \subseteq S_{FNM}$, the set of transitions statically enabled at S is finite. An auxiliary lemma is necessary. Given a single place $s \in S$, by $s \vdash t$ we mean that transition $t = (\{s\}, \sigma, m)$ is derivable by the rules in Table 7.13, hence with $\sigma \in \mathscr{A}^\gamma$.

Lemma 7.11. *The set $T_s = \{t \mid s \vdash t\}$ is finite, for any $s \in S_{FNM}$.*

Proof. By induction on the axiom and rules in Table 7.13. □

Given a finite, well-behaved set of places $S \subseteq S_{FNM}$, let T_1^S be $\bigcup_{s \in S} T_s$, i.e., the set of all transitions, with a singleton pre-set in S, derivable by the rules with labeling in $\mathscr{A}^\gamma$. The set T_1^S is finite, being the finite union (as S is finite) of finite sets (as T_s is finite for any s, by Lemma 7.11).

Let $k \in \mathbb{N}$ be the length of the longest label of any transition in T_1^S. Remark 7.4 explains that if a multi-party transition t is derivable by the rules from the well-behaved set S, then its proof contains k synchronizations at most, each one between a transition (labeled with a sequence of inputs) and a *singleton-pre-set* transition (labeled with a *single* output action); hence, at most $k+1$ participants can take part in a multi-party synchronization. Therefore, the set of all the transitions statically enabled at a finite, well-behaved set S can be defined by means of a sequence of sets T_i^S of transitions, for $2 \leq i \leq k+1$, where each transition $t \in T_i^S$ has a pre-set ${}^\bullet t$ of size i, as follows:

$$\begin{aligned} T_i^S = \{(m_1 \oplus m_2, \sigma, m_1' \oplus m_2') \mid\ & ad(m_1 \oplus m_2), \\ & \exists \sigma_1 \exists \gamma.(m_1, \sigma_1, m_1') \in T_{i-1}^S, (m_2, \overline{\gamma}, m_2') \in T_1^S, Sync(\sigma_1, \overline{\gamma}, \sigma)\}. \end{aligned}$$

Note that T_2^S is finite, because T_1^S is finite; inductively, T_{i+1}^S, for $2 \leq i \leq k$ is finite, because T_i^S and T_1 are finite. In conclusion, the set T_S of all the transitions statically enabled at S is

$$T_S = \{t \mid t \in \bigcup_{i=1}^{k+1} T_i^S \wedge l(t) \in \mathscr{A}\},$$

where only transitions labeled over $\mathscr{A}$ are considered. T_S is finite, being a finite union of finite sets; therefore, we have the following result.

Theorem 7.9. *If $S \subseteq S_{FNM}$ is a finite, well-behaved set of places, then set $T_S \subseteq T_{FNM}$ of all the transitions statically enabled at S is finite.* □

Example 7.8. **(What can go wrong for a non-well-behaved set?)** Consider the process $p = (\nu a)(a.\mathbf{0} \mid (\underline{a}.\overline{a}.\mathbf{0} \mid \overline{a}.\mathbf{0}))$, discussed in Example 7.7, which is not well formed because of the subterm $\underline{a'}.\overline{a'}.\mathbf{0}$. As p is not well formed, we have that $dec(p) = a'.\mathbf{0} \oplus \underline{a'}.\overline{a'}.\mathbf{0} \oplus \overline{a'}.\mathbf{0}$ is not well behaved. It is easy to observe that transition $t_1 = a'.\mathbf{0} \oplus \underline{a'}.\overline{a'}.\mathbf{0} \oplus \overline{a'}.\mathbf{0} \xrightarrow{\tau} \emptyset$ is derivable, because first we synchronize $a'\overline{a'}$ with $\overline{a'}$, yielding $\overline{a'}$, which is then synchronized with a', yielding τ. However, the occurrence of action $\overline{a'}$ produced by the first synchronization may be used to synchronize an additional sequence $a'\overline{a'}$, yielding $\overline{a'}$ again. Therefore, it is not difficult to see that also $t_n = a'.\mathbf{0} \oplus n \cdot \underline{a'}.\overline{a'}.\mathbf{0} \oplus \overline{a'}.\mathbf{0} \xrightarrow{\tau} \emptyset$, is statically enabled at $dom(dec(p))$, for any $n \geq 1$. Hence, the set of transitions statically enabled at $dom(dec(p))$ is infinite. □

Example 7.9. **(1/3 Semi-counter)** Continuing Example 5.13, let us consider a semi-counter such that three occurrences of *inc* are needed to enable one *dec*, whence the name 1/3 semi-counter. The well-formed process $p = (\nu c)A$, where

$$A \doteq inc.(A \mid (\underline{c}.\underline{c}.dec.\mathbf{0} + \overline{c}.\mathbf{0}))$$

is a 1/3 semi-counter, indeed. The initial marking m_0 is $dec(p) = dec((\nu c)A) = dec(A)\{c'/c\} = \{A\}\{c'/c\} = \{s_1\}$; place s_1 is the extended, sequential process $A_{\{c'/c\}}$, where the constant $A_{\{c'/c\}}$ is obtained by applying the substitution $\{c'/c\}$ to the body of A:

$A_{\{c'/c\}} \doteq inc.(A_{\{c'/c\}} \mid (\underline{c'}.\underline{c'}.dec.\mathbf{0} + \overline{c'}.\mathbf{0}))$.

Then, let $S_0 = dom(m_0) = \{s_1\}$. The set of transitions statically enabled at S_0 is $T_1^{S_0} = T_{s_1} = \{t_1\}$, where the only transition is $t_1 = \{s_1\} \xrightarrow{inc} \{s_1, s_2\}$, with $s_2 = \underline{c'}.\underline{c'}.dec.\mathbf{0} + \overline{c}'.\mathbf{0}$. Therefore, the new set of statically reachable places is $S_1 = \{s_1, s_2\}$. Note that s_2 can produce two transitions in T_{s_2}, namely $t' = \{s_2\} \xrightarrow{c'c'dec} \emptyset$ and $t'' = \{s_2\} \xrightarrow{\overline{c'}} \emptyset$, but neither is labeled over $\mathscr{A}$. Since the longest label has length 3, we have to compute the sets $T_i^{S_1}$ for $i = 1, \ldots, 4$:

$T_1^{S_1} = \{t_1, t', t''\}$,

$T_2^{S_1} = \{t'''\}$, where $t''' = (\{s_2, s_2\}, c'dec, \emptyset)$,

$T_3^{S_1} = \{t_2\}$, where $t_2 = (\{s_2, s_2, s_2\}, dec, \emptyset)$, whose proof is shown in Table 7.15,

$T_4^{S_1} = \emptyset$.

Hence, $T_{S_1} = \{t_1, t_2\}$, as these two are the only transitions labeled over $\mathscr{A}$. As t_2 does not add any new reachable place, we have that S_1 is the set of places statically reachable from the initial marking, and T_{S_1} is the set of transitions statically enabled at S_1. This net is depicted in Figure 7.1. □

7.4.4 The Reachable Subnet $Net(p)$

The P/T net system associated with a process $p \in \mathscr{P}_{FNM}$ is the subnet of N_{FNM} statically reachable from the initial marking $dec(p)$, denoted by $Net(p)$.

$$\text{(s-com)}\;\dfrac{\text{(s-com)}\;\dfrac{\text{(sum}_1\text{)}\;\dfrac{\text{(s-pref)}\;\dfrac{\text{(s-pref)}\;\dfrac{\text{(pref)}\;\dfrac{}{\{dec.\mathbf{0}\} \xrightarrow{dec} \emptyset}}{\{\underline{c'}.dec.\mathbf{0}\} \xrightarrow{c'dec} \emptyset}}{\{\underline{c'}.\underline{c'}.dec.\mathbf{0}\} \xrightarrow{c'c'dec} \emptyset}}{\{s_2\} \xrightarrow{c'c'dec} \emptyset} \qquad \text{(sum}_2\text{)}\;\dfrac{\text{(pref)}\;\dfrac{}{\{\overline{c'}.\mathbf{0}\} \xrightarrow{\overline{c'}} \emptyset}}{\{s_2\} \xrightarrow{\overline{c'}} \emptyset}}{\{s_2, s_2\} \xrightarrow{c'dec} \emptyset} \qquad \text{(sum}_2\text{)}\;\dfrac{\text{(pref)}\;\dfrac{}{\{\overline{c'}.\mathbf{0}\} \xrightarrow{\overline{c'}} \emptyset}}{\{s_2\} \xrightarrow{\overline{c'}} \emptyset}}{\{s_2, s_2, s_2\} \xrightarrow{dec} \emptyset}$$

Table 7.15 The proof of a net transition, where $s_2 = \underline{c'}.\underline{c'}.dec.\mathbf{0} + \overline{c}'.\mathbf{0}$

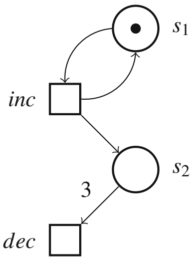


Fig. 7.1 The P/T net for the $1/3$ semi-counter

Definition 7.3. Let p be a process in $\mathscr{P}_{FNM}$. The P/T net system statically associated with p is $Net(p) = (S_p, A_p, T_p, m_0)$, where $m_0 = dec(p)$ and

$$\begin{aligned} S_p &= [\![dom(m_0)\rangle \quad \text{computed in } N_{FNM}, \\ T_p &= \{t \in T_{FNM} \mid S_p[\![t\rangle\}, \\ A_p &= \{\sigma \in \mathscr{A} \mid \exists t \in T_p \text{ such that } l(t) = \sigma\}. \end{aligned}$$ □

The following propositions present three facts that are obviously true by construction of the net $Net(p)$ associated with an FNM process p.

Proposition 7.30. *For any $p \in \mathscr{P}_{FNM}$, $Net(p)$ is a statically reduced P/T net.* □

Proposition 7.31. *If $dec(p) = dec(q)$, then $Net(p) = Net(q)$.* □

Proposition 7.32. *For any restriction-free $t \in \mathscr{P}_{FNM}$ and for any $L \subseteq \mathscr{L}$, the following hold.*

i) If $Net(t) = (S, A, T, m_0)$, then, for any $n \geq 1$, $Net(t^n) = (S, A, T, n \cdot m_0)$, where $t^1 = t$ and $t^{n+1} = t \,|\, t^n$.

ii) If $Net((\nu L)t) = (S, A, T, m_0)$, then $Net((\nu L)(t^n)) = (S, A, T, n \cdot m_0)$, for any $n \geq 1$.

Proof. If $dec(t) = m_0$, then $dec(t^n) = n \cdot m_0$. The thesis follows by observing that $dom(m_0) = dom(n \cdot m_0)$, so that the starting set of places from which to compute all the statically reachable places is the same for both nets. Similarly, when considering processes $(\nu L)t$ and $(\nu L)(t^n)$. □

Definition 7.3 suggests a way of generating $Net(p)$ with an algorithm based on the inductive definition of the static reachability relation (see Definition 3.9): Start with the initial set of places $S_0 = dom(dec(p))$, and then apply the rules in Table 7.13 in order to produce the set T_{S_0} of transitions (labeled over $\mathscr{A}$) statically enabled at S_0, as well as the additional places statically reachable by means of such transitions. Then repeat this procedure from the set of places statically reached so far. An instance of this procedure was given in Example 7.9. There are two problems with this algorithm:

- the obvious *halting condition* is "until no new places are statically reachable"; of course, the algorithm terminates if we know that the set S_p of places statically reachable from $dom(dec(p))$ is finite; additionally,
- at each step of the algorithm, we have to be sure that the set of transitions derivable from the current set of statically reachable places is finite.

We are going to prove only the first requirement — S_p is finite for any $p \in \mathscr{P}_{FNM}$ — because it implies also the second one for well-formed processes. As a matter of fact, if p is well formed, then $dom(dec(p))$ is well behaved by Theorem 7.7, and so is any set S of places statically reachable from $dom(dec(p))$ by Corollary 7.2; since $S \subseteq S_p$, S is also finite, and so, by Theorem 7.9, the set T_S of transitions statically enabled at the finite, well-behaved set S is finite, too.

Theorem 7.10. *For any $p \in \mathscr{P}_{FNM}$, let $Net(p) = (S_p, A_p, T_p, m_0)$ be defined as in Definition 7.3. Then, the set S_p is finite.*

Proof. We prove, by induction on the static reachability relation $\Longrightarrow^*$, that any set S_i of places, statically reachable from $dom(m_0)$, is a subset of $sub(dom(m_0))$. This is enough as, by Corollary 7.1, we know that $sub(dom(m_0)) = sub(dom(dec(p))) \subseteq sub(p)$; moreover, by Theorem 7.1, $sub(p)$ is finite and so the thesis follows trivially.

The base case is $dom(m_0) \Longrightarrow^* dom(m_0)$. By Proposition 7.21, we have the required $dom(m_0) \subseteq sub(dom(m_0))$. Now, let us assume that S_i is a set of places statically reachable from $dom(m_0)$ and let $t = m_1 \xrightarrow{\sigma} m_2$ be such that $S_i \overset{t}{\Longrightarrow} S_{i+1}$. By induction, we know that $S_i \subseteq sub(dom(m_0))$. So, we have to prove that the new places reached via t are in $sub(dom(m_0))$. Note that since $dom(m_1) \subseteq S_i$, it follows that $dom(m_1) \subseteq sub(dom(m_0))$ and also that $sub(dom(m_1)) \subseteq sub(dom(m_0))$, by Proposition 7.20. By Proposition 7.21, we have that $dom(m_2) \subseteq sub(dom(m_2))$; by Proposition 7.23, we have that $sub(dom(m_2)) \subseteq sub(dom(m_1))$; by transitivity, $dom(m_2) \subseteq sub(dom(m_0))$, and so $S_{i+1} = S_i \cup dom(m_2) \subseteq sub(dom(m_0))$, as required.

Summing up, any place statically reachable from $dom(m_0)$ is a sequential subterm of p. Since, by Theorem 7.1, $sub(p)$ is finite, then also S_p (the largest set of places statically reachable from $dom(m_0)$) is finite. □

Theorem 7.11. *For any well-formed FNM process p, $Net(p) = (S_p, A_p, T_p, dec(p))$ is a finite P/T net.*

Proof. The set $S_0 = dom(dec(p))$ is finite, by Proposition 7.19, and it is also well-behaved by Theorem 7.7. By Theorem 7.9, the set T_{S_0} of all the transitions statically *enabled at S_0 is finite. Let S_1 be the set of places $S_0 \cup \bigcup_{t \in T_{S_0}} dom(t^\bullet)$. If $S_1 = S_0$, then $S_p = S_0$ and $T_p = T_{S_0}$. Otherwise, repeat the step above for S_1; in fact, S_1 is a finite set of places, because S_0 is finite, the set T_{S_0} is finite and each transition has a finite post-set; moreover, S_1 is well behaved by Corollary 7.2. By repeating the step above for S_1, we compute a new finite set T_{S_1} of transitions statically enabled at S_1, and a new finite, well-behaved set S_2 of places statically reachable from S_1 via the transitions in T_{S_1}; if $S_2 = S_1$, then $S_p = S_1$ and $T_p = T_{S_1}$. Otherwise, repeat the step above for S_2. This procedure will end eventually because, by Theorem 7.10, we are sure that S_p is a finite set.* □

Hence, *only* finite P/T nets can be represented by well-formed FNM processes.

7.5 Representing All Finite P/T Nets

It is not a surprise that *all* finite P/T nets can be represented by well-formed FNM processes. The construction described in Section 6.5 for finite CCS nets is to be modified non-trivially in order to cope with transitions having a pre-set of size larger than two. However, the basic ingredients are all already present in the construction of Section 6.5.

As for the previous case, the translation from nets to processes defines a constant C_i in correspondence with each place s_i; the constant C_i has a summand c_i^j for each transition t_j, which is $\mathbf{0}$ when s_i is not in the pre-set of t_j. The FNM process $\mathscr{T}_{FNM}(N(m_0))$ associated with the finite net system $N(m_0)$, labeled over $\mathscr{L} \cup \{\tau\}$, has a bound name x_i^j for each pair (s_i, t_j), where s_i is a place and t_j is a transition; such bound names are used to force synchronization among the components participating in transition t_j with pre-set of cardinality two or more. Among the many places in the pre-set of t_j, the one with least index i (as we assume that places are indexed) plays the role of *leader* of the synchronization; the corresponding leader constant C_i has a summand c_i^j containing the atomic input sequence needed for the multi-party synchronization, such that each strong input prefix x_h^j (for $h > i$) is synchronized with the corresponding output $\overline{x}_h^j$ performed by the *servant* participant of index h; in case ${}^\bullet t_j(s_i) \geq 2$, then the summand c_i^j is actually a sum of $\overline{x}_i^j.\mathbf{0}$ with the atomic input sequence, so that one instance of C_i acts as the leader, while the others are servants.

Definition 7.4. (Translating finite P/T nets into well-formed FNM processes) Given $A \subseteq \mathscr{L} \cup \{\tau\}$, let $N(m_0) = (S, A, T, m_0)$ — with $S = \{s_1, \dots, s_n\}$, $T = \{t_1, \dots, t_k\}$, and $l(t_j) = \mu_j$ — be a finite P/T net. Function $\mathscr{T}_{FNM}(-)$, from finite P/T nets to well-formed FNM processes, is defined as

$$\mathscr{T}_{FNM}(N(m_0)) = (\nu L)(\underbrace{C_1 | \cdots | C_1}_{m_0(s_1)} | \cdots | \underbrace{C_n | \cdots | C_n}_{m_0(s_n)})$$

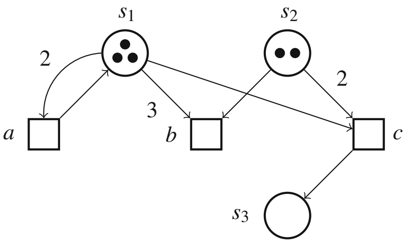


Fig. 7.2 A simple net

where $L = \{x_1^1, \dots, x_n^1, x_1^2, \dots, x_n^2, \dots, x_1^k, \dots, x_n^k\}$ is such that $L \cap A = \emptyset$, each C_i is equipped with a defining equation $C_i \doteq c_i^1 + \cdots + c_i^k$ (with $C_i \doteq \mathbf{0}$ if $k = 0$), and each summand c_i^j, for $j = 1, \dots, k$, is equal to

- $\mathbf{0}$, if $s_i \not\in {}^\bullet t_j$;
- $\mu_j.\Pi_j$, if ${}^\bullet t_j = \{s_i\}$;
- $\overline{x}_i^j.\mathbf{0}$, if ${}^\bullet t_j(s_i) > 0$ and ${}^\bullet t_j(s_{i'}) > 0$ for some $i' < i$ (i.e., s_i is not the leader for the synchronization on t_j);
- $\underbrace{\underline{x}_{i+1}^j . \cdots . \underline{x}_{i+1}^j}_{{}^\bullet t_j(s_{i+1})} . \cdots . \underbrace{\underline{x}_n^j . \cdots . \underline{x}_n^j}_{{}^\bullet t_j(s_n)} .\mu_j.\Pi_j$, if ${}^\bullet t_j(s_i) = 1$ and s_i is the leader of the synchronization (i.e., ${}^\bullet t_j(s_{i'}) > 0$ for no $i' < i$, while ${}^\bullet t_j(s_{i'}) > 0$ for some $i' > i$);
- $\overline{x}_i^j.\mathbf{0} + \underbrace{\underline{x}_i^j . \cdots . \underline{x}_i^j}_{{}^\bullet t_j(s_i)-1} . \underbrace{\underline{x}_{i+1}^j . \cdots . \underline{x}_{i+1}^j}_{{}^\bullet t_j(s_{i+1})} . \cdots . \underbrace{\underline{x}_n^j . \cdots . \underline{x}_n^j}_{{}^\bullet t_j(s_n)} .\mu_j.\Pi_j$, otherwise (i.e., s_i is the leader and ${}^\bullet t_j(s_i) \geq 2$).

Finally, process Π_j is $\underbrace{C_1 | \cdots | C_1}_{t_j^\bullet(s_1)} | \cdots | \underbrace{C_n | \cdots | C_n}_{t_j^\bullet(s_n)}$, meaning that $\Pi_j = \mathbf{0}$ if $t_j^\bullet = \emptyset$. □

Example 7.10. Consider the net $N(m_0)$ of Figure 7.2, where transition t_1 is labeled with a, t_2 with b and t_3 with c. Applying the translation above, we obtain the well-formed, FNM process

$$\mathscr{T}_{FNM}(N(m_0)) = (\nu L)(C_1 \,|\, C_1 \,|\, C_1 \,|\, C_2 \,|\, C_2)$$

where $L = \{x_1^1, x_2^1, x_3^1, x_1^2, x_2^2, x_3^2, x_1^3, x_2^3, x_3^3\}$, and

$$\begin{aligned}
C_1 &\doteq (\overline{x}_1^1.\mathbf{0} + \underline{x}_1^1.a.C_1) + (\overline{x}_1^2.\mathbf{0} + \underline{x}_1^2.\underline{x}_1^2.\underline{x}_2^2.b.\mathbf{0}) + \underline{x}_2^3.\underline{x}_2^3.c.C_3, \\
C_2 &\doteq \mathbf{0} + \overline{x}_2^2.\mathbf{0} + \overline{x}_2^3.\mathbf{0}, \\
C_3 &\doteq \mathbf{0} + \mathbf{0} + \mathbf{0}.
\end{aligned}$$

□

Note that $\mathscr{T}_{FNM}(N(m_0))$ is an FNM process: in fact, the restriction operator occurs only at the top level, applied to the parallel composition of a finite number of constants; each constant has a body that is sequential and restriction-free. Note also that $\mathscr{T}_{FNM}(N(m_0))$ is a *well-formed* process: in fact, each *strong prefix* is a bound input x_i^j, and any sequence ends with an action $\mu_j \in A$, which is either an input or τ; hence, no atomic sequence ends with an output. Therefore, the following proposition holds by Theorem 7.11 and Proposition 7.30.

Proposition 7.33. *For any finite P/T Petri net $N(m_0)$, the net $Net(\mathscr{T}_{FNM}(N(m_0)))$ is a finite, statically reduced P/T net.* □

Moreover, $\mathscr{T}_{FNM}(N(m_0))$ is *output-closed*, as $fn(\mathscr{T}_{FNM}(N(m_0))) \cap \overline{\mathscr{L}} = \emptyset$; in fact, any output occurring in this term is of the form $\overline{x}_i^j$ for suitable i and j, and such a name is bound.

Now we are ready to state our main result, the so-called *representability theorem*.

Theorem 7.12. (Representability theorem 5) *Let $N(m_0) = (S, A, T, m_0)$ be a finite, statically reduced P/T net system such that $A \subseteq \mathscr{L} \cup \{\tau\}$, and let $p = \mathscr{T}_{FNM}(N(m_0))$. Then, $Net(p)$ is isomorphic to $N(m_0)$.*

Proof. *Let $N(m_0) = (S, A, T, m_0)$ be a reduced, finite P/T net, with $S = \{s_1, \ldots, s_n\}$, $A \subseteq \mathscr{L} \cup \{\tau\}$, $T = \{t_1, \ldots, t_k\}$ and $l(t_j) = \mu_j$ for $j = 1, \ldots, k$. The associated FNM process is*

$$\mathscr{T}_{FNM}(N(m_0)) = (\nu L)(\underbrace{C_1|\cdots|C_1}_{m_0(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m_0(s_n)}),$$

where $L = \{x_1^1, \ldots, x_n^1, x_1^2, \ldots, x_n^2, \ldots, x_1^k, \ldots, x_n^k\}$, $L \cap A = \emptyset$, and for each place s_i we have a corresponding constant $C_i \doteq \sum_{j=1}^k c_i^j$, defined as in Definition 7.4. For notational convenience, $\sum_{j=1}^k c_i^j$ is denoted by p_i, i.e., $C_i \doteq p_i$; for the same reason, we use p to denote $\mathscr{T}_{FNM}(N(m_0))$.

Let $\theta = \{L'/L\}$ be a substitution that maps each bound name x_i^j to its corresponding restricted name $x_i'^j$ in $\mathscr{L}'$, for $i = 1, \ldots, n$ and $j = 1, \ldots, k$. Let $Net(p) = (S', A', T', m_0')$. Then, $m_0' = dec(p)$ is the multiset

$$dec((\nu L)(\underbrace{C_1|\cdots|C_1}_{m_0(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m_0(s_n)})) = dec(\underbrace{C_1|\cdots|C_1}_{m_0(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m_0(s_n)})\theta =$$
$$m_0(s_1) \cdot C_1\theta \oplus \cdots \oplus m_0(s_n) \cdot C_n\theta,$$

where $C_i\theta$ gives rise to the new constant $C_{i,\theta} \doteq p_i\theta$. Hence, the initial places are all of the form $C_i\theta$, where such a place is present in m_0' only if $m_0(s_i) > 0$.

Note that, by Definition 7.4, any transition $t' \in T'$ with ${}^\bullet t' \subseteq dec(p)$ is such that, for some suitable j, $t'^\bullet = dec(\Pi_j)\theta$, and so equal to $k_1 \cdot C_1\theta \oplus k_2 \cdot C_2\theta \oplus \ldots \oplus k_n \cdot C_n\theta$ for suitable $k_i \geq 0$, $i = 1, \ldots, n$; by iterating this observation, each transition in T' has a post-set of the form $dec(\Pi_j)\theta$, for some suitable $j = 1, \ldots, k$. Hence, each statically reachable place s_i' in S' is of the form $C_i\theta$. Moreover, by Proposition 7.33, $Net(p)$ is statically reduced, implying that all the $C_i\theta$'s are statically reachable, for $i = 1, \ldots, n$. Hence, there is a bijection $f : S \to S'$ defined by $f(s_i) = s_i' = C_i\theta$, which is the natural candidate isomorphism function. To prove that f is an isomorphism, we have to prove that

1. $f(m_0) = m'_0$,
2. $t = (m, \mu, m') \in T$ implies $f(t) = (f(m), \mu, f(m')) \in T'$, and
3. $t' = (m'_1, \mu, m'_2) \in T'$ implies there exists $t = (m_1, \mu, m_2) \in T$ such that $f(t) = t'$, i.e., $f(m_1) = m'_1$ and $f(m_2) = m'_2$.

From items 2) and 3) above, it follows that $A = A'$.

Proof of *1: Let $m_0 = k_1 \cdot s_1 \oplus k_2 \cdot s_2 \oplus \cdots \oplus k_n \cdot s_n$, where $k_i = m_0(s_i) \geq 0$ for $i = 1, \ldots, n$. The mapping via f of the initial marking m_0 is*

$$f(m_0) = k_1 \cdot f(s_1) \oplus \cdots \oplus k_n \cdot f(s_n) = k_1 \cdot C_1\theta \oplus \cdots \oplus k_n \cdot C_n\theta$$
$$= dec(\underbrace{C_1 | \cdots | C_1}_{k_1\, times} | \cdots | \underbrace{C_n | \cdots | C_n}_{k_n\, times})\theta = dec(p) = m'_0.$$

Proof of 2*: We prove that, for $j = 1, \ldots, k$, if $t_j = (m, \mu_j, m') \in T$, then $t'_j = (f(m), \mu_j, f(m')) \in T'$. We now examine the possible cases, by inspecting the shape of ${}^\bullet t_j$. Let us examine the three cases separately:*

- *${}^\bullet t_j = \{s_i\}$ and so $f({}^\bullet t_j) = \{C_i\theta\}$. By Definition 7.4, for $C_i = p_i$, we have in p_i a summand $c^j_i = \mu_j.\Pi_j$, with $\Pi_j = \underbrace{C_1 | \cdots | C_1}_{m'(s_1)} | \cdots | \underbrace{C_n | \cdots | C_n}_{m'(s_n)}$. Therefore, not only transition $\{c^j_i\} \xrightarrow{\mu_j} dec(\Pi_j)$ is derivable by axiom (pref), but also $\{p_i\} \xrightarrow{\mu_j} dec(\Pi_j)$ is derivable by (possibly repeated applications of) rules (sum$_1$) and (sum$_2$), and so $\{C_i\} \xrightarrow{\mu_j} dec(\Pi_j)$, by rule (cons). Hence, by Proposition 7.26, $\{C_i\theta\} \xrightarrow{\mu_j} dec(\Pi_j)\theta$ is derivable, as $\mu_j \notin L$. In conclusion, starting from transition $t_j = (\{s_i\}, \mu_j, m')$, we have shown that transition $t'_j = (\{C_i\theta\}, \mu_j, dec(\Pi_j)\theta)$, with $f(m') = dec(\Pi_j)\theta$, belongs to T', as required.*
- *${}^\bullet t_j = s_i \oplus \overline{m}$, such that $\overline{m}(s_h) > 0$ only if $h > i$ (i.e., s_i is the leader and ${}^\bullet t_j(s_i) = 1$), and so $f({}^\bullet t_j) = C_i\theta \oplus f(\overline{m})$. By Definition 7.4, for $C_i = p_i$, we have in p_i a summand $c^j_i = \underbrace{\underline{x}^j_{i+1}. \cdots .\underline{x}^j_{i+1}}_{\overline{m}(s_{i+1})} . \cdots . \underbrace{\underline{x}^j_n . \cdots .\underline{x}^j_n}_{\overline{m}(s_n)} .\mu_j.\Pi_j$, such that $\{c^j_i\} \xrightarrow{\sigma} dec(\Pi_j)$ is derivable by repeated applications of rule (s-pref), starting from one instance of axiom (pref), namely $\{\mu_j.\Pi_j\} \xrightarrow{\mu_j} dec(\Pi_j)$, where σ is the atomic sequence of the strong input prefixes ending with μ_j; but also, by (possibly repeated applications of) rules (sum$_1$) and (sum$_2$), $\{p_i\} \xrightarrow{\sigma} dec(\Pi_j)$ is derivable, and so $\{C_i\} \xrightarrow{\sigma} dec(\Pi_j)$ by rule (cons). Each C_h such that $\overline{m}(s_h) > 0$ contributes its summand $c^j_h = \overline{x}^j_h.\mathbf{0}$, so that $\{c^j_h\} \xrightarrow{\overline{x}^j_h} \emptyset$ is derivable by axiom (pref), but also, by (possibly repeated applications of) rules (sum$_1$) and (sum$_2$), $\{p_h\} \xrightarrow{\overline{x}^j_h} \emptyset$ is derivable, and so $\{C_h\} \xrightarrow{\overline{x}^j_h} \emptyset$, by rule (cons). Therefore, by repeated applications of rule (s-com), the transition $\{C_i\} \xrightarrow{\sigma} dec(\Pi_j)$ can be synchronized with each*

instance of $\{C_h\} \xrightarrow{\overline{x}_h^j} \emptyset$ for each suitable h, so that at the end we get the transition $dec(P_j) \xrightarrow{\mu_j} dec(\Pi_j)$, where $P_j = (C_i | \underbrace{C_{i+1} | \cdots | C_{i+1}}_{\overline{m}(s_{i+1})} | \cdots | \underbrace{C_n | \cdots | C_n}_{\overline{m}(s_n)})$ and $\Pi_j = (\underbrace{C_1 | \cdots | C_1}_{m'(s_1)} | \cdots | \underbrace{C_n | \cdots | C_n}_{m'(s_n)})$. Finally, also transition $dec(P_j)\theta \xrightarrow{\mu_j} dec(\Pi_j)\theta$ is derivable by Proposition 7.26, as $\mu_j \notin L$, where $dec(P_j)\theta = f(s_i \oplus \overline{m}) = f({}^\bullet t_j)$ and $dec(\Pi_j)\theta = f(m') = f(t_j^\bullet)$. In conclusion, starting from transition $t_j = (s_i \oplus \overline{m}, \mu_j, m')$, we have shown that transition $t'_j = (f(s_i \oplus \overline{m}), \mu_j, f(m'))$ belongs to T', as required.

- *${}^\bullet t_j = k \cdot s_i \oplus \overline{m}$, such that $\overline{m}(s_h) > 0$ only if $h > i$ (i.e., s_i is the leader and ${}^\bullet t_j(s_i) = k \geq 2$), and so $f({}^\bullet t_j) = k \cdot C_i\theta \oplus f(\overline{m})$. According to Definition 7.4, for $C_i = p_i$, we have in p_i a summand $c_i^j = \overline{x}_i^j.\mathbf{0} + \underbrace{\underline{x}_i^j.\cdots.\underline{x}_i^j}_{k-1}.\underbrace{\underline{x}_{i+1}^j.\cdots.\underline{x}_{i+1}^j}_{\overline{m}(s_{i+1})}.\cdots.\underbrace{\underline{x}_n^j.\cdots.\underline{x}_n^j}_{\overline{m}(s_n)}.\mu_j.\Pi_j$, with $\Pi_j = \underbrace{C_1 | \cdots | C_1}_{m'(s_1)} | \cdots | \underbrace{C_n | \cdots | C_n}_{m'(s_n)}$. Of course, $\{c_i^j\} \xrightarrow{\overline{x}_i^j} \emptyset$ is derivable by axiom (pref) and rule (sum$_1$), so that $\{p_i\} \xrightarrow{\overline{x}_i^j} \emptyset$ by (possibly repeated applications of) rules (sum$_1$) or (sum$_2$), and, finally, $\{C_i\} \xrightarrow{\overline{x}_i^j} \emptyset$, by rule (cons); actually, $k-1$ instances of C_i will perform this transition; one single instance of C_i will perform $\{C_i\} \xrightarrow{\sigma} dec(\Pi_j)$, where σ is the atomic sequence of the strong input prefixes starting with $k-1$ occurrences of action x_i^j and ending with μ_j, which is derivable by rule (cons), then by (possibly repeated applications of) (sum$_1$) or (sum$_2$), then by one application of (sum$_2$), then repeated applications of rule (s-pref), starting from one instance of axiom (pref), namely $\{\mu_j.\Pi_j\} \xrightarrow{\mu_j} dec(\Pi_j)$. Moreover, each C_h such that $\overline{m}(s_h) > 0$ contributes a transition $\{C_h\} \xrightarrow{\overline{x}_h^j} \emptyset$, as discussed in the previous item. Therefore, by repeated applications of rule (s-com), the transition $\{C_i\} \xrightarrow{\sigma} dec(\Pi_j)$ can be synchronized with the $k-1$ instances of $\{C_i\} \xrightarrow{\overline{x}_i^j} \emptyset$ and with all the other transitions of the form $\{C_h\} \xrightarrow{\overline{x}_h^j} \emptyset$ for each suitable h, so that at the end we get the transition $dec(P_j) \xrightarrow{\mu_j} dec(\Pi_j)$, where $P_j = (\underbrace{C_i | \cdots | C_i}_{k} | \underbrace{C_{i+1} | \cdots | C_{i+1}}_{\overline{m}(s_{i+1})} | \cdots | \underbrace{C_n | \cdots | C_n}_{\overline{m}(s_n)})$ and $\Pi_j = (\underbrace{C_1 | \cdots | C_1}_{m'(s_1)} | \cdots | \underbrace{C_n | \cdots | C_n}_{m'(s_n)})$. Finally, also transition $dec(P_j)\theta \xrightarrow{\mu_j} dec(\Pi_j)\theta$ is derivable by Proposition 7.26, as $\mu_j \notin L$, where $dec(P_j)\theta = f(k \cdot s_i \oplus \overline{m}) = f({}^\bullet t_j)$ and $dec(\Pi_j)\theta = f(m') = f(t_j^\bullet)$. In conclusion, starting from transition $t_j = (k \cdot s_i \oplus \overline{m}, \mu_j, m')$, we have shown that transition $t'_j = (f(k \cdot s_i \oplus \overline{m}), \mu_j, f(m'))$ belongs to T', as required.*

Proof of *3: We prove that if $t'_j = (m'_1, \mu_j, m'_2) \in T'$, then there exists a transition $t_j = (m_1, \mu_j, m_2) \in T$ such that $f(m_1) = m'_1$ and $f(m_2) = m'_2$. This is proved by case analysis on the three possible shapes of the marking m'_1, which can be $\{C_i\theta\}$, or $dec(P_j)\theta$, where $P_j = (\underbrace{C_i|\cdots|C_i}_{k_i}|\underbrace{C_{i+1}|\cdots|C_{i+1}}_{k_{i+1}}|\cdots|\underbrace{C_n|\cdots|C_n}_{k_n})$, with either $k_i = 1$, or $k_i \geq 2$.*

- *If $m'_1 = \{C_i\theta\}$ for some $i = 1,\ldots,n$, then $t'_j = \{C_i\theta\} \xrightarrow{\mu_j} m'_2$. By Proposition 7.26, also transition $\{C_i\} \xrightarrow{\mu_j} m''_2$ is derivable, with $m''_2\theta = m'_2$. According to Definition 7.4, such a transition is derivable by the rules only if among the many summands composing p_i, there exists a summand $c_i^j = \mu_j.\Pi_j$, which is possible only if in $N(m_0)$ we have a transition t_j with ${}^\bullet t_j = \{s_i\}$, $f(\{s_i\}) = \{C_i\theta\}$, $f(t_j^\bullet) = dec(\Pi_j)\theta = m'_2$ and $l(t_j) = \mu_j$, as required.*
- *If $m'_1 = C_i\theta \oplus k_{i+1} \cdot C_{i+1}\theta \oplus \ldots \oplus k_n \cdot C_n\theta$ for some $i = 1,\ldots,n$, then from transition $t'_j = m'_1 \xrightarrow{\mu_j} m'_2$, we get also transition $C_i \oplus k_{i+1} \cdot C_{i+1} \oplus \ldots \oplus k_n \cdot C_n \xrightarrow{\mu_j} m''_2$, with $m''_2\theta = m'_2$, by Proposition 7.26. According to Definition 7.4, such a transition is derivable by the rules only if among the many summands composing p_i, there exists a summand $c_i^j = \underbrace{\underline{x}_{i+1}^j.\cdots.\underline{x}_{i+1}^j}_{k_{i+1}}.\ldots.\underbrace{\underline{x}_n^j.\cdots.\underline{x}_n^j}_{k_n}.\mu_j.\Pi_j$, such that $\{C_i\} \xrightarrow{\sigma} dec(\Pi_j)$ is derivable, where σ is the atomic sequence of the strong input prefixes ending with μ_j; and moreover, each C_h, with $h > i$, such that x_h^j occurs in σ, has a summand $c_h^j = \overline{x}_h^j.\mathbf{0}$, so that $\{C_h\} \xrightarrow{\overline{x}_h^j} \emptyset$ is derivable. These summands are present only if in $N(m_0)$ we have a transition t_j with ${}^\bullet t_j = s_i \oplus k_{i+1} \cdot s_{i+1} \oplus \ldots \oplus k_n \cdot s_n$, $f({}^\bullet t_j) = m'_1$, $f(t_j^\bullet) = dec(\Pi_j)\theta = m'_2$ and $l(t_j) = \mu_j$, as required.*
- *If $m'_1 = k_i \cdot C_i\theta \oplus k_{i+1} \cdot C_{i+1}\theta \oplus \ldots \oplus k_n \cdot C_n\theta$, with $k_i \geq 2$, then from transition $t'_j = m'_1 \xrightarrow{\mu_j} m'_2$, we get also transition $k_i \cdot C_i \oplus k_{i+1} \cdot C_{i+1} \oplus \ldots \oplus k_n \cdot C_n \xrightarrow{\mu_j} m''_2$, with $m''_2\theta = m'_2$, by Proposition 7.26. According to Definition 7.4, such a transition is derivable by the rules only if among the many summands composing p_i, there exists a summand $c_i^j = \overline{x}_i^j.\mathbf{0} + \underbrace{\underline{x}_i^j.\cdots.\underline{x}_i^j}_{k_i-1}.\underbrace{\underline{x}_{i+1}^j.\cdots.\underline{x}_{i+1}^j}_{k_{i+1}}.\ldots.\underbrace{\underline{x}_n^j.\cdots.\underline{x}_n^j}_{k_n}.\mu_j.\Pi_j$ such that $\{C_i\} \xrightarrow{\sigma} dec(\Pi_j)$ is derivable, where σ is the atomic sequence of the strong input prefixes starting with $k_i - 1$ occurrences of x_i^j and ending with μ_j; and also $\{C_i\} \xrightarrow{\overline{x}_i^j} \emptyset$; and, additionally, each C_h, with $h > i$, such that x_h^j occurs in σ, has a summand $c_h^j = \overline{x}_h^j.\mathbf{0}$, so that $\{C_h\} \xrightarrow{\overline{x}_h^j} \emptyset$ is derivable. These summands are present only if in $N(m_0)$ we have a transition t_j with ${}^\bullet t_j = k_i \cdot s_i \oplus k_{i+1} \cdot s_{i+1} \oplus \ldots \oplus k_n \cdot s_n$, $f({}^\bullet t_j) = m'_1$, $f(t_j^\bullet) = dec(\Pi_j)\theta = m'_2$ and $l(t_j) = \mu_j$, as required.*

No further cases are possible, because a transition $t'_j = (m'_1, \mu_j, m'_2) \in T'$ must be derivable by the rules. □

Function $\mathscr{T}_{FNM}(-)$ can be applied to any finite P/T net $N(m_0)$. However, if $N(m_0)$ is not *statically reduced*, the representability theorem does not hold. Not surprisingly, the same net discussed in Example 6.13 for the case of FNC shows the problem also in this case.

In Remark 6.6 we argued that the extension of the approach described in this section to *communicating* Petri nets, labeled over the set $Act = \mathscr{L} \cup \overline{\mathscr{L}} \cup \{\tau\}$ of *structured* actions and co-actions, is not trivial. First of all, we noted that the translation in Definition 6.12 is no longer accurate; the same example in Remark 6.6 applies also to the case of FNM. A possible solution to this more general problem is outlined in Section 9.1.

7.5.1 Expressiveness

Two interesting results are proved in this section, by exploiting the representability theorem above. First, the class of languages recognized by finite P/T nets, labeled over $\mathscr{L} \cup \{\tau\}$, coincides with the class of languages recognized by finite CCS nets, labeled over $\mathscr{L} \cup \{\tau\}$. This is achieved by resorting to the encoding from output-closed, well-formed FNM processes to output-closed FNC processes of Table 7.8 in Section 7.2.1.

Theorem 7.13. (Finite P/T net languages = Finite CCS net languages) *The class of languages recognized by finite P/T nets, labeled over $\mathscr{L} \cup \{\tau\}$, coincides with the class of languages recognized by finite CCS nets, labeled over $\mathscr{L} \cup \{\tau\}$.*

Proof. By Definition 3.18, the language L_1 recognized by a finite P/T system $N_1(m_1)$ is the set of its weak completed traces, i.e., $L_1 = WCTr(IMG(N_1(m_1)))$. By Theorem 7.12, the finite P/T system $N_1(m_1)$ is represented, up to net isomorphism, by the output-closed, well-formed FNM process $p = \mathscr{T}_{FNM}(N_1(m_1))$, i.e., $N_1(m_1)$ and $Net(p)$ are isomorphic; hence, $L_1 = WCTr(IMG(Net(p)))$. By the soundness theorem (Theorem 7.15) presented in the next section, we have that p, in the step transition system semantics, is step bisimilar to $dec(p)$, in the step marking graph $SMG(Net(p))$, and so p and $dec(p)$ are also interleaving bisimilar, by Theorem 7.5, hence $WCTr(p) = L_1$ because interleaving bisimilarity implies weak completed trace equivalence. By the encoding in Table 7.8, the term $[\![p]\!]$ is an output-closed FNC process, such that $WCTr([\![p]\!]) = WCTr(p)$. By Corollary 6.3, the net semantics of $[\![p]\!]$ is the finite CCS net $Net([\![p]\!])$, such that its recognized language L_2 is $WCTr(IMG(Net([\![p]\!])))$. By Theorem 6.9 (soundness, up to step bisimilarity), $[\![p]\!]$ in the step transition system and $dec([\![p]\!])$ in the step marking graph are step bisimilar, and so also interleaving bisimilar by Proposition 6.7, hence $WCTr([\![p]\!]) = L_2$ because interleaving bisimilarity implies weak completed trace equivalence. Therefore, the thesis $L_1 = L_2$ follows by transitivity. For the reverse implication, let L_2 be the language recognized by a finite CCS net system $N_2(m_2)$, labeled over $\mathscr{L} \cup \{\tau\}$. Since a finite CCS net is also a finite P/T net, L_2 is also a finite P/T net language. □

The second result we want to show, by exploiting the representation theorem, is that any finite P/T net, labeled over $\mathscr{L} \cup \{\tau\}$, is weak step simulation equivalent to a suitable finite CCS net, labeled over $\mathscr{L} \cup \{\tau\}$.

Theorem 7.14. *For any finite P/T net system $N(m_0)$, labeled over $\mathscr{L} \cup \{\tau\}$, there exists a finite CCS net system $N'(m_0')$, labeled over $\mathscr{L} \cup \{\tau\}$, such that $N(m_0)$ is weak step simulation equivalent to $N'(m_0')$.* □

The proof is sketched below. Starting from a finite P/T net $N(m_0)$ labeled over $\mathscr{L} \cup \{\tau\}$, we get a well-formed, output-closed FNM process $p = \mathscr{T}_{FNM}(N(m_0))$ such that $Net(p)$ is isomorphic to $N(m_0)$, thanks to Theorem 7.12. We define a *forgetful* encoding $[\![-]\!]$, mapping any output-closed, well-formed FNM process p to the output-closed FNC process $[\![p]\!]$, by simply turning every strong prefix $\underline{x}_i^j$ into a normal prefix x_i^j; the resulting term $[\![p]\!]$ is indeed an output-closed FNC process and, by Corollary 6.3, $Net([\![p]\!])$ is a finite CCS net. The step semantics of the FNC process $[\![p]\!]$ and of its associated net $Net([\![p]\!])$ is the same, up to step bisimilarity, as proved in Theorem 6.9. Similarly, we shall prove in the next section that the original net $N(m_0)$ is step bisimilar to the STS associated with the FNM process p. Therefore, in order to prove that $N(m_0)$ is weak step simulation equivalent to $Net([\![p]\!])$, we can simply work on the STSs for p and $[\![p]\!]$, hence taking advantage of the process term representation of the two nets. Now, in one direction, consider the relation

$R_1^p = \{(q, [\![q]\!]) \mid q \text{ is step reachable from } p\}$.

It is an easy exercise to check that R_1^p is a weak step simulation. Any step transition $q \xrightarrow{M}_s q'$ that q can perform can be mimicked by $[\![q]\!]$ by implementing each multi-party transition of q, labeled a in M, as a sequence of binary handshake communications over the restricted actions, enabling the same action a as its last action; hence, for some q'' $[\![q]\!] \overset{\varepsilon}{\Longrightarrow}_s [\![q'']\!] \xrightarrow{M}_s [\![q']\!]$; this can be proved by induction on the proof of transition $q \xrightarrow{M}_s q'$. As the pair $(p, [\![p]\!])$ belongs to R_1^p, we have that p is weakly step simulated by $[\![p]\!]$. In the other direction, consider the relation

$R_2^p = \{(r, q) \mid q \text{ is step reachable from } p \text{ and } r \text{ is step reachable from } [\![q]\!]$
by performing only synchronizations on restricted actions$\}$.

Also in this case, it is not difficult to check that R_2^p is a weak step simulation. If r performs a step $M = M' \cup M''$, where M'' contains all and only the synchronizations on restricted actions, and in doing so it reaches r', then q responds by performing the step M', reaching q', so that r' is step reachable from $[\![q']\!]$ by performing the synchronizations specified in M''; hence, the pair (r', q') belongs to R_2^p, as required. This fact can be proved by induction on the proof of transition $r \xrightarrow{M}_s r'$. As the pair $([\![p]\!], p)$ belongs to R_2^p, we can conclude that $[\![p]\!]$ is weak step simulated by p. Summing up, p and $[\![p]\!]$ are weak step simulation equivalent, and so are the finite P/T net $N(m_0)$ and the finite CCS net $Net([\![p]\!])$.

The theorem above proves that multi-party communication (generating a label in $\mathscr{L} \cup \{\tau\}$) can be implemented, up to weak step similarity, by binary handshake communication. However, this implementation is not very satisfactory because weak (step) similarity does not preserve deadlocks, hence not even the weak completed traces.[4] For instance, consider the net $N(m_0) = (S, A, T, m_0)$, where $S = \{s_1, s_2, s_3\}$, $A = \{a, b\}$, $m_0 = s_1 \oplus s_2 \oplus 2 \cdot s_3$ and $T = \{t_1, t_2\}$, with $t_1 = (s_1 \oplus 2 \cdot s_3, a, \emptyset)$ and $t_2 = (s_2 \oplus 2 \cdot s_3, b, \emptyset)\}$. The associated FNM process p is $(\nu L)(C_1 \,|\, C_2 \,|\, C_3 \,|\, C_3)$ where $L = \{x_1^1, x_2^1, x_3^1, x_1^2, x_2^2, x_3^2\}$, and

$$\begin{aligned} C_1 &\doteq \underline{x}_3^1.\underline{x}_3^1.a.\mathbf{0} + \mathbf{0}, \\ C_2 &\doteq \mathbf{0} + \underline{x}_3^2.\underline{x}_3^2.b.\mathbf{0}, \\ C_3 &\doteq \overline{x}_3^1.\mathbf{0} + \overline{x}_3^2.\mathbf{0}, \end{aligned}$$

and its encoded FNC process is $[\![p]\!] = (\nu L)(C_1' \,|\, C_2' \,|\, C_3' \,|\, C_3')$, where

$$\begin{aligned} C_1' &\doteq x_3^1.x_3^1.a.\mathbf{0} + \mathbf{0}, \\ C_2' &\doteq \mathbf{0} + x_3^2.x_3^2.b.\mathbf{0}, \\ C_3' &\doteq \overline{x}_3^1.\mathbf{0} + \overline{x}_3^2.\mathbf{0}. \end{aligned}$$

Now, $[\![p]\!]$ can perform the silent step $\{\tau, \tau\}$, and, in doing so, it reaches the deadlock state $(\nu L)(x_3^1.a.\mathbf{0} \,|\, x_3^2.b.\mathbf{0} \,|\, \mathbf{0} \,|\, \mathbf{0})$: the two possible multi-party transactions have been started but neither has been completed, because no atomicity mechanism is available in FNC; on the contrary, p cannot silently reach a deadlock state.

7.6 Soundness

In this section, we want to prove that the operational net semantics preserves the step semantics of Section 7.3. More precisely, we want to show that any FNM process p, in the STS semantics, is step bisimilar to $dec(p)$ in the step marking graph $SMG(Net(p))$.

Lemma 7.12. *For any $p \in \mathscr{P}_{FNM}$, any $G \in \mathscr{M}_{fin}(T_{FNM})$ and any $M \in \mathscr{B}$, if $dec(p)[G\rangle m$ and $MSync(l(G), M)$, then there exists G' such that $dec(p)[G'\rangle m$ and $l(G') = M$.*

Proof. By induction on the size of G. If $|G| = 1$, then only $MSync(l(G), l(G))$ holds and the thesis follows trivially by choosing $G' = G$. If $|G| = n > 1$, then two subcases are in order: either there are no synchronizable $\sigma_1, \sigma_2 \in l(G)$, or there exists σ such that $Sync(\sigma_1, \sigma_2, \sigma)$ holds for some $\{\sigma_1, \sigma_2\} \subseteq l(G)$. In the former case, only $MSync(l(G), l(G))$ holds and the thesis follows trivially. In the latter case, besides the trivial case $MSync(l(G), l(G))$, we have to investigate also the following: take two transitions t_1 and t_2 in G such that $l(t_i) = \sigma_i$, for $i = 1, 2$. Then, take the step $\overline{G} = (G \setminus \{t_1, t_2\}) \cup \{t_1 \,|\, t_2\}$, where transition $t_1 \,|\, t_2 = ({}^\bullet t_1 \oplus {}^\bullet t_2, \sigma, t_1^\bullet \oplus t_2^\bullet)$ is derived by rule (s-com) with t_1 and t_2 as premises.[5] Clearly, also $dec(p)[\overline{G}\rangle m$

[4] In Table 7.8 of Section 7.2.1, we showed a different encoding from well-formed, output-closed FNM processes to output-closed FNC processes, where possibly concurrent actions are strictly sequentialized, but which preserves the weak completed traces.

[5] Of course, $ad({}^\bullet t_1 \oplus {}^\bullet t_2)$ holds because ${}^\bullet t_1 \oplus {}^\bullet t_2 \subseteq dec(p)$, which is admissible.

and $|\overline{G}| < n$, so that induction can be applied to conclude that for any M such that $MSync(l(\overline{G}),M)$, there exists G' such that $dec(p)[G'\rangle m$ and $l(G') = M$. Note that also $MSync(l(G),M)$ holds, because $MSync(l(\overline{G}),M)$, so that the thesis follows trivially for the chosen $\overline{G}$. Since this argument can be repeated for any pair of transitions t_1 and t_2 in G such that $l(t_1) = \sigma_1$ and $l(t_2) = \sigma_2$, for some synchronizable $\sigma_1, \sigma_2 \in l(G)$ (hence, for any $\overline{G}$ of the form above), we can conclude that the thesis holds: if $dec(p)[G\rangle m$ and $MSync(l(G),M)$, then there exists G' such that $dec(p)[G'\rangle m$ and $l(G') = M$. □

Proposition 7.34. *For any $p \in \mathscr{P}_{FNM}$, if $p \xrightarrow{M}_s p'$, then there exists a step G, with $l(G) = M$, such that $dec(p)[G\rangle dec(p')$.*

Proof. By induction on the proof of $p \xrightarrow{M}_s p'$. The proof is very similar to the analogous Proposition 6.24, so we describe only a few cases.

If the last rule used in deriving $p \xrightarrow{M}_s p'$ is (S-Pref^s), then $M = \{a \diamond \sigma\}$ for some suitable a and σ, p must be $\underline{a}.q$ and the premise transition is $q \xrightarrow{\{\sigma\}}_s p'$. By induction, we know that $dec(q)[\{t'\}\rangle dec(p')$ with $l(t') = \sigma$. Since q is sequential, this step is possible only if $t' = \{q\} \xrightarrow{\sigma} dec(p')$ is derivable by the rules in Table 7.13; hence, by rule (s-pref), also the net transition $t = \{\underline{a}.q\} \xrightarrow{a \diamond \sigma} dec(p')$ is derivable, and so the singleton step $dec(\underline{a}.q)[\{t\}\rangle dec(p')$ is also derivable, as required.

If the last rule used in deriving $p \xrightarrow{M}_s p'$ is (S-Com^s), then $p = p_1 \,|\, p_2$, $p' = p_1' \,|\, p_2'$ and the two premises are $p_1 \xrightarrow{M_1}_s p_1'$, $p_2 \xrightarrow{M_2}_s p_2'$, with $MSync(M_1 \oplus M_2, M)$. By induction, we have $dec(p_1)[G_1\rangle dec(p_1')$, with $l(G_1) = M_1$, and $dec(p_2)[G_2\rangle dec(p_2')$, with $l(G_2) = M_2$; therefore, $dec(p_1 \,|\, p_2) = dec(p_1) \oplus dec(p_2)[G_1 \oplus G_2\rangle dec(p_1') \oplus dec(p_2') = dec(p_1' \,|\, p_2')$. By Lemma 7.12, since $MSync(l(G_1 \oplus G_2),M)$, there exists a step G' such that $dec(p_1 \,|\, p_2)[G'\rangle dec(p_1' \,|\, p_2')$ and $l(G') = M$, as required. □

Proposition 7.35. *For any $p \in \mathscr{P}_{FNM}$ and any $G \in \mathscr{M}_{fin}(T_{FNM})$, if $dec(p)[G\rangle m$, then there exists $p' \in \mathscr{P}_{FNM}$ such that $p \xrightarrow{M}_s p'$, with $l(G) = M$ and $dec(p') = m$.*

Proof. By induction on the definition of $dec(p)$. The proof is very similar to the analogous Proposition 6.25, so we describe only a few cases.

If $p = \underline{a}.q$, then $dec(p) = \{\underline{a}.q\}$. If $\{\underline{a}.q\}[G\rangle m$, then G is a singleton $\{t\}$, with $t = (\{\underline{a}.q\}, \sigma, dec(p'))$ for some suitable σ and p', because, by Proposition 7.24, the post-set of any derivable net transition is admissible and, by Theorem 7.6, any admissible marking is complete. The proof of the net transition t can be mimicked by the corresponding rules in Table 7.10 to produce the transition $p \xrightarrow{\{\sigma\}}_s p'$, as required.

If $p = p_1 \,|\, p_2$, then $dec(p) = dec(p_1) \oplus dec(p_2)$. The following three sub-cases are possible:

(i) $dec(p_1)[G\rangle m_1$ and $m = m_1 \oplus dec(p_2)$: in such a case, by induction, we know that there exists p_1' such that $p_1 \xrightarrow{M}_s p_1'$, with $l(G) = M$ and $dec(p_1') = m_1$; therefore, by rule (Par$_1^s$), also $p_1 \,|\, p_2 \xrightarrow{M}_s p_1' \,|\, p_2$ is derivable, with $dec(p_1' \,|\, p_2) = m$, as required.

$$\frac{}{GSync(G,G)} \qquad \frac{GSync(G\oplus\{t\},G') \quad t=({}^\bullet t_1 \oplus {}^\bullet t_2, \sigma, t_1^\bullet \oplus t_2^\bullet) \quad l(t_i)=\sigma_i \quad Sync(\sigma_1,\sigma_2,\sigma)}{GSync(G\oplus\{t_1,t_2\},G')}$$

Table 7.16 Step transition synchronization relation

(ii) $dec(p_2)[G\rangle m_2$ and $m = dec(p_1)\oplus m_2$: this is the symmetric case of the above, hence omitted.

(iii) there exist two steps G_1 and G_2 such that $dec(p_1)[G_1\rangle m_1$, $dec(p_2)[G_2\rangle m_2$, with $l(G_1)=M_1$, $l(G_2)=M_2$, $m=m_1\oplus m_2$ and $GSync(G_1\oplus G_2,G)$, where relation $GSync(-)$ of Table 7.16 mimics, over multisets of transitions, what relation $MSync(-)$ does over multisets of labels, so that $MSync(M_1\oplus M_2,M)$ holds, too. In this case, by induction, we know that there exist p_1' and p_2' such that $p_1 \xrightarrow{M_1}_s p_1'$, $p_2 \xrightarrow{M_2}_s p_2'$, with $dec(p_1')=m_1$ and $dec(p_2')=m_2$. By rule (S-Com^s) of Table 7.10, also $p_1 \,|\, p_2 \xrightarrow{M}_s p_1' \,|\, p_2'$ is derivable, with $dec(p_1' \,|\, p_2')=m$, as required. □

Theorem 7.15. *For any $p\in\mathscr{P}_{FNM}$, $p\sim_{step} dec(p)$.*

Proof. The relation $R=\{(p,dec(p)) \;\big|\; p\in\mathscr{P}_{FNM}\}$ is a bisimulation between the step transition system $\mathscr{C}_p^{step}=(\mathscr{P}_p^{step}, sort(p)_s, \rightarrow_s^p, p)$, originating from the step semantics of Section 7.3, and the step marking graph $SMG(Net(p))$. Given $(p,dec(p))\in R$, if $p\xrightarrow{M}_s p'$, then by Proposition 7.34 we have $dec(p)[G\rangle dec(p')$, with $l(G)=M$ and $(p',dec(p'))\in R$; conversely, if $dec(p)[G\rangle m$, with $l(G)=M$, then, by Proposition 7.35, there exists $p'\in\mathscr{P}_{FNM}$ such that $p\xrightarrow{M}_s p'$ and $dec(p')=m$, so that $(p',dec(p'))\in R$, as required. □

7.7 Denotational Net Semantics

The extension to FNM of the denotational semantics we have defined for FNC in the previous chapters is not too complex; we have only to cope with multi-party synchronization, so that the binary operator of synchronous product $-\otimes-$ (see Definition 6.13) is to be replaced by the unary operator of *synchronous closure* $-^\otimes$ (Definition 7.5 below). As for FNC, the correspondence with the operational semantics is strictly preserved for restriction-free processes: the operational net $Net(t)$ is exactly the denotational net $[\![t]\!]_\emptyset$ we are going to define. On the contrary, for a restricted process $p=(\nu L)t$, we have the weaker result that $Net(p)$ is exactly the *statically reachable subnet* of $[\![p]\!]_\emptyset$.

Table 7.17 shows the denotational semantics for the empty process $\mathbf{0}$, for the action prefixing operator, for the strong prefixing operator and for the constants. As usual, the semantics definition $[\![-]\!]_I$ is parametrized by a set I of constants that have been already found while scanning p; such a set is initially empty and it is used to avoid looping on recursive constants. The definition is syntax driven and also the places of the constructed net are syntactic objects, i.e., (extended) sequential FNM processes. The definition for $\mathbf{0}$ is obvious: the associated net is empty. About the semantics of the action prefixing operator, the only non-trivial aspect is that the newly added transition $t = (\{\mu.p\}, \mu, dec(p))$ may be able to synchronize with the transitions that are in the set T', because we have to consider also all the possible synchronizations that are statically enabled at the set $[\![dom(\{\mu.p\})\rangle$ of the places statically reachable from $\{\mu.p\}$; moreover, this new transition can trigger other synchronizations involving more transitions in T', as Example 7.12 will show. The set of all such transitions is denoted by $(\{t\} \cup T')^{\otimes}$, where the auxiliary operation $-^{\otimes}$ of *synchronous closure* of a finite set of transitions is defined as follows.

Definition 7.5. (Synchronous closure) Given a finite set $T \subseteq T_{FNM}$ of transitions, we define its *synchronous closure* $T^{\otimes}$ as the least set such that

$$\begin{aligned} T^{\otimes} = T \cup \{({}^{\bullet}t_1 \oplus {}^{\bullet}t_2, \sigma, t_1^{\bullet} \oplus t_2^{\bullet}) \;\big|\; \\ t_1, t_2 \in T^{\otimes}, l(t_1) = \sigma_1, l(t_2) = \sigma_2, Sync(\sigma_1, \sigma_2, \sigma)\}. \end{aligned} \qquad \square$$

The set $T^{\otimes}$ can be computed inductively by adding, step by step, new transitions to the base, finite set T, as the following chain of sets shows:

$$\begin{aligned} T_0^{\otimes} &= T, \\ T_{i+1}^{\otimes} &= T_i^{\otimes} \cup \{({}^{\bullet}t_1 \oplus {}^{\bullet}t_2, \sigma, t_1^{\bullet} \oplus t_2^{\bullet}) \;\big|\; \\ &\qquad t_1, t_2 \in T_i^{\otimes}, l(t_1) = \sigma_1, l(t_2) = \sigma_2, Sync(\sigma_1, \sigma_2, \sigma)\} \qquad \text{for } i > 0. \end{aligned}$$

Of course, $T_i^{\otimes} \subseteq T_{i+1}^{\otimes}$ for any $i \geq 0$; the computation will eventually end if there exists an index j such that $T_j^{\otimes} = T_{j+1}^{\otimes}$. The following proposition states that this is the case when the involved places constitute a well-behaved set.

Proposition 7.36. *Given a finite set $T \subseteq T_{FNM}$, such that $S = \bigcup_{t \in T} dom({}^{\bullet}t)$ is well behaved, then its synchronous closure $T^{\otimes}$ is finite.*

Proof. *As for the proof of Theorem 7.9, the key observation is that for the well-behaved set of places S, the longest label σ in T defines the largest number of possible binary synchronizations forming one multi-party synchronization, as with each synchronization the resulting synchronized sequence is shortened. If $|\sigma| = k$, then $T_k^{\otimes} = T_{k+1}^{\otimes}$.* □

Example 7.11. **(What can go wrong for a non-well-behaved set?)** Continuing Example 7.8, let us consider the set $T = \{(\{a.\mathbf{0}\}, a, \emptyset), (\{\underline{a}.\overline{a}.\mathbf{0}\}, a\overline{a}, \emptyset), (\{\overline{a}.\mathbf{0}\}, \overline{a}, \emptyset)\}$. Of course, $S = \bigcup_{t \in T} dom({}^{\bullet}t)$ is *not* well behaved, because of the place $\underline{a}.\overline{a}.\mathbf{0}$, which is not well formed. The synchronous closure $T^{\otimes}$ is an infinite set; in fact, $T_k^{\otimes} = T \cup \{(n \cdot \underline{a}.\overline{a}.\mathbf{0} \oplus \overline{a}.\mathbf{0}, \overline{a}, \emptyset) \;\big|\; 1 \leq n \leq k\} \cup \{(n \cdot \underline{a}.\overline{a}.\mathbf{0} \oplus \overline{a}.\mathbf{0} \oplus a.\mathbf{0}, \tau, \emptyset) \;\big|\; 1 \leq n \leq k-1\}$, so that $T_k^{\otimes} \subset T_{k+1}^{\otimes}$ for any $k \geq 0$. □

The denotational definition for the strong prefixing operator is a bit involved: besides (i) taking the place of p (if $p \neq \mathbf{0}$) — and the transitions having p in their preset — in the resulting net only in case there is some transition t producing at least one

$$
\begin{array}{rcl}
[\![\mathbf{0}]\!]_I &=& (\emptyset,\emptyset,\emptyset,\emptyset)\\[1ex]
[\![\mu.p]\!]_I &=& (S,A,T,\{\mu.p\}) \ \text{ given } [\![p]\!]_I = (S',A',T',dec(p)) \ \text{ and where}\\
&& S = \{\mu.p\} \cup S',\ t = (\{\mu.p\},\mu,dec(p))\\
&& T = (\{t\} \cup T')^{\otimes} \quad A = l(T)\\[1ex]
[\![\underline{a}.p]\!]_I &=& (S,A,T,\{\underline{a}.p\}) \ \text{ given } [\![p]\!]_I = (S',A',T',dec(p)) \ \text{ and where}\\
&& S = \{\underline{a}.p\} \cup S'',\ \text{where}\\
&& S'' = \begin{cases} S' & \exists t \in T' . t^{\bullet}(p) > 0 \ \vee \ p = \mathbf{0},\\ S' \setminus \{p\} & \text{otherwise} \end{cases}\\
&& A = l(T),\ T = (T_1 \cup T'')^{\otimes} \ \text{where}\\
&& T_1 = \{(\{\underline{a}.p\}, a \diamond \sigma, m) \mid (\{p\},\sigma,m) \in T'\}\\
&& T'' = \begin{cases} T' & \exists t \in T' . t^{\bullet}(p) > 0 \ \vee \ p = \mathbf{0},\\ T' \setminus \{t \in T' \mid {}^{\bullet}t(p) > 0\} & \text{otherwise} \end{cases}\\[1ex]
[\![C]\!]_I &=& (\{C\},\emptyset,\emptyset,\{C\}) \ \text{ if } C \in I\\[1ex]
[\![C]\!]_I &=& (S,A,T,\{C\}) \quad \text{ if } C \notin I, \text{ given } C \doteq p \text{ and } [\![p]\!]_{I\cup\{C\}} = (S',A',T',dec(p))\\
&& S = \{C\} \cup S'',\ \text{where}\\
&& S'' = \begin{cases} S' & \exists t \in T' . t^{\bullet}(p) > 0 \ \vee \ p = \mathbf{0},\\ S' \setminus \{p\} & \text{otherwise} \end{cases}\\
&& A = A',\ T = T'' \cup T_1 \ \text{where}\\
&& T'' = \begin{cases} T' & \exists t \in T' . t^{\bullet}(p) > 0 \ \vee \ p = \mathbf{0},\\ T' \setminus \{t \in T' \mid {}^{\bullet}t(p) > 0\} & \text{otherwise} \end{cases}\\
&& T_C = \{(n \cdot C \oplus m_1, \sigma, m_2) \mid (n \cdot p \oplus m_1, \sigma, m_2) \in T', n \geq 1\}\\
&& T_1 = \begin{cases} \{(n \cdot C \oplus m_1, \sigma, m_2) \in T_C \mid m_1(p) = 0\} & \text{if } p \notin S,\\ T_C & \text{otherwise} \end{cases}
\end{array}
$$

Table 7.17 Denotational net semantics — part I

token in the place p (definition of the sets S'' and T''), (ii) we have to consider the new transitions with pre-set $\{\underline{a}.p\}$ in correspondence with the analogous transitions with pre-set $\{p\}$ (definition of the set T_1); and finally, (iii) we have to take the synchronous closure of the set $T_1 \cup T''$ (see Example 7.13).

The denotational definition for the constant C, with $C \doteq p$ and $C \notin I$, is the same as for FNC: the only difference is that, in the definition of the set T_1, now the parameter n can be an arbitrary number (for FNC, $1 \leq n \leq 2$, because synchronization is strictly binary) and $|m_1|$ can be larger than one.

Table 7.18 shows the denotational semantics for the parallel operator and for the choice operator. The case of parallel composition is easy: the net for $p_1 \mid p_2$ is

$$[\![p_1 \,|\, p_2]\!]_I = (S, A, T, m_0) \text{ given } [\![p_i]\!]_I = (S_i, A_i, T_i, m_i) \text{ for } i = 1,2, \text{ and where}$$

$$S = S_1 \cup S_2,\ m_0 = m_1 \oplus m_2,\ T = (T_1 \cup T_2)^{\otimes},\ A = l(T)$$

$$[\![p_1 + p_2]\!]_I = (S, A, T, \{p_1 + p_2\}) \text{ given } [\![p_i]\!]_I = (S_i, A_i, T_i, dec(p_i)) \text{ for } i = 1,2, \text{ and where}$$

$$S = \{p_1 + p_2\} \cup S_1' \cup S_2', \text{ with, for } i = 1,2,$$

$$S_i' = \begin{cases} S_i & \exists t \in T_i \text{ such that } t^{\bullet}(p_i) > 0 \lor p_i = \mathbf{0}, \\ S_i \setminus \{p_i\} & \text{otherwise} \end{cases}$$

$$A = l(T),\ T = (T_1' \cup T_1'' \cup T_2' \cup T_2'')^{\otimes} \text{ with, for } i = 1,2,$$

$$T_i' = \begin{cases} T_i & \exists t \in T_i \,.\, t^{\bullet}(p_i) > 0 \lor p_i = \mathbf{0}, \\ T_i \setminus \{t \in T_i \mid {}^{\bullet}t(p_i) > 0\} & \text{otherwise} \end{cases}$$

$$T_i^{+} = \{(n \cdot (p_1 + p_2) \oplus m_1, \sigma, m_2) \mid (n \cdot p_i \oplus m_1, \sigma, m_2) \in T_i, n \geq 1\}$$

$$T_i'' = \begin{cases} \{(n \cdot (p_1 + p_2) \oplus m_1, \sigma, m_2) \in T_i^{+} \mid m_1(p_i) = 0\} & \text{if } p_i \notin S \\ T_i^{+} & \text{otherwise} \end{cases}$$

Table 7.18 Denotational net semantics — part II

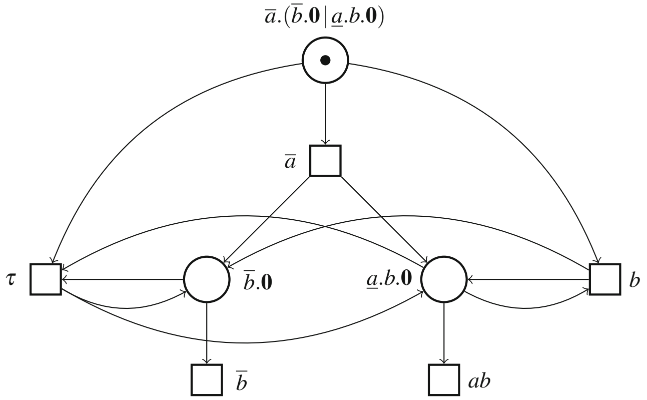

$\overline{a}.(\overline{b}.\mathbf{0} \,|\, \underline{a}.b.\mathbf{0})$
$\overline{a}$
τ
$\overline{b}.\mathbf{0}$
$\underline{a}.b.\mathbf{0}$
b
$\overline{b}$
ab

Fig. 7.3 The P/T net for $\overline{a}.(\overline{b}.\mathbf{0} \,|\, \underline{a}.b.\mathbf{0})$

obtained by putting together the two nets of p_1 and p_2, by taking the multiset union of the respective initial markings, and by making the synchronous closure of $T_1 \cup T_2$.

The denotational semantics of $p_1 + p_2$ is also very similar to that for FNC; the only difference is that the synchronous product $(T_1' \cup T_1'') \otimes (T_2' \cup T_2'')$ is turned into the synchronous closure $(T_1' \cup T_1'' \cup T_2' \cup T_2'')^{\otimes}$, since the transitions resulting from a synchronization may be used to synchronize further some transitions from p_1 or from p_2 (see Example 7.13).

A couple of examples may be useful to clarify the definitions of the various operators in the denotational semantics.

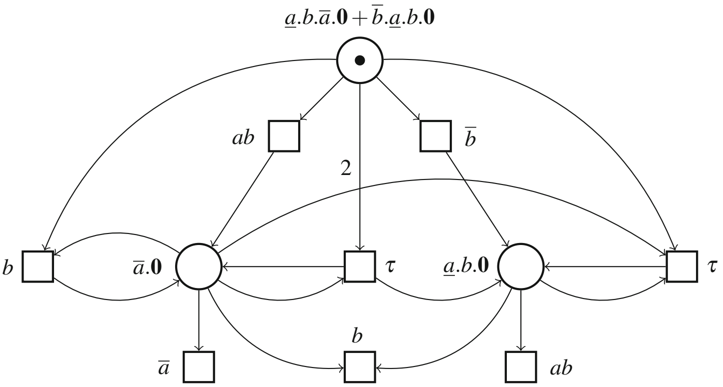

$\underline{a}.b.\overline{a}.\mathbf{0}+\overline{b}.\underline{a}.b.\mathbf{0}$
ab
2
$\overline{b}$
b
$\overline{a}.\mathbf{0}$
τ
$\underline{a}.b.\mathbf{0}$
τ
b
$\overline{a}$
ab

Fig. 7.4 The P/T net for $\underline{a}.b.\overline{a}.\mathbf{0}+\overline{b}.\underline{a}.b.\mathbf{0}$

Example 7.12. Let us consider the sequential FNM process $\overline{a}.(\overline{b}.\mathbf{0}\,|\,\underline{a}.b.\mathbf{0})$. Since $[\![\mathbf{0}]\!]_I = (\emptyset,\emptyset,\emptyset,\emptyset)$, the net for $b.\mathbf{0}$ is

$[\![b.\mathbf{0}]\!]_\emptyset = (\{b.\mathbf{0}\},\{b\},\{(\{b.\mathbf{0}\},b,\emptyset)\},\{b.\mathbf{0}\})$, and so the net for $\underline{a}.b.\mathbf{0}$ is
$[\![\underline{a}.b.\mathbf{0}]\!]_\emptyset = (\{\underline{a}.b.\mathbf{0}\},\{ab\},\{(\{\underline{a}.b.\mathbf{0}\},ab,\emptyset)\},\{\underline{a}.b.\mathbf{0}\})$.

The net for $\overline{b}.\mathbf{0}$ is

$[\![\overline{b}.\mathbf{0}]\!]_\emptyset = (\{\overline{b}.\mathbf{0}\},\{\overline{b}\},\{(\{\overline{b}.\mathbf{0}\},\overline{b},\emptyset)\},\{\overline{b}.\mathbf{0}\})$, and so the net for $p = \overline{b}.\mathbf{0}\,|\,\underline{a}.b.\mathbf{0}$ is
$[\![p]\!]_\emptyset = (\{\overline{b}.\mathbf{0},\underline{a}.b.\mathbf{0}\},\{\overline{b},ab\},\{(\{b.\mathbf{0}\},b,\emptyset),(\{\underline{a}.b.\mathbf{0}\},ab,\emptyset)\},\{\overline{b}.\mathbf{0},\underline{a}.b.\mathbf{0}\})$,

where no synchronization is possible between the two transitions of the constituent nets. Now the net for $\overline{a}.(\overline{b}.\mathbf{0}\,|\,\underline{a}.b.\mathbf{0})$ is $(S,A,T,\{\overline{a}.(\overline{b}.\mathbf{0}\,|\,\underline{a}.b.\mathbf{0})\})$, where

$$\begin{aligned}
S &= \{\overline{a}.(\overline{b}.\mathbf{0}\,|\,\underline{a}.b.\mathbf{0}),\overline{b}.\mathbf{0},\underline{a}.b.\mathbf{0}\},\\
A &= \{\overline{a},ab,b,\overline{b},\tau\},\\
T &= (\{t\}\cup T')^{\otimes} = \{t\}\cup T'\cup T'',\\
t &= (\{\overline{a}.(\overline{b}.\mathbf{0}\,|\,\underline{a}.b.\mathbf{0})\},\overline{a},\{\overline{b}.\mathbf{0},\underline{a}.b.\mathbf{0}\}),\\
T' &= \{(\{\overline{b}.\mathbf{0}\},\overline{b},\emptyset),(\{\underline{a}.b.\mathbf{0}\},ab,\emptyset)\},\\
T'' &= \{(\{\overline{a}.(\overline{b}.\mathbf{0}\,|\,\underline{a}.b.\mathbf{0}),\underline{a}.b.\mathbf{0}\},b,\{\overline{b}.\mathbf{0},\underline{a}.b.\mathbf{0}\}),\\
&\quad\;(\{\overline{a}.(\overline{b}.\mathbf{0}\,|\,\underline{a}.b.\mathbf{0}),\underline{a}.b.\mathbf{0},\overline{b}.\mathbf{0}\},\tau,\{\overline{b}.\mathbf{0},\underline{a}.b.\mathbf{0}\})\},
\end{aligned}$$

which is depicted in Figure 7.3. Note that the synchronized transitions in T'' are not executable from the initial marking $m_0 = \{\overline{a}.(\overline{b}.\mathbf{0}\,|\,\underline{a}.b.\mathbf{0})\}$; however, they are statically enabled at $[\![dom(m_0)\rangle = S$ and even dynamically enabled at the marking $\{\overline{a}.(\overline{b}.\mathbf{0}\,|\,\underline{a}.b.\mathbf{0}),\underline{a}.b.\mathbf{0},\ \overline{b}.\mathbf{0}\}$, which is reachable from $2\cdot m_0$. □

Example 7.13. Consider the FNM sequential process $p = \underline{a}.b.\overline{a}.\mathbf{0}+\overline{b}.\underline{a}.b.\mathbf{0}$. The net for $\overline{a}.\mathbf{0}$ is $[\![\overline{a}.\mathbf{0}]\!]_\emptyset = (\{\overline{a}.\mathbf{0}\},\{\overline{a}\},\{(\{\overline{a}.\mathbf{0}\},\overline{a},\emptyset)\},\{\overline{a}.\mathbf{0}\})$, and so the net for $b.\overline{a}.\mathbf{0}$ is

$$[\![b.\overline{a}.\mathbf{0}]\!]_\emptyset = (\{b.\overline{a}.\mathbf{0}, \overline{a}.\mathbf{0}\}, \{b, \overline{a}\}, \{(\{b.\overline{a}.\mathbf{0}\}, b, \{\overline{a}.\mathbf{0}\}), (\{\overline{a}.\mathbf{0}\}, \overline{a}, \emptyset)\}, \{b.\overline{a}.\mathbf{0}\}),$$

and so, the net for $\underline{a}.b.\overline{a}.\mathbf{0}$ is $[\![\underline{a}.b.\overline{a}.\mathbf{0}]\!]_\emptyset = (S, A, T, \{\underline{a}.b.\overline{a}.\mathbf{0}\})$, where

$$\begin{aligned}
S &= \{\underline{a}.b.\overline{a}.\mathbf{0}, \overline{a}.\mathbf{0}\},\\
A &= \{\overline{a}, ab, b\},\\
T &= (T_1 \cup T'')^{\otimes} = T_1 \cup T'' \cup T''',\\
T_1 &= \{(\{\underline{a}.b.\overline{a}.\mathbf{0}\}, ab, \{\overline{a}.\mathbf{0}\})\},\\
T'' &= \{(\{\overline{a}.\mathbf{0}\}, \overline{a}, \emptyset)\},\\
T''' &= \{(\{\underline{a}.b.\overline{a}.\mathbf{0}, \overline{a}.\mathbf{0}\}, b, \{\overline{a}.\mathbf{0}\})\}.
\end{aligned}$$

The net for $\underline{a}.b.\mathbf{0}$ is $[\![\underline{a}.b.\mathbf{0}]\!]_\emptyset = (\{\underline{a}.b.\mathbf{0}\}, \{ab\}, \{(\{\underline{a}.b.\mathbf{0}\}, ab, \emptyset)\}, \{\underline{a}.b.\mathbf{0}\})$, and so that for $\overline{b}.\underline{a}.b.\mathbf{0}$ is

$$\begin{aligned}
[\![\overline{b}.\underline{a}.b.\mathbf{0}]\!]_\emptyset = (&\{\overline{b}.\underline{a}.b.\mathbf{0}, \underline{a}.b.\mathbf{0}\}, \{\overline{b}, ab\}, \{(\{\overline{b}.\underline{a}.b.\mathbf{0}\}, \overline{b}, \{\underline{a}.b.\mathbf{0}\}), (\{\underline{a}.b.\mathbf{0}\}, ab, \emptyset)\},\\
&\{\overline{b}.\underline{a}.b.\mathbf{0}\}).
\end{aligned}$$

Finally, the net for $p = \underline{a}.b.\overline{a}.\mathbf{0} + \overline{b}.\underline{a}.b.\mathbf{0}$ is $[\![p]\!]_\emptyset = (S, A, T, \{p\})$, where

$$\begin{aligned}
S &= \{p, \overline{a}.\mathbf{0}, \underline{a}.b.\mathbf{0}\},\\
A &= \{\overline{a}, ab, b, \overline{b}, \tau\},\\
T &= (T_1' \cup T_1'' \cup T_2' \cup T_2'')^{\otimes} = T_1' \cup T_1'' \cup T_2' \cup T_2'' \cup T''',\\
T_1' &= \{(\{\overline{a}.\mathbf{0}\}, \overline{a}, \emptyset)\},\\
T_1'' &= \{(\{p\}, ab, \{\overline{a}.\mathbf{0}\}), (\{p, \overline{a}.\mathbf{0}\}, b, \{\overline{a}.\mathbf{0}\})\},\\
T_2' &= \{(\{\underline{a}.b.\mathbf{0}\}, ab, \emptyset)\},\\
T_2'' &= \{(\{p\}, \overline{b}, \{\underline{a}.b.\mathbf{0}\})\},\\
T''' &= \{(\{p, p, \overline{a}.\mathbf{0}\}, \tau, \{\overline{a}.\mathbf{0}, \underline{a}.b.\mathbf{0}\}), (\{\overline{a}.\mathbf{0}, \underline{a}.b.\mathbf{0}\}, b, \emptyset),\\
&\quad\ (\{p, \overline{a}.\mathbf{0}, \underline{a}.b.\mathbf{0}\}, \tau, \{\underline{a}.b.\mathbf{0}\})\},
\end{aligned}$$

which is depicted in Figure 7.4. Note that the first transition (and the third one) in T''' is enabled at a marking reachable from $3 \cdot p$. □

In order to prove that, for any restriction-free FNM process p, the operational net $Net(p)$ is exactly the same as the denotational net $[\![p]\!]_\emptyset$, we need five auxiliary lemmata, dealing with $Net(p, I)$, i.e., the operational net associated with p, assuming that the constants in I are undefined.

Lemma 7.13. *For any restriction-free process p, let $Net(p, I) = (S', A', T', dec(p))$. It follows that $Net(\mu.p, I) = (S, A, T, \{\mu.p\})$, where S, A and T are defined as in Table 7.17.*

Proof. By axiom (pref), the only transition enabled at the initial marking $\{\mu.p\}$ is $t = (\{\mu.p\}, \mu, dec(p))$. Therefore, all the places in $dom(dec(p))$ are (statically) reachable from the initial marking $\{\mu.p\}$ and so all the places in S' are statically reachable as well, as $Net(p, I)$ is statically reduced. Therefore, $S = \{\mu.p\} \cup S'$. Now, since $Net(\mu.p, I)$ is statically reduced, T is the set of all the transitions that are statically enabled at S. Set T' contains all the transitions statically enabled at S', so the transitions that are statically enabled at S are exactly the set $(\{t\} \cup T')^{\otimes}$, as required. □

Lemma 7.14. *For any sequential process* p*, let* $Net(p,I) = (S', A', T', dec(p))$*. It follows that* $Net(\underline{a}.p, I) = (S, A, T, \{\underline{a}.p\})$*, where* S, A *and* T *are defined as in Table 7.17.*

Proof. By rule (s-pref), the only transitions enabled at the initial marking $\{\underline{a}.p\}$ *are those obtained with premise of the form* $(\{p\}, \sigma, m)$*. These transitions constitute the set* T_1*. Therefore, all the places in* $dom(m)$ *are (statically) reachable from the initial marking* $\{\underline{a}.p\}$ *and so all the places in* S' *are statically reachable as well, except possibly the place* p *itself, as* $Net(p,I)$ *is statically reduced (definition of the set* S''*). Therefore,* $S = \{\underline{a}.p\} \cup S''$*. Now, since* $Net(\underline{a}.p, I)$ *is statically reduced,* T *is the set of all the transitions that are statically enabled at* S*. Set* T'' *contains all the transitions statically enabled at* S''*, so all the transitions statically enabled at* S *are exactly the set* $(T_1 \cup T'')^{\otimes}$*, as required.* □

Lemma 7.15. *For any sequential FNM process* p *and constant* C *such that* $C \doteq p$*, let* $Net(p, I \cup \{C\}) = (S', A', T', dec(p))$*. It follows that* $Net(C, I) = (S, A, T, \{C\})$*, where* S, A *and* T *are defined as in Table 7.17.*

Proof. Analogous to the proof of Lemma 6.9 for FNC, and so omitted. □

Lemma 7.16. *For any restriction-free FNM processes* p_1 *and* p_2*, let* $Net(p_i, I) = (S_i, A_i, T_i, m_i)$*, for* $i = 1,2$*. It follows that* $Net(p_1 \,|\, p_2, I) = (S, A, T, m_0)$ *where* S, A, T *and* m_0 *are defined as in Table 7.18.*

Proof. As $dec(p_i) = m_i$ *for* $i = 1,2$*, it follows that* $m_0 = dec(p_1 \,|\, p_2) = dec(p_1) \oplus dec(p_2) = m_1 \oplus m_2$*. By induction on the static reachability relation* $dom(m_1 \oplus m_2) \Longrightarrow^* V$*, we shall prove that* $V \subseteq S_1 \cup S_2$*. This is enough, because all the places in* S_1 *or* S_2 *are statically reachable from* $dom(m_1)$ *or* $dom(m_2)$*, so that* S *must be* $S_1 \cup S_2$*. The base case of the induction is* $dom(m_1 \oplus m_2) \Longrightarrow^* dom(m_1 \oplus m_2)$ *and this case is trivial:* $dom(m_1 \oplus m_2) \subseteq S_1 \cup S_2$ *because* $dom(m_1 \oplus m_2) = dom(m_1) \cup dom(m_2)$ *and* $dom(m_i) \subseteq S_i$ *for* $i = 1,2$*. The inductive case is as follows:* $dom(m_1 \oplus m_2) \Longrightarrow^* V_j \overset{t}{\Longrightarrow} V_{j+1}$*. By induction, we can assume that* $V_j \subseteq S_1 \cup S_2$*, and so* $V_j = V_j^1 \cup V_j^2$*, with* $V_j^i \subseteq S_i$ *for* $i = 1,2$*. Now, by induction on the proof of* t*, we show that* $dom(t^{\bullet}) \subseteq S_1 \cup S_2$*; hence, the set* $V_{j+1} = V_j \cup dom(t^{\bullet})$ *of places statically reachable in one step from* V_j *is a subset of* $S_1 \cup S_2$*, as well. Any transition* $t \in T_{FNM}$ *is such that either* ${}^{\bullet}t$ *is a singleton, and so* ${}^{\bullet}t \subseteq V_j^i$ *(for some* $i = 1,2$*), or, by rule (s-com), there exist two transitions* t_1 *and* t_2 *such that* $t = ({}^{\bullet}t_1 \oplus {}^{\bullet}t_2, \sigma, t_1^{\bullet} \oplus t_2^{\bullet})$ *and* $Sync(l(t_1), l(t_2), \sigma)$*. In the former case,* $dom(t^{\bullet}) \subseteq S_i$ *because* $Net(p_i)$ *is statically reduced. In the latter case, by induction on the proof of the transition* t*, we can assume that* $dom(t_i^{\bullet}) \subseteq S_1 \cup S_2$ *for* $i = 1,2$*, and so* $dom(t_1^{\bullet} \oplus t_2^{\bullet}) \subseteq S_1 \cup S_2$*, as well. As a final observation, we note that if* $S = S_1 \cup S_2$*, necessarily* $T = (T_1 \cup T_2)^{\otimes}$*, where the new transitions not in* $T_1 \cup T_2$ *are derived by means of (possibly repeated applications of) rule (s-com).* □

Lemma 7.17. *For any sequential processes* p_1 *and* p_2*, let* $Net(p_i, I) = (S_i, A_i, T_i, m_i)$*, for* $i = 1,2$*. It follows that* $Net(p_1 + p_2, I) = (S, A, T, \{p_1 + p_2\})$ *where* S, A *and* T *are defined as in Table 7.18.*

Proof. *From the initial marking $\{p_1 + p_2\}$, by rules (sum$_1$) and (sum$_2$), we can derive only the transitions that are derivable from the places $\{p_1\}$ or $\{p_2\}$ (assuming the summands are not the empty process $\mathbf{0}$). These transitions are included in the sets T_1'' and T_2'', respectively. Therefore, all the places reachable from the places p_1 or p_2 are also reachable from $p_1 + p_2$, where the place p_1 (or p_2) is to be taken only if it is reachable in one or more steps from p_1 itself (or p_2 itself) — definition of the sets S_1' and S_2'. Therefore, the set T of statically enabled transitions must include the following: (i) all the transitions statically enabled at S_1' or at S_2', i.e., the sets T_1' and T_2'; moreover, (ii) all the transitions with pre-set $n \cdot (p_1 + p_2) \oplus m_1$, in correspondence with the transitions with pre-set $n \cdot p_i \oplus m_1$ for $i = 1, 2$, i.e., the sets T_1'' and T_2''; finally, (iii) the synchronous closure of all such transitions: the set $(T_1' \cup T_1'' \cup T_2' \cup T_2'')^{\otimes}$. It is possible to prove, by induction on the static reachability relation $\Longrightarrow^*$, that these additional transitions do not extend the set of reachable places, as done in the proof of Lemma 7.16. Hence, both S and T satisfy the definition in Table 7.18, as required.* □

Now we are ready to state the correspondence theorem, stating that for any restriction-free process p, the net computed by the operational semantics is exactly the same net computed by the denotational semantics.

Theorem 7.16. *For any restriction-free FNM process t, $Net(t) = [\![t]\!]_{\emptyset}$.*

Proof. *By induction on the definitions of $[\![t]\!]_I$ and $Net(p,I)$, where the latter is the operational net associated with p, assuming that the constants in I are undefined. Then, the thesis follows for $I = \emptyset$.*

The proof, which exploits the five lemmata above, is very similar to that of the analogous Theorem 6.10, and so omitted. □

The denotational semantics for the restriction operator is described in Table 7.19 and is very similar to that for FNC. The net $[\![(\nu a)p]\!]_I$ is essentially computed in two steps starting from the net $[\![p]\!]_I$: first, the substitution $\{a'/a\}$ is applied elementwise to all the places and transitions of the net for p; then, a *pruning* operation $- \setminus a'$ is performed to remove all the relabeled transitions that are now labeled with sequences containing occurrences of the restricted action a' or $\overline{a'}$. Differently from the previous cases, now the places are *extended* sequential FNM processes.

We are going to prove that the statically reachable subnet of the denotational semantics net is the operational semantics net for any restricted FNM process.

Lemma 7.18. *For any FNM process p, let $Net(p) = (S, A, T, m_0)$. It follows that $Net((\nu a)p) = Net_s((Net(p)\{a'/a\}) \setminus a')$, where $(Net(p)\{a'/a\}) \setminus a' = (S\{a'/a\}, (A\{a'/a\}) \setminus a', (T\{a'/a\}) \setminus a', m_0\{a'/a\})$, and the operations of substitution and pruning are defined in Table 7.19.*

Proof. *By induction on the static reachability relation $\Longrightarrow^*$, we can prove that the set $S_{(\nu a)p}$ of the statically reachable places of $Net((\nu a)p)$ is the subset of $S\{a'/a\}$ of the places statically reachable from $m_0\{a'/a\}$ in $(Net(p)\{a'/a\}) \setminus a'$. And vice versa, the places statically reachable from $m_0\{a'/a\}$ in $(Net(p)\{a'/a\}) \setminus a'$ constitute exactly the set $S_{(\nu a)p}$. The details of the proof are very similar to the analogous Lemma 6.12, and so omitted.* □

$$[\![(\nu a)p]\!]_I = ([\![p]\!]_I\{a'/a\})\setminus a' = (S\{a'/a\},(A\{a'/a\})\setminus a',(T\{a'/a\})\setminus a',m_0\{a'/a\})$$

$$\text{given } [\![p]\!]_I = (S,A,T,m_0) \text{ where}$$

$$\begin{aligned}
S\{a'/a\} &= \{s\{a'/a\} \mid s \in S\}\\
A\{a'/a\} &= \{\sigma\{a'/a\} \mid \sigma \in A\}\\
(A')\setminus a' &= \{\sigma \in A' \mid a' \notin n(\sigma)\}\\
T\{a'/a\} &= \{(m_1\{a'/a\},\sigma\{a'/a\},m_2\{a'/a\}) \mid (m_1,\sigma,m_2) \in T\}\\
(T')\setminus a' &= \{(m_1,\sigma,m_2) \mid (m_1,\sigma,m_2) \in T' \wedge a' \notin n(\sigma)\}
\end{aligned}$$

Table 7.19 Denotational net semantics — part III

Theorem 7.17. *For any FNM process p, $Net(p) = Net_s([\![p]\!]_\emptyset)$, i.e., $Net(p)$ is exactly the statically reachable subnet of $[\![p]\!]_\emptyset$.*

Proof. An FNM process p is either a restriction-free process t, or there exists an action a and an FNM process p' such that $p = (\nu a)p'$. In the former case, $Net(t) = [\![t]\!]_\emptyset$ by Theorem 7.16, so that the thesis follows trivially, because $[\![t]\!]_\emptyset$ is statically reduced: $Net_s([\![t]\!]_\emptyset) = [\![t]\!]_\emptyset$.

In the latter case, we can assume, by induction, that $Net(p') = Net_s([\![p']\!]_\emptyset)$. If $Net(p') = (S,A,T,m_0)$, then $Net((\nu a)p') = Net_s((Net(p')\{a'/a\})\setminus a')$, by Lemma 7.18. By definition in Table 7.19, $[\![(\nu a)p']\!]_\emptyset = ([\![p']\!]_\emptyset\{a'/a\})\setminus a'$. Moreover, it holds trivially that $Net_s(N(m_0)) = Net_s(Net_s(N(m_0)))$ for any net $N(m_0)$. Therefore,

$$\begin{aligned}
Net_s([\![(\nu a)p']\!]_\emptyset) &= Net_s(([\![p']\!]_\emptyset\{a'/a\})\setminus a')\\
&= Net_s((Net_s([\![p']\!]_\emptyset)\{a'/a\})\setminus a')\\
&= Net_s((Net(p')\{a'/a\})\setminus a')\\
&= Net((\nu a)p').
\end{aligned}$$

□

7.8 RMCS

RMCS is the calculus whose terms are generated from actions in $Act = \mathscr{L} \cup \overline{\mathscr{L}} \cup \{\tau\}$ and constants, as described by the following abstract syntax:

$$\begin{array}{llll}
s ::= \mathbf{0} \mid & \mu.q \quad \mid \underline{a}.s \mid s+s & & \\
q ::= s \mid & C & & \textit{sequential terms}\\
t ::= q \mid & t \,|\, t & & \textit{restriction-free terms}\\
p ::= t \mid & (\nu a)p & & \textit{general terms}
\end{array}$$

where, as usual, we assume that $Const(p)$ is finite, p is *well formed*, and any constant C is equipped with a definition in syntactic category s. The only difference with respect to well-formed FNM is that the action prefixing operator $\mu.-$ is applied to processes of category q, instead of category t. RMCS is essentially *regular* Multi-CCS [GV15].

With respect to the LTS semantics, RMCS is slightly more expressive than SFM: for any RMCS process p, the LTS $\mathscr{C}_p = (\mathscr{P}_p, \text{sort(p)}, \rightarrow_p, p)$ reachable from p is finite-state, as for any SFM process, but it is labeled over the richer set $\{\tau\} \cup \overline{\mathscr{L}} \cup \mathscr{L}^+$. This fact can be proved by structural induction.

For the inductive base case, a trivial adaptation of Corollary 4.2 ensures that the LTS reachable from p, $\mathscr{C}_p = (\mathscr{P}_p, sort(p), \rightarrow_p, p)$, is finite-state for any process p in syntactic category q.[6]

Now, by induction, we can assume that the cardinality of the set of states reachable from a restriction-free RMCS process p_i is k_i, and moreover that $sort(p_i)$ is finite, for $i = 1, 2$; by inspecting the SOS rules (Par), (S-Com) and (Cong), it is possible to prove that the cardinality of the set of states reachable from the RMCS process $p_1 \,|\, p_2$ is $O(k_1 \times k_2)$; in fact, for any restriction-free RMCS process p, the set $Cong(p) = \{q \;\big|\; p \equiv q\}$ is finite,[7] and so, by rule (Cong), only finitely many different states are reachable for each of the $k_1 \times k_2$ states that are reachable from $p_1 \,|\, p_2$ without using rule (Cong). Moreover, it is easily seen that $sort(p_1 \,|\, p_2)$ is the least set satisfying the following recursive definition: $sort(p_1 \,|\, p_2) = sort(p_1) \cup sort(p_2) \cup \{\sigma \;\big|\; \{\sigma_1, \sigma_2\} \subseteq sort(p_1 \,|\, p_2), Sync(\sigma_1, \sigma_2, \sigma)\}$, which is indeed a finite set, as $sort(p_1)$ and $sort(p_2)$ are finite, by induction, and, by the well-formedness assumption, only finitely many synchronizations can be generated in this way (*cf.* Proposition 7.36). Finally, we can assume that the cardinality of the set of states reachable from the (possibly restricted) RMCS process p is k, and moreover that $sort(p)$ is finite; by inspecting the SOS rules (Res), it is clear that the cardinality of the set of states reachable from the RMCS process $(\nu a)p$ is less than or equal to k, and that $sort((\nu a)p) \subseteq sort(p)$. Therefore, we have the following fact.

Proposition 7.37. (Finite-state LTS) *For any RMCS process p, the LTS reachable from p,* $\mathscr{C}_p = (\mathscr{P}_p, sort(p), \rightarrow_p, p)$, *is finite-state.* □

With respect to the net semantics, RMCS is more expressive than RCS (see Section 6.8), as it represents, besides all the concurrent FSMs and many finite CCS nets, also many finite P/T nets that are not CCS nets. For instance, the net in Figure 7.4 is a finite P/T net, but not a CCS net, originating from the RMCS process $\underline{a}.b.\overline{a}.\mathbf{0} + \overline{b}.\underline{a}.b.\mathbf{0}$.

Now we want to prove that, for any RMCS process p, $Net(p)$ is a *bounded*, finite P/T net. By Theorem 7.11, any well-formed FNM process p is such that $Net(p)$ is a finite P/T net; hence, since RMCS is a subcalculus of well-formed FNM, $Net(p)$ is a finite P/T net for any RMCS process p, too. So, it remains to prove that such a net is bounded. By syntactic definition, the basic constituents of any RMCS process

[6] Note that the parallel composition operator cannot occur syntactically in any term p of category q; hence, only p is structurally congruent to p, so that rule (Cong) is useless in this case.

[7] This fact does hold for the structural congruence $\equiv$ induced only by the associativity and commutativity axioms in Table 7.5; however, more generous structural congruences, e.g., those including the identity axiom $p = p \,|\, \mathbf{0}$, do not enjoy this finiteness property.

$p = (\nu L)(p_1 \mid \ldots \mid p_n)$ are the processes $p_1, \ldots, p_n$ in syntactic category q. Of course, the net transitions that any sequential p_i may generate have a singleton pre-set and a post-set which is a singleton at most. By rule (s-com), two net transitions t_1 and t_2 can be synchronized, producing a transition t with pre-set ${}^\bullet t = {}^\bullet t_1 \oplus {}^\bullet t_2$ and post-set $t^\bullet = t_1^\bullet \oplus t_2^\bullet$, so that $|{}^\bullet t| = 2$ and $|t^\bullet| \leq 2$; of course, t can be used as a premise for rule (s-com) to generate a new transition with a pre-set of size three and a post-set of size not larger than three; however, since p is well formed, the number of applications of rule (s-com) is bounded by the length of the longest sequence that any p_i can generate. Hence, if j is the length of the longest sequence of all the p_i's, the generable net transitions have a pre-set of size $j+1$ and a post-set of size $j+1$ at most. Summing up, for any m and any t, if $m[t\rangle m'$, then $|m'| \leq |m|$, because t does not produce more tokens than it consumes. Therefore, the following theorem holds.

Theorem 7.18. *For any process p of RMCS, $Net(p)$ is a k-bounded, finite P/T net, where $k = |dec(p)|$.* □

However, RMCS is not able to express all the possible bounded, finite P/T nets. For instance, consider the simple net

$$N(\{s_1\}) = (\{s_1, s_2, s_3\}, \{a, b, c\}, \{(\{s_1\}, a, \{s_2, s_3\}), (\{s_2\}, b, \emptyset), (\{s_3\}, c, \emptyset)\}, \{s_1\}),$$

depicted in Figure 3.6(a). Clearly, no RMCS process can model such a net, because the transition $(\{s_1\}, a, \{s_2, s_3\})$ produces more tokens than it consumes.

Chapter 8
Adding Atomic Tests for Absence: NPL

Abstract FNM is extended with a construct for atomic tests, which is exploited to check the absence of those parallel processes whose presence may impede the executability of the current transition. The effect is that parallel composition is no longer a permissive operator, as different parallel contexts may enable or disable the transitions performable by some component. The resulting language, called *Nonpermissive Petri net Language* (NPL for short), is Turing-complete and represents the class of finite Nonpermissive Petri nets.

8.1 Syntax

As for FNM, let $Act = \mathscr{L} \cup \overline{\mathscr{L}} \cup \{\tau\}$, such that $\tau \not\in \mathscr{L} \cup \overline{\mathscr{L}}$, be the finite set of *actions*, ranged over by μ. Let $\mathscr{C}ons$ be a countable set of process *constants*, disjoint from Act, ranged over by $A, B, C, \ldots$, possibly indexed. Differently from FNM, let $\mathscr{T}cons \subseteq \mathscr{C}ons$ be a countable set of *testable* process constants, ranged over by $D, D', \ldots$, possibly indexed; moreover, let $\neg\mathscr{T}cons = \{\neg D \bigm| D \in \mathscr{T}cons\}$ be the set of *negated testable constant names*.

Nonpermissive Petri net Language (NPL for short) is the calculus whose processes are generated from actions, process constants and negated testable constant names as described by the following abstract syntax:

$$\begin{array}{lllllll}
s ::= a.t & \mid\ \tau.t & \mid\ \underline{a}.s & \mid\ \underline{\neg D}.s & \mid\ s+s & & \\
q ::= s & \mid\ \mathbf{0} & \mid\ \overline{a}.t & \mid\ q+q & & & \\
r ::= q & \mid\ C & & & & & \textit{sequential terms} \\
t ::= r & \mid\ t\,|\,t & & & & & \textit{restriction-free terms} \\
p ::= t & \mid\ (\nu a)p & & & & & \textit{general terms}
\end{array}$$

where $a \in \mathscr{L}$, $\overline{a} \in \overline{\mathscr{L}}$, $\neg D \in \neg\mathscr{T}cons$ and any constant C is equipped with a defining equation in syntactic category q. The only addition to the syntax of Section 7.1.3 for *well-formed* FNM is that now the strong prefixing operator may use negated testable constants as strong prefixes. In this enriched setting, the label of a transition is a pair

R. Gorrieri, *Process Algebras for Petri Nets*, Monographs in Theoretical Computer Science. An EATCS Series, DOI 10.1007/978-3-319-55559-1_8

(σ,I), where the atomic sequence σ may contain only actions, as for FNM, or be a single testable constant, while $I \subseteq \mathscr{T}cons$ is the set of the testable constants that occurred negated in some strong prefixes. The operational rule for this new construct is

$$(\text{S-Pref}_2)\ \frac{p \xrightarrow{(\sigma,I)} p'}{\underline{\neg D}.p \xrightarrow{(\sigma,I\cup\{D\})} p'}$$

where the strong prefix $\neg D$ contributes to the enlargement of the set I. In a transition $p \xrightarrow{(\sigma,I)} p'$, the set I signals that p may perform σ in parallel with some process q only if, for each $D \in I$, q does not contain any currently active instance of D (contextual atomic test for absence). This can be checked because a testable constant D can now visibly show its active presence by performing an idling, self-loop transition labeled $(D,\emptyset)$. Hence, the parallel composition operator is context-sensitive and no longer permissive, as the asynchronous activities of p may be enabled or disabled by different parallel contexts.

An NPL term p is an NPL *process* if it is fully defined and $Const(p)$ is finite, where function $Const(-)$, defined for the FNC operators in Section 6.1, is extended to strong prefixing as follows:

$Const(\underline{a}.s) = Const(s), \qquad Const(\underline{\neg D}.s) = \{D\} \cup Const(s).$

The set of NPL processes is denoted by $\mathscr{P}_{NPL}$, and the set of its sequential processes, i.e., of the processes in syntactic category r, by $\mathscr{P}_{NPL}^{seq}$.

We extend the definitions of free names $F(p,I)$ and free outputs $G(p,I)$ (see Definition 6.1) to strong prefixing as follows:

$F(\underline{a}.s,I) = F(s,I) \cup \{a\}, \qquad G(\underline{a}.s,I) = G(s,I),$

$F(\underline{\neg D}.s,I) = F(s,I), \qquad G(\underline{\neg D}.s,I) = G(s,I).$

Of course, for any NPL process p, the sets $fn(p)$ (free names of p), $bn(p)$ (bound names of p) and $fo(p)$ (free outputs of p) are finite. A process p is *closed* if the set $fn(p)$ of its free names is empty. Process p is *output-closed* if the set $fo(p)$ of its free outputs is empty.

As for FNC and FNM, we need to use *extended terms*, whose syntax is

$$\begin{array}{llllllll}
s ::= \gamma.t & \mid\ \tau.t & \mid\ \underline{\gamma}.s & \mid\ \underline{\neg D}.s & \mid\ s+s & & \gamma \in \mathscr{G} = \mathscr{L} \cup \mathscr{L}' \\
q ::= \ s & \mid\ \mathbf{0} & \mid\ \overline{\gamma}.t & \mid\ q+q & & & \overline{\gamma} \in \overline{\mathscr{G}} = \overline{\mathscr{L}} \cup \overline{\mathscr{L}'} \\
r ::= \ q & \mid\ C & & & & & \textit{sequential extended terms} \\
t ::= \ r & \mid\ t \,|\, t & & & & & \textit{restriction-free extended terms} \\
p ::= \ t & \mid\ (\nu a)p & & & & & \textit{general extended terms}
\end{array}$$

where the prefixes are taken from the set $Act_\gamma = \mathscr{G} \cup \overline{\mathscr{G}} \cup \{\tau\}$, the strong prefixes from the set $\mathscr{G} \cup \neg\mathscr{T}cons$ and the bound actions from the set $\mathscr{L}$. An extended NPL term $p = (\nu L)t$ is an *extended process* if $Const(p)$ is finite and t is *admissible*, i.e.,

$$\forall a \in \mathscr{L}, \{a,a'\} \not\subseteq fn(t),$$

where the function $fn(-)$ is defined on extended terms in the obvious way. By the notation $ad(t)$ we indicate that t is admissible.

By $\mathscr{P}_{NPL}^{\gamma}$ we denote the set of all extended NPL processes. By $\mathscr{P}_{NPL}^{\gamma,par}$ we denote the set of all restriction-free, extended NPL processes, i.e., those processes in syntactic category t. By $\mathscr{P}_{NPL}^{\gamma,seq}$ we denote the set of all sequential, extended NPL processes, i.e., those processes in syntactic category r.

Syntactic substitution of a restricted action a' for all the occurrences of the visible action a in a process p, denoted by $p\{a'/a\}$, was formally defined for extended FNC processes in Section 6.1.2; it is defined for strong prefixing as follows:

$$\begin{aligned}(\underline{a}.p)\{a'/a\} &= \underline{a'}.(p\{a'/a\}),\\ (\underline{\gamma}.p)\{a'/a\} &= \underline{\gamma}.(p\{a'/a\}) \qquad\qquad \text{if } \gamma \neq a,\\ (\underline{\neg D}.p)\{a'/a\} &= \underline{\neg D}_{\{a'/a\}}.(p\{a'/a\}),\end{aligned}$$

where, in the last case, the application of the substitution creates the new negated name $\neg D_{\{a'/a\}}$. We remark that the application of a substitution $\{a'/a\}$ to a constant C preserves its type: if C is (not) testable, then the new constant $C_{\{a'/a\}}$ is (not) testable, too. Of course, for any extended NPL process p and for any $a \in \mathscr{L}$, $p\{a'/a\}$ is an extended NPL process. Moreover, for any extended, restriction-free NPL process t such that $L' = fn(t) \cap \mathscr{L}'$, we have that $(t\{L/L'\})\{L'/L\} = t$, by admissibility of t, where $\{L/L'\}$ is the inverse substitution (see Remark 6.2).

Function $i : \mathscr{P}_{NPL} \to \mathscr{P}_{NPL}^{\gamma,par}$, defined as $i((\nu L)t) = t\{L'/L\}$, maps NPL processes to restriction-free, extended NPL processes. Function i is the identity over restriction-free processes: $i(t) = t$. In general, i is not injective; however, it is possible to prove that i is surjective, as done for FNM. We will see that the net semantics for an NPL process p is actually given to its corresponding restriction-free, extended NPL process $i(p)$, because it holds that $dec(p) = dec(i(p))$ (Proposition 8.13).

The set of sequential subterms $sub(p)$ of an NPL process p is computed as for FNM, where the function $sub(p,I)$ is extended to the new strong prefixing as

$$sub(\underline{\neg D}.s,I) = \{\underline{\neg D}.s\} \cup sub(D,I) \cup sub(s,I),$$

where the addition of $sub(D,I)$ reflects the fact that the testable constant D is involved in the transitions that $\underline{\neg D}.s$ may perform; in particular, in the net semantics, D belongs to the neg-set of its initial net transition; this addition is needed in order to prove that the places reachable from a process p are included into the set $sub(p)$ of its sequential subterms (Theorem 8.6). Not surprisingly, all the results proved in Section 7.1.2 hold for NPL, too. We simply list them, without details of the proofs.

Lemma 8.1. *For any extended, restriction-free NPL process t and for any $L \subseteq \mathscr{L}$, $sub(t)\{L'/L\} = sub(t\{L'/L\})$.*

Proof. *By induction on the definition of $sub(t,\emptyset)$.* □

Proposition 8.1. *For any NPL process p, $sub(p) = sub(i(p))$.* □

Theorem 8.1. *For any NPL process p, the set of its sequential subterms $sub(p)$ is finite; moreover, $sub(p) \subseteq \mathscr{P}_{NPL}^{\gamma,seq}$.*

Proof. *By induction on the definition of $sub(p,\emptyset)$.* □

Proposition 8.2. *For any NPL processes p and q, the following hold:*

(i) *if p is sequential, then* $p \in sub(p)$*, and*
(ii) *if* $p \in sub(q)$*, then* $sub(p) \subseteq sub(q)$*.* □

8.2 Operational LTS Semantics

The *concrete* interleaving operational semantics for NPL is given by the LTS $\mathscr{C}^c_{NPL} = (\mathscr{P}_{NPL}, Lab, \longrightarrow)$, where the states are the processes in $\mathscr{P}_{NPL}$, the set Lab of labels is $\mathscr{A}_d \times \mathscr{P}_{fin}(\mathscr{T}cons)$, where $\mathscr{A}_d = \{\tau\} \cup \mathscr{T}cons \cup \overline{\mathscr{L}} \cup \mathscr{L}^+$ is ranged over by σ, while $\mathscr{P}_{fin}(\mathscr{T}cons)$ is ranged over by I, and $\longrightarrow \subseteq \mathscr{P}_{NPL} \times Lab \times \mathscr{P}_{NPL}$ is the least transition relation generated by the axioms and rules in Table 8.1.

We briefly comment on the rules that are less standard. Rule (S-Pref$_2$) allows for the creation of transitions with a nonempty set I of testable constant names. Axiom (Cons$_1$) allows any testable constant D to exhibit its active presence by performing the idling, self-loop transition labeled $(D, \emptyset)$.

Rule (Par) requires a contextual check by means of *negative* premises: p can perform (σ, I) also in the parallel context of a process q, provided that, for all $D \in I$, q cannot exhibit any label $(D, \emptyset)$, i.e., no instance of D is ready to execute. Therefore, the parallel composition operator of NPL is not *permissive*, as q can impede p from performing its sequence σ asynchronously.

For rule (S-Com), relation $Sync(\sigma_1, \overline{a}, \sigma)$, defined by the axiom and rule of Table 8.2, follows the synchronization discipline of FNM.[1] The crucial request in the premise of the rule is that q be a *sequential* NPL process, and additionally that the set of testable constant names is empty. This is not a restrictive request because any *well-formed* parallel process t (and all the NPL processes are well formed by syntactic definition) such that $t \xrightarrow{(\overline{a},I)} t'$ is structurally congruent to $q \,|\, r$, for some suitable sequential q such that $q \xrightarrow{(\overline{a},I)} q'$ and $t' \equiv q' \,|\, r$. Moreover, we also prove that the set I is always empty when the transition is labeled by an output (see Proposition 8.3 below). By means of rule (Cong), the parallel process $p \,|\, t$ can be first turned into the structurally congruent process $(p \,|\, r) \,|\, q$, so that the synchronization between p and q can be performed by application of rule (S-Com), provided that r does not impede p from performing asynchronously the atomic sequence σ_1 (see Example 8.2).

Rule (Cong) is as usual, with the structural congruence $\equiv$ defined by the two axioms of Table 8.3. This rule justifies the absence of the symmetric rules for (Par) and (S-Com).

Rule (S-Res) is a bit nonstandard, as it requires that the label (σ, I) of the premise transition be subject to the substitution $\{a'/a\}$, which applies to all testable constants only, as $a \notin n(\sigma)$; formally, $(D, \emptyset)\{a'/a\} = (D_{\{a'/a\}}, \emptyset)$ and $(\sigma, I)\{a'/a\} = (\sigma, I\{a'/a\})$ if $\sigma \notin \mathscr{T}cons$. This constant name relabeling has no operational effect, as the restricted process $(\nu a)p$ cannot be put in parallel with some other process, by

[1] Note that relation *Sync* is slightly simplified w.r.t. the definition in Chapter 7; see Remark 7.2.

$$\text{(Pref)}\ \frac{}{\mu.p \xrightarrow{(\mu,\emptyset)} p} \qquad \text{(Cong)}\ \frac{p \equiv p' \xrightarrow{(\sigma,I)} q' \equiv q}{p \xrightarrow{(\sigma,I)} q}$$

$$\text{(Sum}_1\text{)}\ \frac{p \xrightarrow{(\sigma,I)} p'}{p+q \xrightarrow{(\sigma,I)} p'} \qquad \text{(Cons}_1\text{)}\ \frac{}{D \xrightarrow{(D,\emptyset)} D} \quad D \in \mathscr{T}cons$$

$$\text{(Sum}_2\text{)}\ \frac{q \xrightarrow{(\sigma,I)} q'}{p+q \xrightarrow{(\sigma,I)} q'} \qquad \text{(Cons}_2\text{)}\ \frac{p \xrightarrow{(\sigma,I)} p'}{C \xrightarrow{(\sigma,I)} p'} \quad C \doteq p$$

$$\text{(S-Pref}_1\text{)}\ \frac{p \xrightarrow{(\sigma,I)} p'}{\underline{a}.p \xrightarrow{(a\diamond\sigma,I)} p'} \qquad a \diamond \sigma = \begin{cases} a & \text{if } \sigma = \tau, \\ a\sigma & \text{otherwise} \end{cases}$$

$$\text{(Par)}\ \frac{p \xrightarrow{(\sigma,I)} p' \quad \forall D \in I\ q \overset{(D,\emptyset)}{\not\rightarrow}}{p\,|\,q \xrightarrow{(\sigma,I)} p'\,|\,q} \qquad \text{(S-Pref}_2\text{)}\ \frac{p \xrightarrow{(\sigma,I)} p'}{\underline{\neg D}.p \xrightarrow{(\sigma,I\cup\{D\})} p'}$$

$$\text{(S-Com)}\ \frac{p \xrightarrow{(\sigma_1,I)} p' \quad q \xrightarrow{(\bar{a},\emptyset)} q' \quad q \in \mathscr{P}^{seq}_{NPL}}{p\,|\,q \xrightarrow{(\sigma,I)} p'\,|\,q'} \quad Sync(\sigma_1,\bar{a},\sigma)$$

$$\text{(S-Res)}\ \frac{p \xrightarrow{(\sigma,I)} p'}{(\nu a)p \xrightarrow{(\sigma,I)\{a'/a\}} (\nu a)p'} \quad a \notin n(\sigma)$$

$$\text{(Abs)}\ \frac{p \xrightarrow{(\sigma,I)} p'}{p \overset{\sigma}{\longmapsto} p'} \quad \sigma \notin \mathscr{T}cons$$

Table 8.1 LTS operational rules for NPL

$$\frac{}{Sync(a,\bar{a},\tau)} \qquad \frac{\sigma \neq \varepsilon}{Sync(a\sigma,\bar{a},\sigma)}$$

Table 8.2 Synchronization relation *Sync*

E1	$(p\,|\,q)\,|\,r = p\,|\,(q\,|\,r)$
E2	$p\,|\,q = q\,|\,p$

Table 8.3 Axioms generating the structural congruence $\equiv$

syntactic definition. The usefulness of this substitution is only for getting a precise correspondence between the LTS semantics and the net semantics we will define in Section 8.4; in particular, see the proofs of Propositions 8.21 and 8.26.

Proposition 8.3. *For any restriction-free NPL process p, if $p \xrightarrow{(\bar{a},I)} p'$ is a transition derivable by the rules in Table 8.1, then $I = \emptyset$ and, additionally, either $p \in \mathscr{P}^{seq}_{NPL}$ or there exist $q \in \mathscr{P}^{seq}_{NPL}$ and t such that $p \equiv q\,|\,t$, $q \xrightarrow{(\bar{a},\emptyset)} q'$ and $p' \equiv q'\,|\,t$.*

Proof. *By induction on the proof of $p \xrightarrow{(\overline{a},I)} p'$. The base case is axiom (Pref) when $p = \overline{a}.p'$: the derivable transition is $\overline{a}.p' \xrightarrow{(\overline{a},\emptyset)} p'$, and the thesis is satisfied, as $\overline{a}.p'$ is sequential. If the last rule used is (Cons$_2$) with $C \doteq r$, then $C \xrightarrow{(\overline{a},I)} r'$ is derivable by the premise transition $r \xrightarrow{(\overline{a},I)} r'$. By induction, we can assume that $I = \emptyset$; hence, the thesis is satisfied as C is sequential. The case of rules (Sum$_1$) and (Sum$_2$) is similar, and so omitted.*

If the last rule used is (Par), then $p = p_1 \,|\, p_2$ and the premise transition is $p_1 \xrightarrow{(\overline{a},I)} p_1'$, with $p' = p_1' \,|\, p_2$. By induction, we can assume that the thesis holds for the premise transition; hence, $I = \emptyset$ and either p_1 is sequential or there exists a sequential process q_1 such that $p_1 \equiv q_1 \,|\, t$, $q_1 \xrightarrow{(\overline{a},\emptyset)} q_1'$ and $p_1' \equiv q_1' \,|\, t$. Hence, the thesis holds also for $p_1 \,|\, p_2$.

If the last rule used is (Cong), then $p \equiv p_1$, $p' \equiv p_1'$ and the premise transition is $p_1 \xrightarrow{(\overline{a},I)} p_1'$. By induction, we can assume that the thesis holds for the premise transition; hence, $I = \emptyset$ and either p_1 is sequential or there exists a sequential process q_1 such that $p_1 \equiv q_1 \,|\, t$, $q_1 \xrightarrow{(\overline{a},\emptyset)} q_1'$ and $p_1' \equiv q_1' \,|\, t$. Hence, the thesis holds also for p.

No other rule is applicable, because by syntactic definition an output is not producible by a strongly prefixed process (which is in syntactic category s) and, moreover, the synchronization of an input sequence with one output produces a shorter input sequence or τ, which is not an output. □

Example 8.1. Let us consider the constant definitions

$$C \doteq \underline{\neg D}.a.0, \qquad D \doteq \overline{a}.\mathbf{0}.$$

Constants C and D can perform, besides the obvious self-loop transition $D \xrightarrow{(D,\emptyset)} D$ due to axiom (Cons$_1$), also the following transitions:

$$\text{(Cons)}\ \dfrac{\text{(S-Pref}_2\text{)}\ \dfrac{\text{(Pref)}\ \dfrac{}{a.\mathbf{0} \xrightarrow{(a,\emptyset)} \mathbf{0}}}{\underline{\neg D}.a.\mathbf{0} \xrightarrow{(a,\{D\})} \mathbf{0}}}{C \xrightarrow{(a,\{D\})} \mathbf{0}} \qquad\qquad \text{(Cons)}\ \dfrac{\text{(Pref)}\ \dfrac{}{\overline{a}.\mathbf{0} \xrightarrow{(\overline{a},\emptyset)} \mathbf{0}}}{D \xrightarrow{(\overline{a},\emptyset)} \mathbf{0}}$$

The process $C \,|\, D$ can synchronize as follows:

$$\text{(S-Com)}\ \dfrac{\dfrac{}{C \xrightarrow{(a,\{D\})} \mathbf{0}} \qquad \dfrac{}{D \xrightarrow{(\overline{a},\emptyset)} \mathbf{0}}}{C \,|\, D \xrightarrow{(\tau,\{D\})} \mathbf{0} \,|\, \mathbf{0}}$$

but C cannot perform a asynchronously: even if $C \xrightarrow{(a,\{D\})} \mathbf{0}$ is derivable, the premise of rule (Par) is invalidated by the presence of D. Note also that $(C \,|\, D) \,|\, D$ cannot perform the silent transition proved above: even if $C \,|\, D \xrightarrow{(\tau,\{D\})} \mathbf{0} \,|\, \mathbf{0}$, the premise of rule (Par) is invalidated by the presence of the additional constant D. □

$$
\text{(Cong)}\;\dfrac{\text{(S-Com)}\;\dfrac{\text{(Par)}\;\dfrac{\text{(Par)}\;\dfrac{\dfrac{}{C \xrightarrow{(a,\{D\})} \mathbf{0}}}{C\,|\,E \xrightarrow{(a,\{D\})} \mathbf{0}\,|\,E}}{(C\,|\,E)\,|\,E \xrightarrow{(a,\{D\})} (\mathbf{0}\,|\,E)\,|\,E} \qquad \dfrac{}{F \xrightarrow{(\overline{a},\emptyset)} \mathbf{0}}}{(C\,|\,E)\,|\,(E\,|\,F) \equiv ((C\,|\,E)\,|\,E)\,|\,F \xrightarrow{(\tau,\{D\})} ((\mathbf{0}\,|\,E)\,|\,E)\,|\,\mathbf{0} \equiv (\mathbf{0}\,|\,E)\,|\,(E\,|\,\mathbf{0})}}{(C\,|\,E)\,|\,(E\,|\,F) \xrightarrow{(\tau,\{D\})} (\mathbf{0}\,|\,E)\,|\,(E\,|\,\mathbf{0})}
$$

Table 8.4 Contextual synchronization between C and F

Example 8.2. Continuing the example above, let us consider also

$$E \doteq b.0, \qquad F \doteq \overline{a}.\mathbf{0},$$

and the process $(C\,|\,E)\,|\,(E\,|\,F)$. The synchronization between C and F can be proved as described in Table 8.4, where rule (Cong) was used to give a different association to the four processes. Note that $(C\,|\,E)\,|\,(D\,|\,F) \overset{(\tau,\{D\})}{\not\longrightarrow} (\mathbf{0}\,|\,E)\,|\,(D\,|\,\mathbf{0})$, because the instance of the only testable constant D inside $(C\,|\,E)\,|\,D$ does not allow the asynchronous execution of transition $C \xrightarrow{(a,\{D\})} \mathbf{0}$. □

The *abstract* interleaving operational semantics for NPL is given by the LTS $\mathscr{C}^a_{NPL} = (\mathscr{P}_{NPL}, \mathscr{A}, \longmapsto)$, where the set of *abstract* labels is $\mathscr{A} = \{\tau\} \cup \overline{\mathscr{L}} \cup \mathscr{L}^+$ and $\longmapsto \subseteq \mathscr{P}_{NPL} \times \mathscr{A} \times \mathscr{P}_{NPL}$ is the least transition relation generated by rule (Abs) in Table 8.1. This rule states that idling transitions of testable constants, due to axiom (Cons$_1$), in the concrete interleaving LTS are not imported into the abstract LTS (condition $\sigma \notin \mathscr{T}cons$); however, all the other transitions of the concrete LTS are imported into the abstract LTS, but forgetting the set I. The abstract LTS makes sense only for observing *fixed* processes, i.e., processes not composable with any other process, working in isolation with respect to the external environment. In this case, the set of testable constant names in the labels are no longer necessary and can be safely abstracted away.

8.2.1 Expressiveness

From a syntactic point of view, well-formed FNM is a proper subcalculus of NPL. From a semantic point of view, we can prove that NPL is a *conservative extension* of well-formed FNM: for any well-formed FNM process p, the rooted LTS for p generated by means of the operational rules in Table 7.3 is isomorphic to the rooted *abstract* LTS for p generated by means of the operational rules in Table 8.1.

However, NPL describes systems that cannot be represented by well-formed FNM, as illustrated in the following example.

Example 8.3. **(Counter)** Let us now consider a real counter, i.e., a semi-counter (discussed in Example 5.9) that can also test for zero:

$$\begin{aligned}
Counter_0 &\doteq zero.Counter_0 + inc.Counter_1,\\
Counter_n &\doteq inc.Counter_{n+1} + dec.Counter_{n-1} \qquad n > 0.
\end{aligned}$$

Its definition is not finitary because it uses infinitely many constants. Its LTS is very similar to the one for the semi-counter (see Figure 2.3(b)), with the only difference being that there is a self-loop transition on the first state q_0, labeled *zero*. Indeed, this process can *test for zero*, while the semi-counter process $SCount_0$ of Example 5.9 cannot. This process can be proved bisimulation equivalent to the following NPL (fixed, i.e., observed in the abstract interleaving LTS) process C, using only two constants:

$$\begin{aligned}
C &\doteq \underline{\neg D}.zero.C + inc.(C\,|\,D),\\
D &\doteq dec.\mathbf{0}.
\end{aligned}$$

The relation $R = \{(Counter_n, C\,|\,\Pi_{i=1}^{n} D) \;\big|\; n \geq 0\}$ is a strong bisimulation, up to $\equiv_1$ (Definition 2.14), where $\equiv_1$ is the structural congruence induced by the three axioms **E1-E2** of associativity and commutativity in Table 8.3, and **E3**, stating that $\mathbf{0}$ is the neutral element for parallel composition: $p\,|\,\mathbf{0} = p$. Note that $\equiv_1$ is finer than $\sim$ (strong bisimilarity on the abstract LTS) and so a strong bisimulation up to $\equiv_1$ is a sound proof technique for strong bisimilarity. First, observe that for $n = 0$, the pair in R is $(Counter_0, C\,|\,\mathbf{0})$. If R is a bisimulation up to $\equiv_1$, then $Counter_0 \sim C\,|\,\mathbf{0}$. As $C\,|\,\mathbf{0} \sim C$, by transitivity we get our expected result: $Counter_0 \sim C$. Now, let us prove that R is indeed a strong bisimulation up to $\equiv_1$.

Assume that $Counter_n \xrightarrow{\alpha} q$. Then, three cases are possible: $\alpha = inc$ and $q = Counter_{n+1}$, or $n > 0$, $\alpha = dec$ and $q = Counter_{n-1}$, or $n = 0$, $\alpha = zero$ and $q = Counter_0$. In the first case, the matching abstract transition is

$$C\,|\,\Pi_{i=1}^{n} D \xmapsto{inc} (C\,|\,D)\,|\,\Pi_{i=1}^{n} D,$$

where the reached state is in relation $\equiv_1$ with $C\,|\,\Pi_{i=1}^{n+1} D$, and the pair $(Counter_{n+1}, C\,|\,\Pi_{i=1}^{n+1} D)$ is in R. In the second case, (one of) the matching abstract transition(s), labeled *dec*, starts from $C\,|\,\Pi_{i=1}^{n} D$ and reaches $(C\,|\,\Pi_{i=1}^{n-1} D)\,|\,\mathbf{0}$, which is in relation $\equiv_1$ with $C\,|\,\Pi_{i=1}^{n-1} D$, and the pair $(Counter_{n-1}, C\,|\,\Pi_{i=1}^{n-1} D)$ belongs to R. In the third case, the required abstract transition is $C\,|\,\mathbf{0} \xmapsto{zero} C\,|\,\mathbf{0}$ and $(Counter_0, C\,|\,\mathbf{0}) \in R$.

Assume now $C\,|\,\Pi_{i=1}^{n} D \xmapsto{\alpha} p$. Then, by inspecting the rules for parallel composition, we have that

1. *Either* $C \xrightarrow{(inc,\emptyset)} C\,|\,D$ and thus $\alpha = inc$ and process p is $(C\,|\,D)\,|\,\Pi_{i=1}^{n} D$ (which is in relation $\equiv_1$ with $C\,|\,\Pi_{i=1}^{n+1} D$). In this case, $Counter_n \xrightarrow{inc} Counter_{n+1}$ is the matching transition, and the pair $(Counter_{n+1}, C\,|\,\Pi_{i=1}^{n+1} D)$ is in R.

2. *Or* $n > 0$, $D \xrightarrow{(dec,\emptyset)} \mathbf{0}$, $\alpha = dec$ and p is one of the following three terms: $(C\,|\,\mathbf{0})\,|\,\Pi_{i=1}^{n-1}D$, $(C\,|\,\Pi_{i=1}^{n-1}D)\,|\,\mathbf{0}$ or $(((C\,|\,\Pi_{i=1}^{n-k}D)\,|\,\mathbf{0})\,|\,\Pi_{i=1}^{k-1}D)$ for some $1 < k < n$. In any case, p is in relation $\equiv_1$ with $C\,|\,\Pi_{i=1}^{n-1}D$. The matching transition is
   $$Counter_n \xrightarrow{dec} Counter_{n-1},$$
   and the pair $(Counter_{n-1}, C\,|\,\Pi_{i=1}^{n-1}D)$ is in R.
3. *Or* $n = 0$, $C \xrightarrow{(zero,\{D\})} C$, $\alpha = zero$ and $p = C\,|\,\mathbf{0}$. The matching transition is
   $$Counter_0 \xrightarrow{zero} Counter_0,$$
   and the pair $(Counter_0, C\,|\,\mathbf{0})$ is in R.

And this completes the proof. So, we have shown that a counter can be represented, up to $\sim$, by a simple NPL fixed process.

We remark that there is no well-formed FNM process q such that q is equivalent to C (for any reasonable semantics, e.g., weak trace equivalence), i.e., the counter C is a proper representative of NPL. As a matter of fact, on the one hand, any well-formed FNM process generates a finite P/T Petri net. On the other hand, Agerwala in [Age75] (see also [Tau89], page 120) proved that no finite P/T net can faithfully represent a counter; hence, the conclusion is that no well-formed FNM process can represent a counter. □

Moreover, we will show that NPL represents the class of finite Nonpermissive Petri nets (finite NP/T nets for short), which strictly includes the class of finite P/T nets, which is representable by well-formed FNM. Finite NP/T nets are a Turing-complete model of computation, because they can model faithfully any counter machine, as shown in Section 3.5; therefore, NPL is Turing-complete, too. A direct proof that NPL can faithfully represent any counter machine (CM, for short) can be given in the line of what [BGZ98, BGZ09, GV15] did for other calculi. First, [Min67] observed that two counters are enough to get Turing-completeness, so we can limit ourselves to this subclass of counter machines.

Assuming that the tuple (v_1, v_2) denotes the initial inputs for the two counters of the CM $M = (I, 2)$, with $|I| = m$, the associated NPL process is

$$CM_{M(v_1,v_2)} \doteq (\nu N)(C_1\,|\,C_2\,|\,B_{(v_1,v_2)}),$$

where each constant C_j defines the CM counter r_j, for $j = 1, 2$, as follows:

$$C_j \doteq \underline{\neg D}_j.zero_j.C_j + inc_j.(C_j\,|\,D_j),$$
$$D_j \doteq dec_j.\mathbf{0},$$

and constant $B_{(v_1,v_2)}$, which performs the bootstrapping of the system by initializing the counters and by activating the first instruction associated with constant P_1, is defined as follows:

$$B_{(v_1,v_2)} \doteq \underbrace{\overline{inc}_1.\cdots.\overline{inc}_1}_{v_1 \text{ times}}.\underbrace{\overline{inc}_2.\cdots.\overline{inc}_2}_{v_2 \text{ times}}.P_1.$$

Each instruction $(i : I_i)$ corresponds to the definition of a constant P_i.

- An increment operation $(i : Inc(r_j))$ is modeled as

$$P_i \doteq \overline{inc}_j.P_{i+1},$$

where $\overline{inc}_j$ is the increment operation on register C_j, and P_{i+1} is the constant for the next instruction.

- A "jump if zero/decrement" operation $(i : DecJump(r_j, s))$ is modeled as

$$P_i \doteq \overline{zero}_j.P_s + \overline{dec}_j.P_{i+1},$$

where the choice between $\overline{zero}_j$ and $\overline{dec}_j$ is driven by the current status of counter C_j: if C_j holds 0, then only the synchronization on $zero_j$ can take place (and the instruction of index s is activated), while if C_j is not 0, then only the synchronization on dec_j can take place (with the effect of decrementing the register and of activating instruction of index $i+1$).

The set N of bound actions is $\{inc_j, zero_j, dec_j \mid 1 \leq j \leq 2\}$, in order to force the synchronizations between the instructions and the counters. Therefore, the modeling of the CM $M = (I, 2)$ is given in NPL by using only six actions (those in N), two negated testable constant prefixes ($\neg D_1, \neg D_2$) and $4+m+1$ constants, i.e., 4 for the two counters and $m+1$ for the m instructions.[2] It is an open problem to find the least number of constants that are needed to get Turing-completeness in NPL. However, note that a universal Turing machine (UTM, for short) can be simulated by a 2-counter machine [Min67]; hence, the number of NPL constants that are sufficient to get Turing-completeness is $4+m+1$, if m are the instructions of the 2-counter machine simulating a UTM.

Definition 8.1. (NPL language) A language $L \subseteq \mathscr{L}^*$ is an *NPL language* if there exists an output-closed NPL process p such that the set of its weak completed traces is L, i.e., $WCTr(p) = L$. □

As NPL is Turing-complete, it can be proved that the class of NPL languages coincides with the class of recursively enumerable languages [HMU01].

Example 8.4. **(LMS problem)** A possible solution to the Last Man Standing (LMS) problem, discussed at the end of Section 3.5, can be given, for any $n > 0$, by the NPL process $(\nu c, d)\Pi_{i=1}^{n} C$, where

$$\begin{aligned}
&C \doteq \underline{\neg C}.\underline{\neg D}.a.\mathbf{0} \;+\; (\underline{\neg D}.c.D + \overline{c}.\mathbf{0}) \;+\; d.\mathbf{0},\\
&D \doteq \overline{d}.D + \underline{\neg C}.b.\mathbf{0}.
\end{aligned}$$

The associated, extended NPL process $i((\nu c, d)\Pi_{i=1}^{n} C)$ is the restriction-free term

$$\Pi_{i=1}^{n}\overline{C} = \underbrace{\overline{C} \mid \cdots \mid \overline{C}}_{n},$$

[2] Indeed, constant B is not necessary, as it can be replaced by its definition; similarly, constant $CM_{M(v_1,v_2)}$ is just for notational convenience; moreover, all the indexes of undefined instructions can be identified with just one additional index/constant.

where $\overline{C} = C\{c'/c\}\{d'/d\}$. Then, when $n = 1$, $\Pi_{i=1}^{1}\overline{C} = \overline{C}$ is such that it can perform a immediately, but not b. On the contrary, when $n \geq 2$, $\Pi_{i=1}^{n}\overline{C}$ is such that it can perform b, after some internal transitions, but not a; in fact, the only available initial transition is a synchronization between two instances of $\overline{C}$ on c', which has the effect of activating an instance of $D\{d'/d\}$; then, any further instance of $\overline{C}$ can only synchronize with $D\{d'/d\}$ on d', by regenerating $D\{d'/d\}$; finally, when no instances of $\overline{C}$ are present, $D\{d'/d\}$ can perform b. Comparing this solution to the NP/T net of Figure 3.22, we note that $\overline{C}$ corresponds to place s_1, while $D\{d'/d\}$ corresponds to place s_2. □

Example 8.5. **(The language $a^n b^n c^n$)** An NPL process whose (abstract) weak completed traces are of the form $a^n b^n c^n$, with $n \geq 0$, is $(\nu d)A$, where

$$
\begin{aligned}
A &\doteq a.(A\,|\,B_1) + \tau.B_2,\\
B_1 &\doteq \overline{d}.\mathbf{0},\\
B_2 &\doteq \underline{d}.b.(B_2\,|\,C),\\
C &\doteq \underline{\neg B_1}.c.\mathbf{0}.
\end{aligned}
$$

Observe that first a certain number $n \geq 0$ of occurrences of action a are generated, as well as parallel instances of subprocess B_1. Then, when A performs the internal τ-labeled transition and becomes B_2, the same number n of synchronizations between B_1 and B_2 are performed over the bound action d, each one labeled with action b and generating one instance of C. When all the n occurrences of action b are performed, each C becomes ready, as no occurrence of the subprocess B_1 is present anymore. Finally, the same number n of occurrences of action c can be performed. (See also Examples 3.10 and 6.5.) □

8.2.2 Congruence Problem

A bisimulation over the concrete interleaving LTS $\mathscr{C}_{NPL}^{c} = (\mathscr{P}_{NPL}, Lab, \longrightarrow)$ is a relation $R \subseteq \mathscr{P}_{NPL} \times \mathscr{P}_{NPL}$ such that if $(q_1, q_2) \in R$ then for all $(\sigma, I) \in Lab$

- $\forall q_1'$ such that $q_1 \xrightarrow{(\sigma,I)} q_1'$, $\exists q_2'$ such that $q_2 \xrightarrow{(\sigma,I)} q_2'$ and $(q_1', q_2') \in R$,
- $\forall q_2'$ such that $q_2 \xrightarrow{(\sigma,I)} q_2'$, $\exists q_1'$ such that $q_1 \xrightarrow{(\sigma,I)} q_1'$ and $(q_1', q_2') \in R$.

Two NPL processes p and q are *concrete interleaving bisimilar*, denoted by $p \sim_{int}^{c} q$, if there exists a bisimulation $R \subseteq \mathscr{P}_{NPL} \times \mathscr{P}_{NPL}$ such that $(p, q) \in R$.

Proposition 8.4. *Let $p, q \in \mathscr{P}_{NPL}$ be two NPL processes. If $p \equiv q$, then $p \sim_{int}^{c} q$.*

Proof. It is enough to check that relation $R = \{(p,q) \mid p \equiv q\}$ is a bisimulation. If $(p,q) \in R$ and $p \xrightarrow{(\sigma,I)} p'$, then by rule (Cong) also $q \xrightarrow{(\sigma,I)} p'$ and $(p', p') \in R$. Symmetrically, if q moves first. □

Two NPL processes p and q are *abstract interleaving bisimilar*, denoted by $p \sim_{int}^{a} q$, if a bisimulation $R \subseteq \mathscr{P}_{NPL} \times \mathscr{P}_{NPL}$ exists over the abstract LTS $\mathscr{C}_{NPL}^{a} = (\mathscr{P}_{NPL}, \mathscr{A}, \longmapsto)$ such that $(p,q) \in R$.

Proposition 8.5. *Let $p,q \in \mathscr{P}_{NPL}$ be two NPL processes. If $p \sim_{int}^{c} q$, then $p \sim_{int}^{a} q$.*

Proof. *A bisimulation R over the concrete LTS is also a bisimulation over the abstract LTS. If $(p,q) \in R$ and $p \overset{\sigma}{\longmapsto} p'$, then, by rule (Abs), there exists $I \subseteq \mathscr{T}cons$ such that $p \xrightarrow{(\sigma,I)} p'$. Since R is a bisimulation over the concrete LTS, there exists q' such that $q \xrightarrow{(\sigma,I)} q'$ and $(p',q') \in R$. By rule (Abs), also $q \overset{\sigma}{\longmapsto} q'$ is derivable, as required. Symmetrically, if q moves first.* □

It is possible to prove that $\sim_{int}^{c}$ and $\sim_{int}^{a}$ are congruences for almost all the operators of NPL, notably for strong prefixing. Here we list some propositions stating these congruence results for $\sim_{int}^{c}$, but the proof is the same for $\sim_{int}^{a}$.

Proposition 8.6. *Given two processes p and q in syntactic category s, if $p \sim_{int}^{c} q$, then the following hold:*
(i) $\underline{a}.p \sim_{int}^{c} \underline{a}.q$ *for all* $a \in \mathscr{L}$,
(ii) $\underline{\neg D}.p \sim_{int}^{c} \underline{\neg D}.q$ *for all* $D \in \mathscr{T}cons$.

Proof. *Let R be a bisimulation such that $(p,q) \in R$. It is easy to check that relations $R' = \{(\underline{a}.p, \underline{a}.q)\} \cup R$ and $R'' = \{(\underline{\neg D}.p, \underline{\neg D}.q)\} \cup R$ are bisimulations.* □

Proposition 8.7. *Given two NPL processes p_1 and p_2 in syntactic category q, if $p_1 \sim_{int}^{c} p_2$, then $p_1 + r \sim_{int}^{c} p_2 + r$ for all r in syntactic category q.*

Proof. *Let R be a bisimulation such that the pair (p,q) belongs to R. Then, relation $R' = \{(p+r, q+r) \mid r \text{ in syntactic category } q\} \cup R \cup \mathscr{I}$ is a bisimulation up to $\equiv$, with $\mathscr{I} = \{(r,r) \mid r \in \mathscr{P}_{NPL}\}$.* □

Proposition 8.8. *Given two restriction-free NPL processes p and q, if $p \sim_{int}^{c} q$, then $\mu.p \sim_{int}^{c} \mu.q$ for all $\mu \in Act$.*

Proof. *Let R be a bisimulation such that $(p,q) \in R$. Then, relation $R' = \{(\mu.p, \mu.q) \mid \mu \in Act\} \cup R$ is a bisimulation up to $\equiv$.* □

Proposition 8.9. *Given two processes p and q, if $p \sim_{int}^{c} q$, then $(\nu a)p \sim_{int}^{c} (\nu a)q$ for all $a \in \mathscr{L}$.*

Proof. *Let R be a bisimulation with $(p,q) \in R$. Then, $R' = \{((\nu a)p', (\nu a)q') \mid a \in \mathscr{L} \text{ and } (p',q') \in R\}$ is a bisimulation.* □

Even if the congruence results above hold also for $\sim_{int}^{a}$, when considering parallel composition only concrete interleaving bisimilarity $\sim_{int}^{c}$ makes sense. As a matter of fact, the additional, contextual information on the labels is essential to get a congruence. For instance, consider $C \doteq \mathbf{0}$ and $D \doteq \mathbf{0}$, where only D is testable. Note that $C \sim_{int}^{a} D$, as the two are deadlock states in the abstract LTS. Nonetheless, if we consider the constant $E \doteq \underline{\neg D}.a.\mathbf{0}$, then the two processes $C\,|\,E$ and $D\,|\,E$ are not equivalent in the abstract LTS, as the former can do a, while the latter cannot; hence, $\sim^a_{int}$ is not a congruence for the parallel composition operator. Note that C and D are not concrete interleaving bisimilar, as only D can perform an idling transition labeled $(D,\emptyset)$; hence, concrete interleaving bisimilarity seems a better candidate for a congruence semantics for NPL.

However, also $\sim^c_{int}$ is not a congruence for parallel composition, for the same reason that bisimilarity is not a congruence for FNM parallel composition. In fact, the FNC processes $r = \overline{a}.\overline{a}.\mathbf{0}$ and $t = \overline{a}.\mathbf{0}\,|\,\overline{a}.\mathbf{0}$. are interleaving bisimilar. However, if we consider the FNM context $\mathscr{C}[-] = -\,|\,\underline{a}.\underline{a}.c.\mathbf{0}$, we get that $\mathscr{C}[r] \not\sim^c_{int} \mathscr{C}[t]$, because the latter can execute c, i.e., $\mathscr{C}[t] \xrightarrow{(c,\emptyset)} (\mathbf{0}\,|\,\mathbf{0})\,|\,\mathbf{0}$, while $\mathscr{C}[r]$ cannot. The reason for this difference is that the process $\underline{a}.\underline{a}.c.\mathbf{0}$ can react with a number of concurrently active components equal to the length of the trace it can perform. Hence, a congruence semantics for the operator of parallel composition needs to distinguish between r and t on the basis of their different degrees of parallelism. In other words, the interleaving semantics is to be replaced by a *truly concurrent* semantics, as illustrated in the following section.

8.3 Step Semantics

The *concrete* step operational semantics for NPL is given by the STS $\mathscr{C}^{c,step}_{NPL} = (\mathscr{P}_{NPL}, \mathscr{B}, \longrightarrow_s)$, where $\mathscr{B} = \mathscr{M}_{fin}(Lab)$ — i.e., the set of all finite multisets over the set of concrete interleaving labels $Lab = \mathscr{A}_d \times \mathscr{P}_{fin}(\mathscr{T}cons)$ — is the set of labels (ranged over by M, possibly indexed), and $\longrightarrow_s \subseteq \mathscr{P}_{NPL} \times \mathscr{B} \times \mathscr{P}_{NPL}$ is the least transition relation generated by the axioms and rules in Table 8.5.

Let us comment on the rules. For sequential processes, the step transitions are labeled with a singleton. This is the case for the axioms (Pref^s) and (Cons$^s_1$), and for the rules (Cons$^s_2$), (Sum$^s_1$), (Sum$^s_2$), (S-Pref$^s_1$) and (S-Pref$^s_2$). These axioms and rules are very similar to the corresponding ones in the concrete interleaving LTS.

Rule (S-Res^s) is as expected, where the notation $n(M)$ stands for the set of actions $\bigcup_{(\sigma,I)\in M} n(\sigma)$; moreover, the substitution $\{a'/a\}$ applies only to the constant names, as $a \notin n(M)$; formally, $\emptyset\{a'/a\} = \emptyset$, $(M \oplus \{(D,\emptyset)\})\{a'/a\} = M\{a'/a\} \oplus \{(D_{\{a'/a\}},\emptyset)\}$, $(M \oplus \{(\sigma,I)\})\{a'/a\} = M\{a'/a\} \oplus \{(\sigma, I\{a'/a\})\}$ if $\sigma \notin \mathscr{T}cons$.

Rule (Par^s) requires the expected contextual check over the parallel component q: the auxiliary predicate $nct(M,q)$ holds if $\forall D \in I(M)\ q \overset{\{(D,\emptyset)\}}{\not\longrightarrow}_s$, where $I(M) = \bigcup_{(\sigma,J)\in M} J$ is the set of all the constants to be tested for absence; the abbreviation *nct* stands for *no contextual transition*.

Rule (M-Par^s) is responsible for the creation of multisets as labels. If $p \xrightarrow{M_1}_s p'$ and $q \xrightarrow{M_2}_s q'$, then the two multisets can be performed at the same time by $p\,|\,q$ provided that four contextual checks are satisfied: not only it is required that q does not impede p from performing M_1 and p does not impede q from performing M_2 — $nct(M_1,q)$ and $nct(M_2,p)$ – but also that the reached states p' and q' do not

$$(\mathrm{Pref}^s)\ \frac{}{\mu.p \xrightarrow{\{(\mu,\emptyset)\}}_s p} \qquad (\mathrm{Cong}^s)\ \frac{p \equiv p' \xrightarrow{M}_s q' \equiv q}{p \xrightarrow{M}_s q}$$

$$(\mathrm{Sum}^s_1)\ \frac{p \xrightarrow{\{(\sigma,I)\}}_s p'}{p+q \xrightarrow{\{(\sigma,I)\}}_s p'} \qquad (\mathrm{Cons}^s_1)\ \frac{}{D \xrightarrow{\{(D,\emptyset)\}}_s D}\ D \in \mathscr{T}cons$$

$$(\mathrm{Sum}^s_2)\ \frac{q \xrightarrow{\{(\sigma,I)\}}_s q'}{p+q \xrightarrow{\{(\sigma,I)\}}_s q'} \qquad (\mathrm{Cons}^s_2)\ \frac{p \xrightarrow{\{(\sigma,I)\}}_s p'}{C \xrightarrow{\{(\sigma,I)\}}_s p'}\ C \doteq p$$

$$(\mathrm{S\text{-}Pref}^s_1)\ \frac{p \xrightarrow{\{(\sigma,I)\}}_s p'}{\underline{a}.p \xrightarrow{\{(a\diamond\sigma,I)\}}_s p'} \qquad (\mathrm{S\text{-}Pref}^s_2)\ \frac{p \xrightarrow{\{(\sigma,I)\}}_s p'}{\underline{\neg D}.p \xrightarrow{\{(\sigma,I\cup\{D\})\}}_s p'}$$

$$(\mathrm{S\text{-}Res}^s)\ \frac{p \xrightarrow{M}_s p'}{(\nu a)p \xrightarrow{M\{a'/a\}}_s (\nu a)p'}\ a \notin n(M) \qquad (\mathrm{Par}^s)\ \frac{p \xrightarrow{M}_s p' \quad nct(M,q)}{p\,|\,q \xrightarrow{M}_s p'\,|\,q}$$

$$(\mathrm{M\text{-}Par}^s)\ \frac{p \xrightarrow{M_1}_s p' \quad q \xrightarrow{M_2}_s q' \quad nct(M_1,q), nct(M_1,q') \quad nct(M_2,p), nct(M_2,p')}{p\,|\,q \xrightarrow{M_1 \oplus M_2}_s p'\,|\,q'}$$

$$(\mathrm{S\text{-}Com}^s)\ \frac{p \xrightarrow{\{(\sigma_1,I)\}}_s p' \qquad q \xrightarrow{\{(\overline{a},\emptyset)\}}_s q' \quad q \in \mathscr{P}^{seq}_{NPL}}{p\,|\,q \xrightarrow{\{(\sigma,I)\}}_s p'\,|\,q'}\ Sync(\sigma_1,\overline{a},\sigma)$$

$$(\mathrm{Abs}^s)\ \frac{p \xrightarrow{M}_s p'}{p \overset{a(M)}{\longmapsto}_s p'}\ a(M) \neq \emptyset$$

Table 8.5 Step operational rules for NPL

introduce active constants in conflict with the corresponding multisets — $nct(M_1,q')$ and $nct(M_2,p')$. These four checks are necessary in order to prove that whenever such a parallel step is performed, then the two composing steps can be performed in either order by means of rule (Par^s); in fact, $p\,|\,q \xrightarrow{M_1}_s p'\,|\,q \xrightarrow{M_2}_s p'\,|\,q'$ is possible if $nct(M_1,q)$ and $nct(M_2,p')$, while $p\,|\,q \xrightarrow{M_2}_s p\,|\,q' \xrightarrow{M_1}_s p'\,|\,q'$ is possible if $nct(M_2,p)$ and $nct(M_1,q')$.

Rule (S-Com^s) requires that q be a sequential process performing a singleton output transition $(\overline{a},\emptyset)$, and p perform a singleton step containing a synchronizable sequence (σ_1,I). The result is the step where (σ_1,I) is replaced by (σ,I), σ being the result of synchronizing σ_1 with $\overline{a}$. The case when multiple synchronizations are to be performed in one step is covered by means of a combination of rules (Cong^s) and (M-Par^s), as described in Example 8.6.

The *abstract* step operational semantics is given by the STS $\mathscr{C}^{a,step}_{NPL} = (\mathscr{P}_{NPL}, \mathscr{M}_{fin}(\mathscr{A}), \longmapsto_s)$, where an *abstract* step label is a finite multiset of abstract interleaving labels in $\mathscr{A} = \{\tau\} \cup \overline{\mathscr{L}} \cup \mathscr{L}^+$ and $\longmapsto_s \subseteq \mathscr{P}_{NPL} \times \mathscr{M}_{fin}(\mathscr{A}) \times \mathscr{P}_{NPL}$ is the least transition relation generated by rule (Abs^s) in Table 8.5. This rule requires that the abstract content of a concrete step M — denoted by $a(M)$ — be nonempty. The multiset $a(M)$ is computed as follows: $a(\emptyset) = \emptyset$, $a(M \oplus \{(D,\emptyset)\}) = a(M)$,

$$
\text{(S-Com}^s\text{)}\;\dfrac{
\text{(Cons}^s_2\text{)}\;\dfrac{
\text{(S-Pref}^s_2\text{)}\;\dfrac{
\text{(S-Pref}^s_1\text{)}\;\dfrac{
\text{(Pref}^s\text{)}\;\dfrac{}{a.p \xrightarrow{\{(a,\emptyset)\}}_s p}
}{\underline{c}.a.p \xrightarrow{\{(ca,\emptyset)\}}_s p}
}{\underline{\neg D}.\underline{c}.a.p \xrightarrow{\{(ca,\{D\})\}}_s p}
}{A \xrightarrow{\{(ca,\{D\})\}}_s p}
\qquad
\text{(Cons}^s_2\text{)}\;\dfrac{
\text{(Pref}^s\text{)}\;\dfrac{}{\overline{c}.q \xrightarrow{\{(\overline{c},\emptyset)\}}_s q}
}{B \xrightarrow{\{(\overline{c},\emptyset)\}}_s q}
}{A\,|\,B \xrightarrow{\{(a,\{D\})}_s p\,|\,q}
$$

$$
\text{(Abs}^s\text{)}\;\dfrac{
\text{(Cong)}\;\dfrac{
\text{(M-Par}^s\text{)}\;\dfrac{
\dfrac{}{A\,|\,B \xrightarrow{\{(a,\{D\})\}}_s p\,|\,q}
\qquad
\dfrac{}{A\,|\,B \xrightarrow{\{(a,\{D\})\}}_s p\,|\,q}
}{(A\,|\,A)\,|\,(B\,|\,B) \equiv (A\,|\,B)\,|\,(A\,|\,B) \xrightarrow{\{(a,\{D\}),(a,\{D\})\}}_s (p\,|\,q)\,|\,(p\,|\,q) \equiv (p\,|\,p)\,|\,(q\,|\,q)}
}{(A\,|\,A)\,|\,(B\,|\,B) \xrightarrow{\{(a,\{D\}),(a,\{D\})\}}_s (p\,|\,p)\,|\,(q\,|\,q)}
}{(A\,|\,A)\,|\,(B\,|\,B) \overset{\{a,a\}}{\longmapsto}_s (p\,|\,p)\,|\,(q\,|\,q)}
$$

Table 8.6 The proof of a step transition

$a(M \oplus \{(\sigma, I)\}) = a(M) \oplus \{\sigma\}$ if $\sigma \notin \mathscr{T}cons$. This means that, first, all idling transitions of testable constants occurring in the step M are removed and then, if the result is not an empty multiset, the step transition is imported into the abstract STS by removing all the second components from the concrete labels. The abstract STS makes sense only for observing *fixed* processes, i.e., processes not composable with any other process, working in isolation with respect to the external environment. In this case, the set of constant names in the labels is no longer necessary and can be safely abstracted away.

Example 8.6. Let us consider the NPL process $(A\,|\,A)\,|\,(B\,|\,B)$, where the constants A and B are defined as follows:

$$A \doteq \underline{\neg D}.\underline{c}.a.p, \qquad\qquad B \doteq \overline{c}.q,$$

with p and q not specified further. Table 8.6 shows the proof of the abstract step transition $(A\,|\,A)\,|\,(B\,|\,B) \overset{\{a,a\}}{\longmapsto}_s (p\,|\,p)\,|\,(q\,|\,q)$, which can be derived only thanks to rules (Cong^s) and (M-Par^s). □

A couple of properties of the step semantics are listed below. The first one is that the concrete STS for NPL is *fully concurrent* (Definition 2.25). The second one justifies the condition on the step operational rule (S-Com^s) that the process performing the output be sequential.

Lemma 8.2. *For any restriction-free NPL process p, if $p \xrightarrow{M_1 \oplus M_2}_s p'$ and $M_i \neq \emptyset$ for $i = 1, 2$, then there exist p_1 and p_2 such that $p \equiv p_1\,|\,p_2$, $p' \equiv p'_1\,|\,p'_2$, $p_1 \xrightarrow{M_1}_s p'_1$, $p_2 \xrightarrow{M_2}_s p'_2$, $nct(M_1, p_2)$, $nct(M_1, p'_2)$, $nct(M_2, p_1)$ and $nct(M_2, p'_1)$.*

Proof. By induction on the proof of $p \xrightarrow{M_1 \oplus M_2}_s p'$ and on the size of $M_1 \oplus M_2$. Since $M_i \neq \emptyset$ for $i = 1,2$, the least $k = |M_1 \oplus M_2|$ is 2. Only rule (M-Par^s) can be used to generate a label of size 2, with premises labeled by singletons. Therefore, in this base case of the induction, we have that $p = p_1 \,|\, p_2$, $p_1 \xrightarrow{M_1}_s p_1'$, $p_2 \xrightarrow{M_2}_s p_2'$, $p' = p_1' \,|\, p_2'$, $nct(M_1, p_2)$, $nct(M_1, p_2')$, $nct(M_2, p_1)$ and $nct(M_2, p_1')$, as required.

The inductive cases are the following. If rule (Cong^s) is the last rule used in deriving $p \xrightarrow{M_1 \oplus M_2}_s p'$, then $p \equiv q$, $p' \equiv q'$ and the premise transition is $q \xrightarrow{M_1 \oplus M_2}_s q'$; by induction, we can assume that $q \equiv q_1 \,|\, q_2$, $q' \equiv q_1' \,|\, q_2'$, $q_1 \xrightarrow{M_1}_s q_1'$, $q_2 \xrightarrow{M_2}_s q_2'$, $nct(M_1, q_2)$, $nct(M_1, q_2')$, $nct(M_2, q_1)$ and $nct(M_2, q_1')$. Hence, the thesis follows trivially, as $p \equiv q_1 \,|\, q_2$ and $p' \equiv q_1' \,|\, q_2'$.

If rule (Par^s) is the last rule used in deriving $p \xrightarrow{M_1 \oplus M_2}_s p'$, then $p = r \,|\, q$, $r \xrightarrow{M_1 \oplus M_2}_s r'$ and $nct(M_1 \oplus M_2, q)$. By induction, there exist r_1 and r_2 such that $r \equiv r_1 \,|\, r_2$, $r' \equiv r_1' \,|\, r_2'$, $r_1 \xrightarrow{M_1}_s r_1'$, $r_2 \xrightarrow{M_2}_s r_2'$, $nct(M_1, r_2)$, $nct(M_1, r_2')$, $nct(M_2, r_1)$ and $nct(M_2, r_1')$. By rule (Par^s), also $r_2 \,|\, q \xrightarrow{M_2}_s r_2' \,|\, q$ is derivable, because $nct(M_2, q)$ holds, given $nct(M_1 \oplus M_2, q)$.[3] Therefore, the thesis follows by taking $p_1 = r_1$ and $p_2 = r_2 \,|\, q$, because $nct(M_1, r_2 \,|\, q)$ holds, given $nct(M_1 \oplus M_2, q)$ and $nct(M_1, r_2)$,[4] and similarly for $nct(M_1, r_2' \,|\, q)$.

Rule (M-Par^s) can also be used with premises that are not all labeled with singletons. However, the proof is the same as for the base case above. □

Theorem 8.2. (Fully concurrent STS) *For any NPL process p, if $p \xrightarrow{M_1 \oplus M_2}_s p'$ and $M_i \neq \emptyset$ for $i = 1,2$, then there exist q_1 and q_2 such that $p \xrightarrow{M_1}_s q_1 \xrightarrow{M_2}_s p'$ and $p \xrightarrow{M_2}_s q_2 \xrightarrow{M_1}_s p'$.*

Proof. If p is restriction-free, then by Lemma 8.2 there exist p_1 and p_2 such that $p \equiv p_1 \,|\, p_2$, $p' \equiv p_1' \,|\, p_2'$, $p_1 \xrightarrow{M_1}_s p_1'$, $p_2 \xrightarrow{M_2}_s p_2'$, $nct(M_1, p_2)$, $nct(M_1, p_2')$, $nct(M_2, p_1)$ and $nct(M_2, p_1')$. Therefore, by rule (Par^s), all the transitions $p_1 \,|\, p_2 \xrightarrow{M_1}_s p_1' \,|\, p_2$, $p_1' \,|\, p_2 \xrightarrow{M_2}_s p_1' \,|\, p_2'$, $p_1 \,|\, p_2 \xrightarrow{M_2}_s p_1 \,|\, p_2'$ and $p_1 \,|\, p_2' \xrightarrow{M_1}_s p_1' \,|\, p_2'$ are derivable. By rule (Cong^s), also $p \xrightarrow{M_1}_s p_1' \,|\, p_2$, $p_1' \,|\, p_2 \xrightarrow{M_2}_s p'$, $p \xrightarrow{M_2}_s p_1 \,|\, p_2'$ and $p_1 \,|\, p_2' \xrightarrow{M_1}_s p'$, so that the thesis follows by taking $q_1 = p_1' \,|\, p_2$ and $q_2 = p_1 \,|\, p_2'$.

If $p = (\nu L)t$, where t is restriction-free, then transition $p \xrightarrow{M_1 \oplus M_2}_s p'$ is derivable only if, by rule (S-Res^s), $M_1 \oplus M_2 = (M_1' \oplus M_2')\{L'/L\}$, $t \xrightarrow{M_1' \oplus M_2'}_s t'$ and $p' = (\nu L)t'$. By Lemma 8.2, there exist t_1 and t_2 such that $t \equiv t_1 \,|\, t_2$, $t' \equiv t_1' \,|\, t_2'$, $t_1 \xrightarrow{M_1'}_s t_1'$, $t_2 \xrightarrow{M_2'}_s t_2'$, $nct(M_1', t_2)$, $nct(M_1', t_2')$, $nct(M_2', t_1)$ and $nct(M_2', t_1')$. Therefore, by rules (S-Res^s) and (Cong^s), the transitions $p \xrightarrow{M_1}_s (\nu L)(t_1' \,|\, t_2)$, $(\nu L)(t_1' \,|\, t_2) \xrightarrow{M_2}_s p'$, $p \xrightarrow{M_2}_s (\nu L)(t_1 \,|\, t_2')$ and $(\nu L)(t_1 \,|\, t_2') \xrightarrow{M_1}_s p'$ are all derivable. Hence, the thesis follows by taking $q_1 = (\nu L)(t_1' \,|\, t_2)$ and $q_2 = (\nu L)(t_1 \,|\, t_2')$. □

[3] Note that if $nct(N, r)$ holds and $N' \subseteq N$, then also $nct(N', r)$ holds.

[4] Note that if $nct(N, t_1)$ and $nct(N, t_2)$ hold, then also $nct(N, t_1 \,|\, t_2)$ holds.

Lemma 8.3. *For any restriction-free NPL process p, if $p \xrightarrow{M\oplus\{(\overline{a},\emptyset)\}}_s p'$, then either $p \in \mathscr{P}_{NPL}^{seq}$ (and so $M = \emptyset$), or there exists $q \in \mathscr{P}_{NPL}^{seq}$ such that $p \equiv q \,|\, t$ and $q \xrightarrow{\{(\overline{a},\emptyset)\}}_s q'$; and, if $M = \emptyset$, then $p' \equiv q' \,|\, t$, else $t \xrightarrow{M}_s t'$, with $nct(M,q)$, $nct(M,q')$ and $p' \equiv q' \,|\, t'$.*

Proof. By induction on the proof of $p \xrightarrow{M\oplus\{(\overline{a},\emptyset)\}}_s p'$. The proof is similar to the analogous Proposition 8.3 for the concrete interleaving LTS, and so we omit the cases about the rules for sequential processes or when $M = \emptyset$. If $M \neq \emptyset$, the last rule used in deriving $p \xrightarrow{M\oplus\{(\overline{a},\emptyset)\}}_s p'$ may be (M-Par^s), or (Par^s), or (Cong^s).

If (M-Par^s) was used, then $p = p_1 \,|\, p_2$, $M \oplus \{(\overline{a},\emptyset)\} = M_1 \oplus M_2$, $p_1 \xrightarrow{M_1}_s p_1'$, $p_2 \xrightarrow{M_2}_s p_2'$, $nct(M_1,p_2)$, $nct(M_1,p_2')$, $nct(M_2,p_1)$ and $nct(M_2,p_1')$ hold. Of course, $(\overline{a},\emptyset)$ must belong to M_1 or M_2; w.l.o.g., assume that $M_1 = M_1' \oplus \{(\overline{a},\emptyset)\}$. By Lemma 8.2, there exist r_1 and r_2 such that $p_1 \equiv r_1 \,|\, r_2$, $r_1 \xrightarrow{\{(\overline{a},\emptyset)\}}_s r_1'$, $r_2 \xrightarrow{M_1'}_s r_2'$, $nct(\{(\overline{a},\emptyset)\},r_2)$, $nct(\{(\overline{a},\emptyset)\},r_2')$,[5] $nct(M_1',r_1)$ and $nct(M_1',r_1')$. By induction, either r_1 is sequential; or there exists a sequential q_1 such that $r_1 \equiv q_1 \,|\, t_1$, $q_1 \xrightarrow{\{(\overline{a},\emptyset)\}}_s q_1'$ and $p_1' \equiv q_1' \,|\, t_1$. In the former case, the thesis follows by taking $q = r_1$ and $t = r_2 \,|\, p_2$. In fact, by rule (M-Par^s), $r_2 \,|\, p_2 \xrightarrow{M_1'\oplus M_2}_s r_2' \,|\, p_2'$, because $nct(M_1',p_2)$ ($nct(M_1',p_2')$) holds, given $nct(M_1,p_2)$ ($nct(M_1,p_2')$), and $nct(M_2,r_2)$ ($nct(M_2,r_2')$) holds, given $nct(M_2,p_1)$ ($nct(M_2,p_1')$).[6] Then, by rule (M-Par^s), $r_1 \,|\, (r_2 \,|\, p_2) \xrightarrow{M_1\oplus M_2}_s r_1' \,|\, (r_2' \,|\, p_2')$, because $nct(\{(\overline{a},\emptyset)\},r_2 \,|\, p_2)$, $nct(\{(\overline{a},\emptyset)\},r_2' \,|\, p_2')$, $nct(M_1' \oplus M_2,r_1)$ and $nct(M_1' \oplus M_2,r_1')$, given $nct(M_1',r_1)$ ($nct(M_1',r_1')$) and $nct(M_2,p_1)$ ($nct(M_2,p_1')$).[7] In the latter case, the thesis follows by taking $q = q_1$ and $t = (t_1 \,|\, r_2) \,|\, p_2$. In fact, by rule (Cong^s) and (Par^s), $t_1 \,|\, r_2 \xrightarrow{M_1'}_s t_1 \,|\, r_2'$, because $nct(M_1',t_1)$ holds, given $nct(M_1',r_1)$. Then, by rule (M-Par^s), $(t_1 \,|\, r_2) \,|\, p_2 \xrightarrow{M_1'\oplus M_2}_s (t_1 \,|\, r_2') \,|\, p_2'$ is derivable because $nct(M_1',p_2)$ ($nct(M_1',p_2')$), given $nct(M_1,p_2)$ ($nct(M_1,p_2')$), and $nct(M_2,t_1 \,|\, r_2)$ ($nct(M_2,t_1 \,|\, r_2')$), given $nct(M_2,p_1)$ ($nct(M_2,p_1')$). Finally, $q_1 \,|\, ((t_1 \,|\, r_2) \,|\, p_2) \xrightarrow{M_1\oplus M_2}_s q_1' \,|\, ((t_1 \,|\, r_2') \,|\, p_2')$ is derivable by (M-Par^s), because $nct(\{(\overline{a},\emptyset)\},(t_1 \,|\, r_2) \,|\, p_2)$, $nct(\{(\overline{a},\emptyset)\},(t_1 \,|\, r_2') \,|\, p_2)$, $nct(M_1' \oplus M_2,q_1)$ and $nct(M_1' \oplus M_2,q_1')$, the last two given $nct(M_1',r_1)$ ($nct(M_1',r_1')$) and $nct(M_2,p_1)$ ($nct(M_2,p_1')$).

If rule (Par^s) was used, then $p = p_1 \,|\, p_2$, $p_1 \xrightarrow{M\oplus\{(\overline{a},\emptyset)\}}_s p_1'$, and $nct(M \oplus \{(\overline{a},\emptyset)\},p_2)$ holds. By Lemma 8.2, there exist r_1 and r_2 such that $p_1 \equiv r_1 | r_2$, $p_1' \equiv r_1' | r_2'$, $r_1 \xrightarrow{\{(\overline{a},\emptyset)\}}_s r_1'$, $r_2 \xrightarrow{M}_s r_2'$, $nct(\{(\overline{a},\emptyset)\},r_2)$, $nct(\{(\overline{a},\emptyset)\},r_2')$, $nct(M,r_1)$ and $nct(M,r_1')$. By induction, either r_1 is sequential; or there exists a sequential q_1 such that $r_1 \equiv q_1 \,|\, t_1$, $q_1 \xrightarrow{\{(\overline{a},\emptyset)\}}_s q_1'$, with $r_1' \equiv q_1' \,|\, t_1$. In the former case, the thesis follows trivially by taking $q = r_1$ and $t = r_2 \,|\, p_2$. In fact, by rule (Par^s), $r_2 \,|\, p_2 \xrightarrow{M}_s r_2' \,|\, p_2$, because $nct(M,p_2)$ holds, given $nct(M \oplus \{(\overline{a},\emptyset)\},p_2)$. Then, by rule (M-Par^s), transition

[5] Note that $nct(N,r)$ holds if $I(N) = \emptyset$; hence, $nct(\{(\overline{a},\emptyset)\},r)$ holds vacuously for any r.

[6] Note that if $nct(N,r)$ holds for a parallel term r and $r \equiv r' \,|\, t$, then also $nct(N,r')$ holds.

[7] Note that $nct(N \oplus N',r)$ holds if and only if $nct(N,r)$ and $nct(N',r)$ hold.

$r_1\,|\,(r_2\,|\,p_2) \xrightarrow{M\oplus\{(\overline{a},\emptyset)\}}_s r'_1\,|\,(r'_2\,|\,p_2)$, because $nct(\{(\overline{a},\emptyset)\}, r_2\,|\,p_2)$, $nct(\{(\overline{a},\emptyset)\}, r'_2\,|\,p_2)$, $nct(M,r_1)$ and $nct(M,r'_1)$ hold. In the latter case, the thesis follows by taking $q = q_1$ and $t = t_1\,|\,(r_2\,|\,p_2)$. In fact, by rule (Par^s), $r_2\,|\,p_2 \xrightarrow{M}_s r'_2\,|\,p_2$ is derivable, as explained above; then, by rules (Par^s) and (Cong^s), $t_1\,|\,(r_2\,|\,p_2) \xrightarrow{M}_s t_1\,|\,(r'_2\,|\,p_2)$ is derivable, because $nct(M,t_1)$ holds, given $nct(M,r_1)$; finally, by rule (M-Par^s), transition $q_1\,|\,(t_1\,|\,(r_2\,|\,p_2)) \xrightarrow{M\oplus\{(\overline{a},\emptyset)\}}_s q'_1\,|\,(t_1\,|\,(r'_2\,|\,p_2))$ is derivable, because $nct(\{(\overline{a},\emptyset)\}$, $t_1\,|\,(r_2\,|\,p_2))$, $nct(\{(\overline{a},\emptyset)\}, t_1\,|\,(r'_2\,|\,p_2))$, $nct(M,q_1)$ and $nct(M,q'_1)$ hold, given $nct(M$, $r_1)$ and $nct(M,r'_1)$.

If rule (Cong^s) was used, then $p \equiv r$, $p' \equiv r'$ and $r \xrightarrow{M\oplus\{(\overline{a},\emptyset)\}}_s r'$ is the premise transition. By induction, there exists a sequential q such that $r \equiv q\,|\,t$, $q \xrightarrow{\{(\overline{a},\emptyset)\}}_s q'$ and $t \xrightarrow{M}_s t'$, with $nct(M,q)$, $nct(M,q')$ and $r' \equiv q'\,|\,t'$, as required. □

Definition 8.2. Two NPL processes p and q are *concrete step bisimilar*, denoted $p \sim^c_{step} q$, if there exists a bisimulation $R \subseteq \mathscr{P}_{NPL} \times \mathscr{P}_{NPL}$ over the concrete step transition system $\mathscr{C}^{c,step}_{NPL}$ such that $(p,q) \in R$. □

Proposition 8.10. *Let $p,q \in \mathscr{P}_{NPL}$ be two NPL processes. If $p \equiv q$, then $p \sim^c_{step} q$.*

Proof. *It is enough to check that relation $R = \{(p,q) \;\big|\; p \equiv q\}$ is a bisimulation.* □

As expected, concrete step bisimilarity is finer than concrete interleaving bisimilarity.

Lemma 8.4. *For any $p \in \mathscr{P}_{NPL}$, $p \xrightarrow{(\sigma,I)} p'$ if and only if $p \xrightarrow{\{(\sigma,I)\}}_s p'$.*

Proof. *In one direction, by induction on the proof of transition $p \xrightarrow{(\sigma,I)} p'$, we prove that also $p \xrightarrow{\{(\sigma,I)\}}_s p'$ is derivable. This is trivially true as a proof of the former transition can be turned into the proof of the latter transition by using the corresponding step rules with superscript s.*

In the other direction, by induction on the proof of transition $p \xrightarrow{\{(\sigma,I)\}}_s p'$, we prove that also $p \xrightarrow{(\sigma,I)} p'$ is derivable. Note that (M-Par^s) is the only rule that cannot be used in deriving $p \xrightarrow{\{(\sigma,I)\}}_s p'$; therefore, as hinted above, the proof of the former transition can be turned into the proof of the latter transition by simply omitting the superscript s. □

Proposition 8.11. *If $p_1 \sim^c_{step} p_2$, then $p_1 \sim^c_{int} p_2$.*

Proof. *Let R be a concrete step bisimulation such that (p_1,p_2). We prove that R is also a concrete interleaving bisimulation. If $p \xrightarrow{(\sigma,I)} p'$, then by Lemma 8.4, also $p \xrightarrow{\{(\sigma,I)\}}_s p'$ is derivable. Since R is a concrete step bisimulation, then there exists q' such that $q \xrightarrow{\{(\sigma,I)\}}_s q'$, with $(p',q') \in R$. Again, by Lemma 8.4, we have $q \xrightarrow{(\sigma,I)} q'$, as required. The case when q moves first is analogous and so omitted.* □

The following example shows some slightly surprising phenomena of NPL processes w.r.t. concrete step semantics.

Example 8.7. Let us consider the parallel process $p = \overline{b}.D \,|\, \underline{\neg D}.a.\mathbf{0}$, where $D \doteq \mathbf{0}$. It is not difficult to realize that $p \sim_{step} q$, where $q = \overline{b}.D + \underline{\neg D}.a.\overline{b}.D$ is a sequential process. This example shows that a nonsequential, restriction-free NPL process whose sequential subprocesses are not equivalent to $\mathbf{0}$ can be concrete step bisimilar to a sequential NPL process. This is not possible for FNM.

Now, let us consider the parallel process $r = D' \,|\, \underline{\neg D'}.a.b.\mathbf{0}$, where $D' \doteq \overline{a}.\mathbf{0}$. Even if all the finitely many (concrete) steps that r can perform are singletons, nonetheless no sequential process s is concrete step equivalent to r (since D' cannot be defined differently for r and s). Also this phenomenon cannot happen for FNM. □

Definition 8.3. Two NPL processes p and q are *abstract step bisimilar*, denoted $p \sim^{a}_{step} q$, if there exists a bisimulation $R \subseteq \mathscr{P}_{NPL} \times \mathscr{P}_{NPL}$ over the abstract step transition system $\mathscr{C}^{a,step}_{NPL}$, such that $(p,q) \in R$. □

Along the same lines as Proposition 8.11, we can prove that abstract step bisimilarity $\sim^{a}_{step}$ is finer than abstract interleaving bisimilarity $\sim^{a}_{int}$. Abstract step bisimilarity is not a congruence for parallel composition, because it forgets useful information. For instance, consider again $C \doteq \mathbf{0}$ and $D \doteq \mathbf{0}$, where only D is testable; clearly, $C \sim^{a}_{step} D$, as the two are deadlock states in the abstract STS; nonetheless, if we consider the constant $E \doteq \underline{\neg D}.a.\mathbf{0}$, the two processes $C \,|\, E$ and $D \,|\, E$ are not equivalent in the abstract STS, as the former can do $\{a\}$, while the latter cannot.

On the contrary, we conjecture that concrete step bisimilarity $\sim^{c}_{step}$ is a congruence for the parallel composition operator. To support our congruence claim, we give the proof in a simple, restricted case when the involved processes are sequential.

Lemma 8.5. *Given $s \in \mathscr{P}^{seq}_{NPL}$ and p a nonsequential, restriction-free NPL process such that $s \sim^{c}_{step} p$, if $p \xrightarrow{\{(\overline{a},\emptyset)\}}_{s} p'$, then $p \equiv q \,|\, t$, with $q \in \mathscr{P}^{seq}_{NPL}$, such that $q \xrightarrow{\{(\overline{a},\emptyset)\}}_{s} q'$, $p' \equiv q' \,|\, t$ and $t \overset{\{(D,\emptyset)\}}{\nrightarrow}_{s}$ for all $D \in \mathscr{T}cons$.*

Proof. By Lemma 8.3, there exists $q \in \mathscr{P}^{seq}_{NPL}$ such that $p \equiv q \,|\, t$, $q \xrightarrow{\{(\overline{a},\emptyset)\}}_{s} q'$, and $p' \equiv q' \,|\, t$. The additional constraint that t cannot exhibit testable constant names is satisfied because $p \sim^{c}_{step} s$ and s, being sequential, cannot exhibit the parallel step $\{(\overline{a},\emptyset),(D,\emptyset)\}$ for any $D \in \mathscr{T}cons$. □

Proposition 8.12. *Given $p_1, q_1 \in \mathscr{P}^{seq}_{NPL}$ and p_2, q_2 restriction-free, if $p_1 \sim^{c}_{step} p_2$ and $q_1 \sim^{c}_{step} q_2$, then $p_1 \,|\, q_1 \sim^{c}_{step} p_2 \,|\, q_2$.*

Proof. Assume we have concrete step bisimulation relations R_1 and R_2 such that $(p_1, p_2) \in R_1$ and $(q_1, q_2) \in R_2$. Let us consider the relation

$R = \{(p'_1 \,|\, q'_1, p'_2 \,|\, q'_2) \;\big|\; (p'_1, p'_2) \in R_1 \wedge (q'_1, q'_2) \in R_2\}$.

We want to prove that R is a concrete step bisimulation up to $\equiv$ that contains the pair $(p_1 \,|\, q_1, p_2 \,|\, q_2)$, as required. We proceed by case analysis on the possible moves of $p_1 \,|\, q_1$:

1. *$p_1 \mid q_1 \xrightarrow{M}_s p_1' \mid q_1$ because, by rule (Par^s), $p_1 \xrightarrow{M}_s p_1'$ and $nct(M, q_1)$ holds. In this case, as $p_1 \sim_{step} p_2$, also $p_2 \xrightarrow{M}_s p_2'$ is derivable, with $(p_1', p_2') \in R_1$. Since $q_1 \sim_{step} q_2$, also $nct(M, q_2)$ holds. Therefore, by rule (Par^s), also $p_2 \mid q_2 \xrightarrow{M}_s p_2' \mid q_2$ is derivable, with $(p_1' \mid q_1, p_2' \mid q_2) \in R$, as required.*
2. *$p_1 \mid q_1 \xrightarrow{M}_s p_1 \mid q_1'$ because, by rules (Cong^s) and (Par^s), $q_1 \xrightarrow{M}_s q_1'$ and $nct(M, p_1)$ holds. Analogous to the above, and so omitted.*
3. *$p_1 \mid q_1 \xrightarrow{M_1 \oplus M_2}_s p_1' \mid q_1'$ because, by rule (M-Par^s), $p_1 \xrightarrow{M_1}_s p_1'$, $q_1 \xrightarrow{M_2}_s q_1'$, $nct(M_1, q_1)$, $nct(M_1, q_1')$, $nct(M_2, p_1)$ and $nct(M_2, p_1')$ hold. In this case, as $p_1 \sim_{step} p_2$, also $p_2 \xrightarrow{M_1}_s p_2'$ is derivable, with $(p_1', p_2') \in R_1$; as a consequence, $nct(M_2, p_2)$ and $nct(M_2, p_2')$ hold. Since $q_1 \sim_{step} q_2$, also transition $q_2 \xrightarrow{M_2}_s q_2'$ is derivable, with $(q_1', q_2') \in R_2$; as a consequence, $nct(M_1, q_2)$ and $nct(M_1, q_2')$ hold. Therefore, by rule (M-Par^s), $p_2 \mid q_2 \xrightarrow{M_1 \oplus M_2}_s p_2' \mid q_2'$, with $(p_1' \mid q_1', p_2' \mid q_2') \in R$, as required.*
4. *$p_1 \mid q_1 \xrightarrow{\{(\sigma, I)\}}_s p_1' \mid q_1'$ because, by rule (S-Com^s), $p_1 \xrightarrow{\{(\sigma_1, I)\}}_s p_1'$, $q_1 \xrightarrow{\{(\overline{a}, \emptyset)\}}_s q_1'$ and, moreover, $Sync(\sigma_1, \overline{a}, \sigma)$ holds. In this case, as $p_1 \sim_{step} p_2$, also $p_2 \xrightarrow{\{(\sigma_1, I)\}}_s p_2'$ is derivable, with $(p_1', p_2') \in R_1$. Since $q_1 \sim_{step} q_2$, also $q_2 \xrightarrow{\{(\overline{a}, \emptyset)\}}_s q_2'$ is derivable, with $(q_1', q_2') \in R_2$. If q_2 is sequential, then $p_2 \mid q_2 \xrightarrow{\{(\sigma, I)\}}_s p_2' \mid q_2'$ by rule (S-Com^s), with $(p_1' \mid q_1', p_2' \mid q_2') \in R$, as required. If q_2 is not sequential, by Lemma 8.5, $q_2 \equiv r_2 \mid t_2$ with r_2 sequential, such that $r_2 \xrightarrow{\{(\overline{a}, \emptyset)\}}_s r_2'$, $q_2' \equiv r_2' \mid t_2$ and $t_2 \stackrel{\{(D;\emptyset)\}}{\not\longrightarrow}_s$ for any testable D. Therefore, by rule (S-Com^s), $p_2 \mid r_2 \xrightarrow{\{(\sigma, I)\}}_s p_2' \mid r_2'$ is derivable; then, by rule (Par^s), $(p_2 \mid r_2) \mid t_2 \xrightarrow{\{(\sigma, I)\}}_s (p_2' \mid r_2') \mid t_2$ is derivable; finally, by rule (Cong^s), $p_2 \mid q_2 \xrightarrow{\{(\sigma, I)\}}_s p_2' \mid q_2'$ with $(p_1' \mid q_1', p_2' \mid q_2') \in R$, as required.*
5. *$p_1 \mid q_1 \xrightarrow{\{(\sigma, I)\}}_s p_1' \mid q_1'$ because $p_1 \xrightarrow{\{(\overline{a}, \emptyset)\}}_s p_1'$, $q_1 \xrightarrow{\{(\sigma_1, I)\}}_s q_1'$ and $Sync(\sigma_1, \overline{a}, \sigma)$ holds, by rules (Cong^s) and (S-Com^s). This case is analogous to the previous one, and so omitted.*

No other move is possible for $p_1 \mid q_1$, except for possible moves due to the additional use of rule (Cong^s), each one reaching a state that is structurally congruent to one of the above.

The cases when $p_2 \mid q_2$ moves first are similar, where, in places, Lemma 8.5 is to be used, as done in item 4 *above.* □

8.4 Operational Net Semantics

In this section, we describe a technique for building a Nonpermissive Petri net (NP/T net, for short) for the whole of NPL, starting from a description of its places and its net transitions. The resulting net $N_{NPL} = (S_{NPL}, D_{NPL}, \mathscr{A}, T_{NPL})$ is such that, for any $p \in \mathscr{P}_{NPL}$, the net system $N_{NPL}(dec(p))$ statically reachable from the initial marking $dec(p)$, denoted by $Net(p)$, is a statically reduced, finite NP/T net.

8.4.1 Places and Markings

The infinite set of NPL places, ranged over by s, is $S_{NPL} = \mathscr{P}_{NPL}^{\gamma,seq} \setminus \{\mathbf{0}\}$, i.e., the set of all sequential, *extended* NPL processes, except $\mathbf{0}$, while the set D_{NPL} of testable places is $\mathscr{T}cons$, i.e., the set of testable constants; note that $D_{NPL} \subseteq S_{NPL}$.

Function $dec : \mathscr{P}_{NPL}^{\gamma} \to \mathscr{M}_{fin}(S_{NPL})$, which defines the decomposition of extended processes into markings, was outlined in Table 6.8 for the FNC operators. It is extended to the strong prefixing operator as follows:

$$dec(\underline{\gamma}.p, I) = \{\underline{\gamma}.p\} \quad \text{and} \quad dec(\underline{\neg D}.p, I) = \{\underline{\neg D}.p\}.$$

Hence, the additional negated strong prefixes are treated like FNM strong prefixes, so that all the results we proved for FNM markings can be easily adapted to the slightly more general NPL markings.

Lemma 8.6. *For any restriction-free, extended NPL process t, $dec(t)\{L'/L\} = dec(t\{L'/L\})$.*

Proof. *By induction on the definition of $dec(t)$.* □

Proposition 8.13. *For any restriction-free $t \in \mathscr{P}_{NPL}$, $dec((\nu L)t) = dec(i((\nu L)t)) = dec(t\{L'/L\})$.*

Proof. *By definition, $dec((\nu L)t) = dec(t)\{L'/L\}$. Then, by Lemma 8.6 $dec(t)\{L'/L\} = dec(t\{L'/L\})$.* □

This means that we can restrict our attention to extended, restriction-free processes, as a restricted process $(\nu L)t$ in $\mathscr{P}_{NPL}$ is mapped via *dec* to the same marking associated with $i((\nu L)t) = t\{L'/L\}$ in $\mathscr{P}_{NPL}^{\gamma,par}$.

Proposition 8.14. *For any $p \in \mathscr{P}_{NPL}^{\gamma}$, $dec(p)$ is a finite multiset of places.*

Proof. *By induction on the definition of $dec(p)$.* □

Of course, function *dec* is not injective; however, one can prove that function *dec* is surjective over *admissible* markings, as done for FNM. We recall that a marking $m \in \mathscr{M}_{fin}(S_{NPL})$ is *complete* if there is an NPL process $p \in \mathscr{P}_{NPL}$ such that $dec(p) = m$.

Lemma 8.7. *For any $t \in \mathscr{P}_{NPL}^{\gamma,par}$, $fn(t) = fn(dec(t))$.*

Proof. *By induction on the definition of $dec(t)$.* □

Theorem 8.3. *A marking $m \in \mathscr{M}_{fin}(S_{NPL})$ is admissible iff it is complete.*

Proof. *Similar to that for the analogous Theorem 6.4, by using Proposition 8.13 and Lemma 8.7.* □

$$\text{(pref)}\ \frac{}{(\{\mu.p\},\emptyset) \xrightarrow{\mu} dec(p)} \qquad \text{(cons)}\ \frac{(dec(p),I) \xrightarrow{\sigma} m}{(\{C\},I) \xrightarrow{\sigma} m}\ C \doteq p$$

$$\text{(sum}_1\text{)}\ \frac{(dec(p),I) \xrightarrow{\sigma} m}{(\{p+q\},I) \xrightarrow{\sigma} m} \qquad \text{(sum}_2\text{)}\ \frac{(dec(q),I) \xrightarrow{\sigma} m}{(\{p+q\},I) \xrightarrow{\sigma} m}$$

$$\text{(s-pref}_1\text{)}\ \frac{(dec(p),I) \xrightarrow{\sigma} m}{(\{\underline{\gamma}.p\},I) \xrightarrow{\gamma\diamond\sigma} m} \qquad \text{(s-pref}_2\text{)}\ \frac{(dec(p),I) \xrightarrow{\sigma} m}{(\{\underline{\neg D}.p\},I\cup\{D\}) \xrightarrow{\sigma} m}$$

$$\text{(s-com)}\ \frac{(m_1,I) \xrightarrow{\sigma_1} m_1' \quad (m_2,\emptyset) \xrightarrow{\overline{\gamma}} m_2'}{(m_1 \oplus m_2,I) \xrightarrow{\sigma} m_1' \oplus m_2'}\ ad(m_1 \oplus m_2) \wedge Sync(\sigma_1,\overline{\gamma},\sigma)$$

Table 8.7 Rules for net transitions

Hence, this theorem states not only that function *dec* maps NPL processes to admissible markings over S_{NPL}, but also that *dec* is surjective over this set.

We now list some useful properties of places and markings, as done for FNM.

Proposition 8.15. *For any* $t \in \mathscr{P}_{NPL}^{\gamma,par}$, $sub(dom(dec(t))) \subseteq sub(t)$.

Proof. By induction on the definition of $dec(t)$. □

Corollary 8.1. *For any* $p \in \mathscr{P}_{NPL}$, $sub(dom(dec(p))) \subseteq sub(p)$.

Proof. If p is restriction-free, then the thesis follows by Proposition 8.15. If $p = (\nu L)t$, *then* $dec(p) = dec(t\{L'/L\})$ *by Proposition 8.13. By Proposition 8.1,* $sub(p) = sub(t\{L'/L\})$. *By Proposition 8.15,* $sub(dom(dec(t\{L'/L\}))) \subseteq sub(t\{L'/L\})$. *Hence,* $sub(dom(dec(p))) = sub(dom(dec(t\{L'/L\}))) \subseteq sub(t\{L'/L\}) = sub(p)$. □

8.4.2 Net Transitions

Let $\mathscr{A}^\gamma = \{\tau\} \cup \overline{\mathscr{G}} \cup \mathscr{G}^+$, ranged over by σ with abuse of notation, be the set of labels. Let $\rightarrow \subseteq \mathscr{M}_{fin}(S_{NPL}) \times \mathscr{P}_{fin}(D_{NPL}) \times \mathscr{A}^\gamma \times \mathscr{M}_{fin}(S_{NPL})$, be the least set of net transitions generated by the axiom and rules in Table 8.7, where a transition $t = (m,I,\sigma,m')$ is actually represented as $(m,I) \xrightarrow{\sigma} m'$.

Any transition t is often represented as $(^\bullet t, {}^\circ t) \xrightarrow{l(t)} t^\bullet$, where the *neg-set* ${}^\circ t$ of a transition t represents the set of places to be "tested for absence" of *additional* tokens. If $s \in {}^\circ t \cap dom(^\bullet t)$, then this means that t can be executed only if the tokens in s are precisely $^\bullet t(s)$; if $s \in {}^\circ t$ but $s \notin dom(^\bullet t)$, then t can be executed only if s contain no tokens at all.

Let us comment on the new rules of Table 8.7. In rule (s-pref$_2$), the negated strong prefix $\neg D$ is used to enlarge the neg-set I of the premise transition with the testable constant name D; note that the label σ of the premise is not modified in the conclusion, i.e., negated constant names do not occur in labels. Note also that this is the only rule adding elements to the neg-set of a transition; hence, a neg-set I is a subset of $\mathscr{T}cons = D_{NPL}$.

Rule (s-com) requires that the pre-set of the transition be admissible in order to avoid producing synchronized transitions that have no counterpart in the LTS semantics (see Proposition 8.21). The synchronization takes place if m_1 performs the input sequence σ_1 and m_2 performs a single output $\overline{\gamma}$ such that $Sync(\sigma_1, \overline{\gamma}, \sigma)$ holds.[8] Note that we do not need a symmetric rule to (s-com) because we are dealing with multisets whose elements are unordered (i.e., the multiset union operator $\oplus$ is associative and commutative). Note also that the neg-set associated with m_2 is to be empty, reflecting the fact that a net transition producing an output has nothing to do with strong prefixing, as it cannot be the result of a synchronization. The following Proposition 8.16, stating that net transitions labeled over $\overline{\mathscr{G}}$ have a singleton pre-set and an empty neg-set, formally justifies the shape of rule (s-com).

Proposition 8.16. *If $t = (m_1, I, \overline{\gamma}, m_2)$ is a transition derivable by the rules in Table 8.7, then $|m_1| = 1$ and $I = \emptyset$.*

Proof. By induction on the proof of t.

The base case is axiom (pref): $(\{\overline{\gamma}.p\}, \emptyset) \xrightarrow{\overline{\gamma}} dec(p)$, and the thesis is satisfied. If the last rule used in deriving t is (cons) with $C \doteq q$, then $t = (\{C\}, I) \xrightarrow{\overline{\gamma}} m_2$ and the premise transition is $(dec(q), I) \xrightarrow{\overline{\gamma}} m_2$. By induction, we can assume that $I = \emptyset$; hence, the thesis is satisfied as $|\{C\}| = 1$. The case of rules (sum_1) and (sum_2) is similar, and so omitted. No other rule is applicable, because by syntactic definition an output is not producible by a strongly prefixed process (which is in syntactic category s) and, moreover, the synchronization of an input sequence with one output produces a shorter input sequence (or a τ), which is not an output. □

Transitions with labels containing restricted actions must not be taken in the resulting net, as we accept only transitions labeled over $\mathscr{A} = \{\tau\} \cup \overline{\mathscr{L}} \cup \mathscr{L}^+$. However, they are useful in producing acceptable transitions, as two complementary restricted actions can synchronize, producing a τ-labeled transition or shortening the synchronized sequence. Hence, the NP/T net for NPL is the tuple $N_{NPL} = (S_{NPL}, D_{NPL}, \mathscr{A}, T_{NPL})$, where the set

$$T_{NPL} = \{(m_1, I, \sigma, m_2) \mid (m_1, I) \xrightarrow{\sigma} m_2 \text{ is derivable by the rules and } \sigma \in \mathscr{A}\}$$

is obtained by filtering out those transitions derivable by the rules whose label σ contains some restricted name a' or $\overline{a'}$.

[8] We assume that relation *Sync* has been extended also to restricted actions in the obvious way, i.e., a restricted output action $\overline{a'}$ can be synchronized only with its complementary restricted input action a' or with an atomic sequence starting with a'.

8.4.3 Properties of Net Transitions

As for FNM, some useful net transition properties are listed here.

Proposition 8.17. *Let $t = (m_1, I) \xrightarrow{\sigma} m_2$ be a transition derivable by the rules in Table 8.7. Then, $sub(dom(m_2)) \subseteq sub(dom(m_1))$ and $I \subseteq sub(dom(m_1))$.*

Proof. *By induction on the proof of t.* □

Lemma 8.8. *Let $t = (m_1, I) \xrightarrow{\sigma} m_2$ be a transition derivable by the rules in Table 8.7. Then, $fn(m_2) \subseteq fn(m_1)$ and $fn(I) \subseteq fn(m_1)$.*

Proof. *By induction on the proof of t.* □

Proposition 8.18. *Let $t = (m_1, I) \xrightarrow{\sigma} m_2$ be a transition derivable by the rules in Table 8.7. Then, m_1 and m_2 are admissible.*

Proof. *By induction on the proof of t, we can easily conclude that m_1 is admissible. Moreover, by Lemma 8.8, we have that $fn(m_2) \subseteq fn(m_1)$, and so m_2 is admissible, too.* □

Proposition 8.19. *Let $t = (m_1, I) \xrightarrow{\sigma} m_2$ be a transition derivable by the rules in Table 8.7. Let m be an admissible marking such that $m[t\rangle m'$. Then, m' is admissible.*

Proof. *Similar to the analogous Proposition 7.25.* □

Theorem 8.4. *If S_1 is admissible and $S_1 \Longrightarrow^* S_k$, then S_k is admissible.*

Proof. *By induction on the static reachability relation $\Longrightarrow^*$ (Definition 3.24), as done for the analogous Theorem 6.5, by using in places Lemma 8.8.* □

Now we want to prove that the net N_{NPL} contains only transitions that have a counterpart in the LTS semantics for NPL, as outlined in Section 8.2; this depends crucially on the admissibility condition over rule (s-com). This proposition exploits Lemma 8.9, which states a correspondence between net transitions derivable by the rules in Table 8.7 — which can be labeled over $\mathscr{A}^\gamma$ — and LTS transitions derivable by the rules in Table 8.1 from extended,[9] restriction-free processes, labeled over $\mathscr{A}^\gamma \times \mathscr{P}_{fin}(\mathscr{T}cons)$. Some auxiliary results are needed.

Proposition 8.20. *If $t = (m_1, I, \sigma, m_2)$ is a transition derivable by the rules in Table 8.7 and $L \subseteq \mathscr{L}$, then transition $t\{L'/L\} = (m_1\{L'/L\}, I\{L'/L\}, \sigma\{L'/L\}, m_2\{L'/L\})$ is derivable as well, where $\mu\{L'/L\} = \mu$ if $\mu, \overline{\mu} \notin L$, while $\mu\{L'/L\} = \mu'$ otherwise, $(a\sigma)\{L'/L\} = a\{L'/L\}\sigma\{L'/L\}$, and $I\{L'/L\} = \{D_{\{L'/L\}} \,|\, D \in I\}$. And vice versa, if $t\{L'/L\}$ is derivable, then also t is derivable.*

[9] As usual, we assume that the LTS operational rules, although defined for processes, can be used also for extended processes.

Proof. By induction on the proof of t. The proof is very similar to the analogous proof of Proposition 7.26; so, we discuss only a few cases.

If the last rule used in deriving t is (cons) with $C \doteq q$, then $t = (\{C\}, I) \xrightarrow{\sigma} m_2$ and the premise transition is $(dec(q), I) \xrightarrow{\sigma} m_2$. By induction, we can assume that also $(dec(q)\{L'/L\}, I\{L'/L\}) \xrightarrow{\sigma\{L'/L\}} m_2\{L'/L\}$ is derivable, where $dec(q)\{L'/L\} = dec(q\{L'/L\})$ by Lemma 8.6. Therefore, since $C\{L'/L\} = C_{\{L'/L\}} \doteq q\{L'/L\}$, by rule (cons), also $(\{C_{\{L'/L\}}\}, I\{L'/L\}) \xrightarrow{\sigma\{L'/L\}} m_2\{L'/L\}$ is derivable, where, by Lemma 8.6, also $\{C_{\{L'/L\}}\} = \{C\}\{L'/L\}$, as required.

If the last rule in deriving t is (s-pref$_2$), then $t = (\{\underline{\neg D}.p\}, I \cup \{D\}) \xrightarrow{\sigma} m_2$ and the premise transition is $(dec(p), I) \xrightarrow{\sigma} m_2$. By induction, we can assume that $(dec(p)\{L'/L\}, I\{L'/L\}) \xrightarrow{\sigma'\{L'/L\}} m_2\{L'/L\}$ is derivable, where, by Lemma 8.6, $dec(p)\{L'/L\} = dec(p\{L'/L\})$. Now $(\underline{\neg D}.p)\{L'/L\} = \underline{\neg D}_{\{L'/L\}}.p\{L'/L\}$. Hence, by rule (s-pref$_2$), also $(\{(\underline{\neg D}.p)\{L'/L\}\}, I\{L'/L\} \cup \{D_{\{L'/L\}}\}) \xrightarrow{\sigma\{L'/L\}} m_2\{L'/L\}$ is derivable, as required, as $I\{L'/L\} \cup \{D_{\{L'/L\}}\} = (I \cup \{D\})\{L'/L\}$.

For the vice versa — if $t\{L'/L\}$ is derivable, then also t is derivable — observe that this is similar to the above: if $t\{L'/L\}$ is derivable, then $(t\{L'/L\})\{L/L'\} = t$ is derivable, where an inverse substitution $\{L/L'\}$ is used instead. □

Lemma 8.9. *For any $t = (m_1, I, \sigma, m_2)$ derivable by the rules in Table 8.7, there exist two extended, restriction-free NPL processes p_1 and p_2 such that $p_1 \xrightarrow{(\sigma,I)} p_2$ is derivable by the rules in Table 8.1, with $dec(p_1) = m_1$ and $dec(p_2) = m_2$.*

Proof. By induction on the proof of t.

The base case of the induction is axiom (pref): $(\{\mu.p\}, \emptyset) \xrightarrow{\mu} dec(p)$. By axiom (Pref), also $\mu.p \xrightarrow{(\mu,\emptyset)} p$ is derivable and the thesis is satisfied.

If the last rule used to derive t is (cons) with $C \doteq q$, then $t = (\{C\}, I) \xrightarrow{\sigma} m_2$, whose premise is $(\{q\}, I) \xrightarrow{\sigma} m_2$. By induction, we can assume that $q \xrightarrow{(\sigma,I)} p_2$ is derivable, with p_2 an extended, restriction-free process such that $dec(p_2) = m_2$. Hence, by rule (Cons$_2$), also $C \xrightarrow{(\sigma,I)} p_2$ is derivable, as required. The cases of rules (sum$_1$), (sum$_2$) and (s-pref$_1$) are similar, and so omitted.

If the last rule used to derive t is (s-pref$_2$), then $t = (\{\underline{\neg D}.p\}, I \cup \{D\}) \xrightarrow{\sigma} m_2$, whose premise is $(\{p\}, I) \xrightarrow{\sigma} m_2$. By induction, we know that $p \xrightarrow{(\sigma,I)} p_2$ is derivable, with p_2 an extended, restriction-free process such that $dec(p_2) = m_2$. Hence, by rule (S-Pref$_2$), also $\underline{\neg D}.p \xrightarrow{(\sigma, I \cup \{D\})} p_2$ is derivable, so that the thesis is satisfied.

If the last rule used to derive t is (s-com), then there exist two transitions $t_1 = (m_1', I) \xrightarrow{\sigma_1} m_2'$ and $t_2 = (m_1'', \emptyset) \xrightarrow{\overline{\gamma}} m_2''$ such that $t = (m_1' \oplus m_1'', I) \xrightarrow{\sigma} m_2' \oplus m_2''$, $ad(m_1' \oplus m_1'')$ and $Sync(\sigma_1, \overline{\gamma}, \sigma)$. By induction, there exist transition $q_1 \xrightarrow{(\sigma_1,I)} q_2$ — with q_1 and q_2 extended, restriction-free processes such that $dec(q_1) = m_1'$ and $dec(q_2) = m_2'$ — and transition $r_1 \xrightarrow{(\overline{\gamma},\emptyset)} r_2$ — with r_1 and r_2 extended restriction-free processes such that $dec(r_1) = m_1''$ and $dec(r_2) = m_2''$. By Proposition 8.16,

$|m''_1| = 1$ and so r_1 is a sequential process. Therefore, by rule (S-Com), also transition $q_1 \,|\, r_1 \xrightarrow{(\sigma,I)} q_2 \,|\, r_2$ is derivable. Note that, by Lemma 8.7, $fn(q_1 \,|\, r_1) = fn(dec(q_1 \,|\, r_1)) = fn(m'_1 \oplus m''_1)$; therefore, $q_1 \,|\, r_1$ is an extended, restriction-free process because it is admissible, since the marking $m'_1 \oplus m''_1$ is assumed admissible by rule (s-com). Similarly, also $q_2 \,|\, r_2$ is admissible, because $fn(q_2 \,|\, r_2) = fn(dec(q_2 \,|\, r_2)) = fn(m'_2 \oplus m''_2)$ and the marking $m'_2 \oplus m''_2$ is admissible by Proposition 8.18. □

Proposition 8.21. *For any $t \in T_{NPL}$, where $t = (m_1, I, \sigma, m_2)$ with $\sigma \in \mathscr{A}$, there exist two NPL processes p_1 and p_2 such that $p_1 \xrightarrow{(\sigma,I)} p_2$, $dec(p_1) = m_1$ and $dec(p_2) = m_2$.*

Proof. *By Lemma 8.9, transition $q_1 \xrightarrow{(\sigma,I)} q_2$ is derivable by the rules in Table 8.1, for some suitable extended, restriction-free processes q_1 and q_2 such that $dec(q_1) = m_1$ and $dec(q_2) = m_2$. Let $L' = fn(m_1) \cap \mathscr{L}'$ and $L = \{a \mid a' \in L'\}$. If $L' = \emptyset$, then q_1 is actually an NPL process and the required transition is exactly $q_1 \xrightarrow{(\sigma,I)} q_2$, with q_2 also an NPL process, as $fn(q_2) \subseteq fn(q_1)$. On the contrary, if $L' \neq \emptyset$, then for all $a \in L, a \notin fn(m_1)$ because m_1 is admissible, so that $a \notin n(\sigma)$. Moreover, by Proposition 8.20, also $t\{L/L'\} = (m_1\{L/L'\}, I\{L/L'\}, \sigma\{L/L'\}, m_2\{L/L'\})$ is derivable, and the same proof of transition $q_1 \xrightarrow{(\sigma,I)} q_2$ can be used to prove that transition $q_1\{L/L'\} \xrightarrow{(\sigma\{L/L'\}, I\{L/L'\})} q_2\{L/L'\}$ is derivable by the rules in Table 8.1; note that, since $\sigma \in \mathscr{A}$, we have that $\sigma\{L/L'\} = \sigma$; moreover $q_1\{L/L'\}$ and $q_2\{L/L'\}$ are NPL processes, because, for $i = 1,2$, $fn(q_i) = fn(dec(q_i)) = fn(m_i)$ by Lemma 8.7, and so all the occurrences of the restricted action a' occurring in q_i are turned into their corresponding action a in $q_i\{L/L'\}$. By rule (S-Res), the transition $(\nu L)(q_1\{L/L'\}) \xrightarrow{(\sigma, I\{L/L'\})\{L'/L\}} (\nu L)(q_2\{L/L'\})$ is derivable, too, where $(\sigma, I\{L/L'\})\{L'/L\} = (\sigma, I)$, and $(\nu L)(q_i\{L/L'\})$ is an NPL process, for $i = 1,2$. Note that $dec((\nu L)(q_i\{L/L'\})) = m_i$, as required, because $dec((\nu L)(q_i\{L/L'\})) = dec(q_i\{L/L'\})\{L'/L\}$ by definition of function dec, and $dec(q_i\{L/L'\})\{L'/L\} = dec((q_i\{L/L'\})\{L'/L\})$ by Lemma 8.6, and, by Proposition 6.3 (extended to NPL), also $dec((q_i\{L/L'\})\{L'/L\}) = dec(q_i)$.* □

Corollary 8.2. *For any $t \in T_{NPL}$, where $t = (m_1, I, \sigma, m_2)$ with $\sigma \in \mathscr{A}$, there exist two NPL fixed processes p_1 and p_2 such that $p_1 \overset{\sigma}{\longmapsto} p_2$, with $dec(p_1) = m_1$ and $dec(p_2) = m_2$.*

Proof. *By Proposition 8.21, there exist two processes p_1 and p_2 such that $p_1 \xrightarrow{(\sigma,I)} p_2$, $dec(p_1) = m_1$ and $dec(p_2) = m_2$. The thesis then follows by rule (Abs).* □

We now want to prove that for any finite set of places $S \subseteq S_{NPL}$, the set of transitions statically enabled at S is finite. An auxiliary lemma is necessary. Given a single place $s \in S$, by $s \vdash t$ we mean that transition $t = (\{s\}, I, \sigma, m)$ is derivable by the rules in Table 8.7, hence with $\sigma \in \mathscr{A}^{\gamma}$.

Lemma 8.10. *The set $T_s = \{t \mid s \vdash t\}$ is finite, for any $s \in S_{NPL}$.*

Proof. By induction on the axiom and rules in Table 8.7. □

Given a finite set of places $S \subseteq S_{NPL}$, let T_1^S be $\bigcup_{s \in S} T_s$, i.e., the set of all transitions with a singleton pre-set in S derivable by the rules with labeling in $\mathscr{A}^\gamma$. The set T_1^S is finite, being the finite union (as S is finite) of finite sets (as T_s is finite for any s, by Lemma 8.10).

Let $k \in \mathbb{N}$ be the length of the longest label of any transition in T_1^S. If a multi-party transition t is derivable by the rules from the set S, then its proof contains k synchronizations at most, each one between a transition (labeled with a sequence of inputs) and a *singleton-pre-set* transition (labeled with an output action); hence, at most $k+1$ participants can take part in a multi-party synchronization. Therefore, the set of all the transitions statically enabled at a finite set S can be defined by means of a sequence of sets T_i^S of transitions, for $2 \leq i \leq k+1$, where each transition $t \in T_i^S$ has a pre-set ${}^\bullet t$ of size i, as follows:

$$\begin{aligned} T_i^S = \{ & (m_1 \oplus m_2, I, \sigma, m_1' \oplus m_2') \;\big|\; ad(m_1 \oplus m_2), \\ & \exists \sigma_1 \exists \gamma . (m_1, I, \sigma_1, m_1') \in T_{i-1}^S, (m_2, \emptyset, \overline{\gamma}, m_2') \in T_1^S, Sync(\sigma_1, \overline{\gamma}, \sigma) \}. \end{aligned}$$

Note that T_2^S is finite, because T_1^S is finite; inductively, T_{i+1}^S, for $2 \leq i \leq k$, is finite, because T_i^S and T_1 are finite. In conclusion, the set T_S of all the transitions statically enabled at S is

$$T_S = \{ t \;\big|\; t \in \bigcup_{i=1}^{k+1} T_i^S \wedge l(t) \in \mathscr{A} \},$$

where only transitions labeled over $\mathscr{A}$ are considered. T_S is finite, being a finite union of finite sets; therefore, we have the following result.

Theorem 8.5. *If $S \subseteq S_{NPL}$ is a finite set of places, then set $T_S \subseteq T_{NPL}$ of all the transitions statically enabled at S is finite.* □

Example 8.8. **(Counter)** Continuing Example 8.3 about the counter,

$$\begin{aligned} & C \doteq \underline{\neg D}.zero.C + inc.(C \,|\, D), \\ & D \doteq dec.\mathbf{0}, \end{aligned}$$

we have that the initial marking m_0 is $dec(C) = \{C\}$. Let $S_0 = \{C\}$. The set of transitions statically enabled at S_0 is $T_1^{S_0} = T_C = \{t_1, t_2\}$, where $t_1 = (\{C\}, \emptyset) \xrightarrow{inc} \{C, D\}$ and $t_2 = (\{C\}, \{D\}) \xrightarrow{zero} \{C\}$. Therefore, the new set of statically reachable places is $S_1 = \{C, D\}$. Note that D can produce one transition in T_D, i.e., $t_3 = (\{D\}, \emptyset) \xrightarrow{dec} \emptyset$. Hence, $T_{S_1} = \{t_1, t_2, t_3\}$, as these are labeled over $\mathscr{A}$. As t_3 does not add any new reachable place, we have that S_1 is the set of places statically reachable from the initial marking, and T_{S_1} is the set of transitions statically enabled at S_1. The resulting NP/T net is depicted in Figure 8.1. □

Example 8.9. Let us consider the process $p = (\nu L)(D \,|\, C \,|\, C \,|\, E)$, where $L = \{d, e\}$, $\{D, E\} \subseteq \mathscr{T}cons$ and the constant definitions are

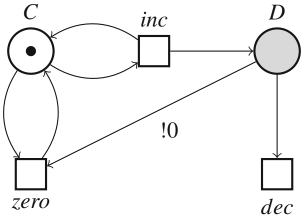


Fig. 8.1 The NP/T net for the counter

$$\begin{aligned}
D &\doteq \underline{\neg D}.\underline{d}.a.D,\\
C &\doteq \overline{d}.C + \overline{e}.C,\\
E &\doteq \underline{\neg E}.\underline{e}.b.E.
\end{aligned}$$

The initial marking $dec(p)$ is $D_{\{L'/L\}} \oplus 2 \cdot C_{\{L'/L\}} \oplus E_{\{L'/L\}}$, abbreviated as $s_1 \oplus 2 \cdot s_2 \oplus s_3$. Hence, $S_0 = dom(dec(p)) = \{s_1, s_2, s_3\}$. The sets of transitions statically enabled at each initial place are $T_{s_1} = \{t_1\}$, where $t_1 = (\{s_1\},\{s_1\}) \xrightarrow{d'a} \{s_1\}$, $T_{s_2} = \{t_2, t_3\}$, where $t_2 = (\{s_2\}, \emptyset) \xrightarrow{\overline{d'}} \{s_2\}$ and $t_3 = (\{s_2\}, \emptyset) \xrightarrow{\overline{e'}} \{s_2\}$, and $T_{s_3} = \{t_4\}$, where $t_4 = (\{s_3\},\{s_3\}) \xrightarrow{e'b} \{s_3\}$. The longest label is of length two and so we have to compute $T_i^{S_0}$ for $i = 1, 2, 3$:

$$\begin{aligned}
T_1^{S_0} &= \{t_1, t_2, t_3, t_4\},\\
T_2^{S_0} &= \{t_5, t_6\}, \text{where transition } t_5 = (\{s_1, s_2\}, \{s_1\}) \xrightarrow{a} \{s_1, s_2\} \text{ and}\\
&\qquad\qquad\qquad\quad \text{transition } t_6 = (\{s_2, s_3\}, \{s_3\}) \xrightarrow{b} \{s_2, s_3\},\\
T_3^{S_0} &= \emptyset.
\end{aligned}$$

Hence, $T_{S_0} = \{t_5, t_6\}$, because only these two transitions are labeled over $\mathscr{A}$. The resulting net is outlined in Figure 3.20(a). □

8.4.4 *The Reachable Subnet* $Net(p)$

The NP/T net system associated with a process $p \in \mathscr{P}_{NPL}$ is the subnet of N_{NPL} statically reachable from the initial marking $dec(p)$, denoted by $Net(p)$.

Definition 8.4. Let p be a process in $\mathscr{P}_{NPL}$. The P/T net system statically associated with p is $Net(p) = (S_p, D_p, A_p, T_p, m_0)$, where $m_0 = dec(p)$ and

$$\begin{aligned}
S_p &= [\![dom(m_0)\rangle \quad \text{computed in } N_{NPL},\\
D_p &= S_p \cap D_{NPL},\\
T_p &= \{t \in T_{NPL} \mid S_p[\![t\rangle\},\\
A_p &= \{\sigma \in \mathscr{A} \mid \exists t \in T_p \text{ such that } l(t) = \sigma\}.
\end{aligned}$$ □

The following propositions present three facts that are obviously true by construction of the net $Net(p)$ associated with an NPL process p.

Proposition 8.22. *For any $p \in \mathscr{P}_{NPL}$, $Net(p)$ is a statically reduced NP/T net.* □

Proposition 8.23. *If $dec(p) = dec(q)$, then $Net(p) = Net(q)$.* □

Proposition 8.24. *For any restriction-free $t \in \mathscr{P}_{NPL}$ and for any $L \subseteq \mathscr{L}$, the following hold.*

i) If $Net(t) = (S,D,A,T,m_0)$, then, for any $n \geq 1$, $Net(t^n) = (S,D,A,T,n \cdot m_0)$, where $t^1 = t$ and $t^{n+1} = t \,|\, t^n$.
ii) If $Net((\nu L)t) = (S,D,A,T,m_0)$, then $Net((\nu L)(t^n)) = (S,D,A,T,n \cdot m_0)$, for any $n \geq 1$. □

An obvious algorithm to compute $Net(p)$ is based on the inductive definition of the static reachability relation: start with the initial set of places $S_0 = dom(dec(p))$, and then apply the rules in Table 8.7 in order to produce the set T_{S_0} of transitions (labeled over $\mathscr{A}$) statically enabled at S_0, as well as the additional places statically reachable by means of such transitions. Then, repeat this procedure from the set of places statically reached so far. An instance of this procedure was given in Examples 8.8 and 8.9. There are two problems with this algorithm:

- the obvious *halting condition* is "until no new places are statically reachable"; of course, the algorithm terminates if we know that the set S_p of places statically reachable from $dom(dec(p))$ is finite; additionally,
- at each step of the algorithm, we have to be sure that the set of transitions derivable from the current set of statically reachable places is finite.

We are going to prove the first requirement — S_p is finite for any $p \in \mathscr{P}_{NPL}$ — because it implies also the second one: if S_p is finite, then any set S of places statically reachable from $dom(dec(p))$ is finite, and so, by Theorem 8.5, the set T_S of transitions statically enabled at the finite set S is finite, too.

Theorem 8.6. *For any $p \in \mathscr{P}_{NPL}$, let $Net(p) = (S_p, D_p, A_p, T_p, m_0)$ be defined as in Definition 8.4. Then, the set S_p is finite.*

Proof. *We prove, by induction on the static reachability relation $\Longrightarrow^*$, that any set S_i of places that is statically reachable from $dom(m_0) = dec(p)$ is a subset of $sub(dom(m_0))$. This is enough as, by Corollary 8.1, we know that $sub(dom(m_0)) \subseteq sub(p)$; moreover, by Theorem 8.1, $sub(p)$ is finite and so the thesis follows trivially.*

The base case is $dom(m_0) \Longrightarrow^* dom(m_0)$. By Proposition 7.21 (trivially extended to NPL), we have the required $dom(m_0) \subseteq sub(dom(m_0))$. Now, let us assume that S_i is a set of places statically reachable from $dom(m_0)$ and let $t = (m_1, I) \xrightarrow{\sigma} m_2$ be such that $S_i \overset{t}{\Longrightarrow} S_{i+1}$. By induction, we know that $S_i \subseteq sub(dom(m_0))$. So, we have to prove that the new places reached via t, i.e., $dom(m_2) \cup I$, are in $sub(dom(m_0))$. Note that since $dom(m_1) \subseteq S_i$, it follows that $dom(m_1) \subseteq sub(dom(m_0))$ and also that $sub(dom(m_1)) \subseteq sub(dom(m_0))$, by Proposition 7.20 (trivially extended to NPL). By

Proposition 7.21, we have that $dom(m_2) \subseteq sub(dom(m_2))$; by Proposition 8.17, we have that $sub(dom(m_2)) \subseteq sub(dom(m_1))$ and $I \subseteq sub(dom(m_1))$; by transitivity, $dom(m_2) \cup I \subseteq sub(dom(m_0))$, and so $S_{i+1} = S_i \cup dom(m_2) \cup I \subseteq sub(dom(m_0))$, as required.

Summing up, any place statically reachable from $dom(m_0)$ is a sequential subterm of p. As by Theorem 7.1, $sub(p)$ is finite, then also S_p (the largest set of places statically reachable from $dom(m_0)$) is finite. □

Theorem 8.7. *For any NPL process p, $Net(p) = (S_p, D_p, A_p, T_p, dec(p))$ is a finite NP/T Petri net.*

Proof. The set $S_0 = dom(dec(p))$ is finite, by Proposition 8.14. By Theorem 8.5, the set T_{S_0} of all the transitions statically enabled at S_0 is finite. Let S_1 be the set of places $S_0 \cup \bigcup_{t \in T_{S_0}} dom(t^\bullet) \cup {}^\circ t$. If $S_1 = S_0$, then $S_p = S_0$ and $T_p = T_{S_0}$. Otherwise, repeat the step above for S_1; in fact, S_1 is a finite set of places, because S_0 is finite, the set T_{S_0} is finite and each transition has a finite post-set. By repeating the step above for S_1, we compute a new finite set T_{S_1} of transitions statically enabled at S_1, and a new finite set S_2 of places statically reachable from S_1 via the transitions in T_{S_1}; if $S_2 = S_1$, then $S_p = S_1$ and $T_p = T_{S_1}$. Otherwise, repeat the step above for S_2. This procedure will end eventually because, by Theorem 8.6, we are sure that S_p is a finite set. □

Hence, *only* finite NP/T nets can be represented by NPL processes.

8.5 Representing All Finite NP/T Nets

It is not a surprise that *all* finite NP/T nets can be represented by NPL processes. The construction described in Section 7.5 for finite P/T nets is to be easily modified in order to cope with transitions having a nonempty neg-set. The only addition is that the summand c_i^j of the leader constant C_i for transition t_j is strongly prefixed by the negated names of the testable constants corresponding to the places in the neg-set of t_j.

Definition 8.5. (Translating finite NP/T nets into NPL processes) Given $A \subseteq \mathscr{L} \cup \{\tau\}$, let $N(m_0) = (S, D, A, T, m_0)$ — with $S = \{s_1, \ldots, s_n\}$, $T = \{t_1, \ldots, t_k\}$ and $l(t_j) = \mu_j$ — be a finite NP/T net. Function $\mathscr{T}_{NPL}(-)$, from finite NP/T nets to NPL processes, is defined as

$$\mathscr{T}_{NPL}(N(m_0)) = (\nu L)(\underbrace{C_1|\cdots|C_1}_{m_0(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m_0(s_n)})$$

where $L = \{x_1^1, \ldots, x_n^1, x_1^2, \ldots, x_n^2, \ldots, x_1^k, \ldots, x_n^k\}$ is such that $L \cap A = \emptyset$, for $i = 1, \ldots, n$, $C_i \in \mathscr{T}cons$ if and only if $s_i \in D$, and each C_i is equipped with a defining equation $C_i \doteq c_i^1 + \cdots + c_i^k$ (with $C_i \doteq \mathbf{0}$ if $k = 0$), where each summand c_i^j, for $j = 1, \ldots, k$, is equal to

- $\mathbf{0}$, if $s_i \not\in {}^\bullet t_j$;
- $\underline{\neg C}^j_1 . \cdots . \underline{\neg C}^j_{h_j} . \mu_j . \Pi_j$, if ${}^\bullet t_j = \{s_i\}$ and ${}^\circ t_j = \{s^j_1, \ldots, s^j_{h_j}\}$ for $h_j \geq 0$;
- $\overline{x}^j_i . \mathbf{0}$, if ${}^\bullet t_j(s_i) > 0$ and ${}^\bullet t_j(s_{i'}) > 0$ for some $i' < i$ (i.e., s_i is not the leader for the synchronization on t_j);
- $\underline{\neg C}^j_1 . \cdots . \underline{\neg C}^j_{h_j} . \underbrace{\underline{x}^j_{i+1} . \cdots . \underline{x}^j_{i+1}}_{{}^\bullet t_j(s_{i+1})} . \cdots . \underbrace{\underline{x}^j_n . \cdots . \underline{x}^j_n}_{{}^\bullet t_j(s_n)} . \mu_j . \Pi_j$, if ${}^\bullet t_j(s_i) = 1$, s_i is the leader of the synchronization (i.e., ${}^\bullet t_j(s_{i'}) > 0$ for no $i' < i$, while ${}^\bullet t_j(s_{i'}) > 0$ for some $i' > i$) and ${}^\circ t_j = \{s^j_1, \ldots, s^j_{h_j}\}$ for $h_j \geq 0$;
- $\overline{x}^j_i . \mathbf{0} + \underline{\neg C}^j_1 . \cdots . \underline{\neg C}^j_{h_j} . \underbrace{\underline{x}^j_i . \cdots . \underline{x}^j_i}_{{}^\bullet t_j(s_i) - 1} . \underbrace{\underline{x}^j_{i+1} . \cdots . \underline{x}^j_{i+1}}_{{}^\bullet t_j(s_{i+1})} . \cdots . \underbrace{\underline{x}^j_n . \cdots . \underline{x}^j_n}_{{}^\bullet t_j(s_n)} . \mu_j . \Pi_j$, otherwise (i.e., s_i is the leader and ${}^\bullet t_j(s_i) \geq 2$, with ${}^\circ t_j = \{s^j_1, \ldots, s^j_{h_j}\}$ for $h_j \geq 0$).

Finally, process Π_j is $\underbrace{C_1 | \cdots | C_1}_{t^\bullet_j(s_1)} | \cdots | \underbrace{C_n | \cdots | C_n}_{t^\bullet_j(s_n)}$, meaning that $\Pi_j = \mathbf{0}$ if $t^\bullet_j = \emptyset$. □

Example 8.10. Let us consider the NP/T net $N(m_0)$ of Figure 3.20(a), where its two transitions are $t_1 = (\{s_1, s_2\}, \{s_1\}) \xrightarrow{a} \{s_1, s_2\}$ and $t_2 = (\{s_2, s_3\}, \{s_3\}) \xrightarrow{b} \{s_2, s_3\}$, and the initial marking is $m_0 = s_1 \oplus 2 \cdot s_2 \oplus s_3$. The NPL term $\mathscr{T}_{NPL}(N(m_0))$ is $q = (\nu L)(C_1 \,|\, C_2 \,|\, C_2 \,|\, C_3)$, where $L = \{x^1_1, x^1_2, x^1_3, x^2_1, x^2_2, x^2_3\}$ and the constants are defined as follows:

$$
\begin{aligned}
C_1 &\doteq \underline{\neg C}_1 . \underline{x}^1_2 . a . (C_1 \,|\, C_2) + \mathbf{0}, \\
C_2 &\doteq \overline{x}^1_2 . \mathbf{0} + \underline{\neg C}_3 . \underline{x}^2_3 . b . (C_2 \,|\, C_3), \\
C_3 &\doteq \mathbf{0} + \overline{x}^2_3 . \mathbf{0}.
\end{aligned}
$$

Note that only C_1 and C_3 are testable constants, because only s_1 and s_3 are testable places. It is easy to realize that $Net(q)$ is isomorphic to $N(m_0)$. Note that also the process p of Example 8.9 generates an NP/T net $Net(p)$ which is isomorphic to $N(m_0)$. □

The term $\mathscr{T}_{NPL}(N(m_0))$ is an NPL process: in fact, the restriction operator occurs only at the top level, applied to the parallel composition of a finite number of constants; each constant has a body that is sequential and restriction-free. Note also that each *strong prefix* is a bound input $\underline{x}^j_i$ or a negated constant name $\underline{\neg C}^j_h$, and any sequence ends with an action $\mu_j \in A$, which is either an input or τ; hence, each strongly prefixed summand generates an atomic sequence without any output. Therefore, the following proposition holds by Theorem 8.7 and Proposition 8.22.

Proposition 8.25. *For any finite NP/T Petri net $N(m_0)$, the net $Net(\mathscr{T}_{NPL}(N(m_0)))$ is a finite, statically reduced NP/T net.* □

Moreover, $\mathscr{T}_{NPL}(N(m_0))$ is *output-closed*, as $fn(\mathscr{T}_{NPL}(N(m_0))) \cap \overline{\mathscr{L}} = \emptyset$; in fact, any output occurring in this term is of the form $\overline{x}^j_i$ for suitable i and j, and such a name is bound.

Now we are ready to state our main result, the so-called *representability theorem*.

Theorem 8.8. (Representability theorem 6) *Let $N(m_0) = (S, D, A, T, m_0)$ be a finite, statically reduced NP/T net system such that $A \subseteq \mathscr{L} \cup \{\tau\}$, and let $p = \mathscr{T}_{NPL}(N(m_0))$. Then, $Net(p)$ is isomorphic to $N(m_0)$.*

Proof. The proof is very similar to the analogous proof of Theorem 7.12 and so we focus only on the new aspect of the neg-set of a transition.

Let $N(m_0) = (S, D, A, T, m_0)$ be a reduced, finite NP/T net, with $S = \{s_1, \ldots, s_n\}$, $A \subseteq \mathscr{L} \cup \{\tau\}$, $T = \{t_1, \ldots, t_k\}$ and $l(t_j) = \mu_j$ for $j = 1, \ldots, k$. The associated NPL process is

$$\mathscr{T}_{NPL}(N(m_0)) = (\nu L)(\underbrace{C_1|\cdots|C_1}_{m_0(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m_0(s_n)}),$$

where $L = \{x_1^1, \ldots, x_n^1, x_1^2, \ldots, x_n^2, \ldots, x_1^k, \ldots, x_n^k\}$, $L \cap A = \emptyset$ and for each place s_i we have a corresponding constant $C_i \doteq \sum_{j=1}^{k} c_i^j$, defined as in Definition 8.5, where $C_i \in \mathscr{T}cons$ iff $s_i \in D$. For notational convenience, $\sum_{j=1}^{k} c_i^j$ is denoted by p_i, i.e., $C_i \doteq p_i$; for the same reason, we use p to denote $\mathscr{T}_{NPL}(N(m_0))$.

Let $\theta = \{L'/L\}$ be a substitution that maps each bound name x_i^j to its corresponding restricted name $x_i'^j$ in $\mathscr{L}'$, for $i = 1, \ldots, n$ and $j = 1, \ldots, k$. Let $Net(p) = (S', D', A', T', m_0')$. Then, $m_0' = dec(p)$ is the multiset

$$dec((\nu L)(\underbrace{C_1|\cdots|C_1}_{m_0(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m_0(s_n)})) = dec(\underbrace{C_1|\cdots|C_1}_{m_0(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m_0(s_n)})\theta =$$
$$m_0(s_1) \cdot C_1\theta \oplus \cdots \oplus m_0(s_n) \cdot C_n\theta,$$

where $C_i\theta$ gives rise to the new constant $C_{i,\theta} \doteq p_i\theta$. Hence, the initial places are all of the form $C_i\theta$, where each such place is present in m_0' only if $m_0(s_i) > 0$.

Note that, by Definition 8.5, any transition $t' \in T'$ with ${}^\bullet t' \subseteq dec(p)$ is such that, for some suitable j, ${}^\circ t' = \{C_1^j\theta, \ldots, C_{h_j}^j\theta\}$ and $t'^\bullet = dec(\Pi_j)\theta$, and so equal to $k_1 \cdot C_1\theta \oplus k_2 \cdot C_2\theta \oplus \ldots \oplus k_n \cdot C_n\theta$ for suitable $k_i \geq 0$, $i = 1, \ldots, n$; by iterating this observation, each transition in T' has a neg-set of the form $\{C_1^j\theta, \ldots, C_{h_j}^j\theta\}$ and a post-set of the form $dec(\Pi_j)\theta$, for some suitable $j = 1, \ldots, k$. Hence, each statically reachable place s_i' in S' is of the form $C_i\theta$. Moreover, by Proposition 8.25, $Net(p)$ is statically reduced, implying that all the $C_i\theta$'s are statically reachable, for $i = 1, \ldots, n$. Hence, there is a bijection $f : S \to S'$ defined by $f(s_i) = s_i' = C_i\theta$, which is the natural candidate isomorphism function; note that f preserves the typing: $s_i \in D$ iff $f(s_i) = C_i\theta \in D'$. To prove that f is an isomorphism, we have to prove that

1. *$f(m_0) = m_0'$,*
2. *$t = (m, I, \mu, m') \in T$ implies $f(t) = (f(m), f(I), \mu, f(m')) \in T'$, and*
3. *$t' = (m_1', I', \mu, m_2') \in T'$ implies there exists $t = (m_1, I, \mu, m_2) \in T$ such that $f(t) = t'$, i.e., $f(m_1) = m_1'$, $f(I) = I'$ and $f(m_2) = m_2'$.*

From items 2) and 3) above, it follows that $A = A'$.

Proof of *1: omitted, as it is the same as for the FNM case.*

Proof of 2*: We prove that, for* $j = 1,\dots,k$, *if* $t_j = (m, I, \mu_j, m') \in T$, *then* $t'_j = (f(m), f(I), \mu_j, f(m')) \in T'$. *We now examine the possible cases, by inspecting the shape of* ${}^\bullet t_j$. *Let us examine the three cases separately:*

- ${}^\bullet t_j = \{s_i\}$ *and so* $f({}^\bullet t_j) = \{C_i\theta\}$. *By Definition 8.5, for* $C_i = p_i$, *we have in* p_i *a summand* $c^j_i = \underline{\neg C}^j_1.\dots.\underline{\neg C}^j_{h_j}.\mu_j.\Pi_j$, *with* $\Pi_j = \underbrace{C_1|\cdots|C_1}_{m'(s_1)}|\cdots|\underbrace{C_n|\cdots|C_n}_{m'(s_n)}$.

  Therefore, not only transition $(\{c^j_i\}, J) \xrightarrow{\mu_j} dec(\Pi_j)$, *where* $J = \{C^j_1,\dots,C^j_{h_j}\}$, *is derivable by axiom (pref), followed by* h_j *applications of rule (s-pref*$_2$*), but also* $(\{p_i\}, J) \xrightarrow{\mu_j} dec(\Pi_j)$ *is derivable by (possibly repeated applications of) rules (sum*$_1$*) and (sum*$_2$*), and so* $(\{C_i\}, J) \xrightarrow{\mu_j} dec(\Pi_j)$, *by rule (cons). Hence, by Proposition 8.20,* $(\{C_i\theta\}, J\theta) \xrightarrow{\mu_j} dec(\Pi_j)\theta$ *is derivable, as* $\mu_j \notin L$. *In conclusion, starting from transition* $t_j = (\{s_i\}, I, \mu_j, m')$, *where* $I = \{s^j_1,\dots,s^j_{h_j}\}$, *we have shown that transition* $t'_j = (\{C_i\theta\}, J\theta, \mu_j, dec(\Pi_j)\theta)$, *with* $f(m') = dec(\Pi_j)\theta$ *and* $f(I) = J\theta$, *belongs to* T', *as required.*
- ${}^\bullet t_j = s_i \oplus \overline{m}$, *such that* $\overline{m}(s_h) > 0$ *only if* $h > i$ *(i.e.,* s_i *is the leader and* ${}^\bullet t_j(s_i) = 1$*), and so* $f({}^\bullet t_j) = C_i\theta \oplus f(\overline{m})$. *This is very similar to the analogous case for FNM, adapted to the strong prefixes for the neg-set, as shown above.*
- ${}^\bullet t_j = k \cdot s_i \oplus \overline{m}$, *such that* $\overline{m}(s_h) > 0$ *only if* $h > i$ *(i.e.,* s_i *is the leader and* ${}^\bullet t_j(s_i) = k \geq 2$*), and so* $f({}^\bullet t_j) = k \cdot C_i\theta \oplus f(\overline{m})$. *This is very similar to the analogous case for FNM, adapted to the strong prefixes for the neg-set, as shown above.*

Proof of *3: We prove that if* $t'_j = (m'_1, I', \mu_j, m'_2) \in T'$, *then there exists a transition* $t_j = (m_1, I, \mu_j, m_2) \in T$ *such that* $f(m_1) = m'_1$, $f(I) = I'$ *and* $f(m_2) = m'_2$. *This is proved by case analysis on the three possible shapes of the marking* m'_1, *which can be* $\{C_i\theta\}$, *or* $dec(P_j)\theta$, *where* $P_j = (\underbrace{C_i|\cdots|C_i}_{k_i}|\underbrace{C_{i+1}|\cdots|C_{i+1}}_{k_{i+1}}|\cdots|\underbrace{C_n|\cdots|C_n}_{k_n})$, *with either* $k_i = 1$ *or* $k_i \geq 2$.

- *If* $m'_1 = \{C_i\theta\}$ *for some* $i = 1,\dots,n$, *then* $t'_j = (\{C_i\theta\}, J\theta) \xrightarrow{\mu_j} m'_2$. *By Proposition 8.20, also transition* $(\{C_i\}, J) \xrightarrow{\mu_j} m''_2$ *is derivable, with* $m''_2\theta = m'_2$. *According to Definition 8.5, such a transition is derivable by the rules only if among the many summands composing* p_i *there exists a summand* $c^j_i = \underline{\neg C}^j_1.\dots.\underline{\neg C}^j_{h_j}.\mu_j.\Pi_j$, *such that* $J = \{C^J_1,\dots,C^J_{h_j}\}$, *which is possible only if in* $N(m_0)$ *we have a transition* t_j *with* ${}^\bullet t_j = \{s_i\}$, $f(\{s_i\}) = \{C_i\theta\}$, ${}^\circ t_j = I = \{s^j_1,\dots,s^j_{h_j}\}$, $f(I) = J\theta$, $f(t^\bullet_j) = dec(\Pi_j)\theta = m'_2$ *and* $l(t_j) = \mu_j$, *as required.*
- *The case when* $m'_1 = C_i\theta \oplus k_{i+1} \cdot C_{i+1}\theta \oplus \dots \oplus k_n \cdot C_n\theta$, *for some* $i = 1,\dots,n$, *is analogous to the corresponding case for FNM, adapted to take care of the neg-set as done above.*

- *the case when $m'_1 = k_i \cdot C_i\theta \oplus k_{i+1} \cdot C_{i+1}\theta \oplus \ldots \oplus k_n \cdot C_n\theta$, with $k_i \geq 2$, is analogous to the corresponding case for FNM, adapted to take care of the neg-set as done above.*

No further cases are possible, because a transition $t'_j = (m'_1, I', \mu_j, m'_2) \in T'$ must be derivable by the rules. □

8.6 Soundness

In this section, we want to prove that the operational net semantics preserves the concrete step semantics of Section 8.3. More precisely, we show that any NPL process p, in the concrete STS semantics of Section 8.3, is bisimilar to $dec(p)$, in the concrete step marking graph $cSMG(Net(p))$ of Definition 3.27.

Proposition 8.26. *For any $p \in \mathscr{P}_{NPL}$ the following hold:*

1. *if $p \xrightarrow{M}_s p'$ in the STS of Section 8.3, then $dec(p) \xrightarrow{M}_s dec(p')$ in $cSMG(Net(p))$;*
2. *if $dec(p) \xrightarrow{M}_s m$ in $cSMG(Net(p))$, then there exists p' such that $dec(p') = m$ and $p \xrightarrow{M}_s p'$ in the STS of Section 8.3.*

Proof. By induction on the definition of $dec(p)$ and then by induction on the proof of the involved transitions; we prove the two statements simultaneously.

- *If $p = \mathbf{0}$, then $dec(p) = \emptyset$; this case is vacuously true, as $\mathbf{0}$ is a deadlock state in the STS of Section 8.3 and also $\emptyset$ is a deadlock state in $cSMG(Net(p))$.*
- *If p is any other sequential process, then $dec(p) = \{p\}$.*
  - *Proof of 1: By additional induction on the proof of $p \xrightarrow{M}_s p'$. The base case is axiom (Pref^s): $\mu.p \xrightarrow{\{(\mu,\emptyset)\}}_s p$. By axiom (pref), $t = (\{\mu.p\}, \emptyset) \xrightarrow{\mu} dec(p)$ is derivable, and so the step $G = \{t\}$ is such that $\{\mu.p\}[G\rangle dec(p)$. Hence, by Definition 3.27, $\{\mu.p\} \xrightarrow{\{(\mu,\emptyset)\}}_s dec(p)$, as required.*

    If the last rule used in deriving $p \xrightarrow{M}_s p'$ is (Cong^s), then $p \equiv q$, $q \xrightarrow{M}_s q'$ and $q' \equiv p'$. By induction, we know that $dec(q) \xrightarrow{M}_s dec(q')$ and so the thesis follows because $dec(p) = dec(q)$ and $dec(p') = dec(q')$.

    If the last rule used in deriving $p \xrightarrow{M}_s p'$ is (Sum$^s_1$), then $p = p_1 + p_2$, $M = \{(\sigma, I)\}$ and the premise is $p_1 \xrightarrow{M}_s p'$. By induction, we know that $dec(p_1)[\{t'\}\rangle dec(p')$ with $l(t') = \sigma$ and ${}^\circ t' = I$. Since p_1 is sequential, this step is possible only if $t' = (\{p_1\}, I) \xrightarrow{\sigma} dec(p')$ is derivable by the rules in Table 8.7; hence, by rule (sum$_1$), also the net transition $t = (\{p_1 + p_2\}, I) \xrightarrow{\sigma} dec(p')$ is derivable, and so the step $dec(p_1 + p_2)[\{t\}\rangle dec(p')$ is also derivable. Hence, by Definition 3.27, $dec(p_1 + p_2) \xrightarrow{\{(\sigma,I)\}}_s dec(p')$, as required.

The cases of rules (Sum^s_2), ($Cons^s_2$), (S-$Pref^s_1$) and (S-$Pref^s_2$) are similar, and so omitted.
If the last rule used in deriving $p \xrightarrow{M}_s p'$ is ($Cons^s_1$), then $p = p' = D$ for some $D \in \mathscr{T}cons$, and $M = \{(D,\emptyset)\}$. In this case, $dec(p) = \{D\} = dec(p')$, and $dec(p) \xrightarrow{M}_s dec(p')$ in $cSMG(Net(p))$ by Definition 3.27, as D is a testable place in $Net(p)$.

- *Proof of 2: If $dec(p) \xrightarrow{M}_s m$ in $cSMG(Net(p))$, then this can be due only to one of the following two cases.*
  (i) If there exists a step G such that $\{p\}[G\rangle m$, then G is a singleton $\{t\}$, with $t = (\{p\}, I, \sigma, dec(p'))$ for some suitable I, σ and p', the last one because, by Proposition 8.18, the post-set of any derivable net transition is admissible and, by Theorem 8.3, any admissible marking is complete. The proof of the net transition t can be mimicked by the corresponding rules in Table 8.5 to produce the transition $p \xrightarrow{\{(\sigma,I)\}}_s p'$, as required.
  (ii) If $p = D$ for some $D \in \mathscr{T}cons$, then $m = dec(p) = \{D\}$ and the transition $dec(p) \xrightarrow{\{(D,\emptyset)\}}_s dec(p)$ is in $cSMG(Net(p))$. By rule ($Cons^s_1$), also $D \xrightarrow{\{(D,\emptyset)\}}_s D$, as required.

- *If $p = p_1 \,|\, p_2$, then $dec(p) = dec(p_1) \oplus dec(p_2)$.*

  - *Proof of 1: By additional induction on the proof of $p \xrightarrow{M}_s p'$. The following four cases are possible:*
    (i) $p_1 \xrightarrow{M}_s p'_1$ and $nct(M, p_2)$, so that rule (Par^s) is applicable with $p' = p'_1 \,|\, p_2$. In this case, by induction, $dec(p_1) \xrightarrow{M}_s dec(p'_1)$ in $cSMG(Net(p))$. Moreover, if $dec(p_2) \xrightarrow{\{(D,\emptyset)\}}_s$ for some $(D,\emptyset) \in M$, then by induction, we can conclude that $p_2 \xrightarrow{\{(D,\emptyset)\}}_s$, which is impossible because $nct(M, p_2)$ holds. Therefore, $dec(p_2) \not\xrightarrow{\{(D,\emptyset)\}}_s$ for all $(D,\emptyset) \in M$. This means that none of the testable places in M is active in $dec(p_2)$, so that in $cSMG(Net(p))$ there is also
    $dec(p) = dec(p_1) \oplus dec(p_2) \xrightarrow{M}_s dec(p'_1) \oplus dec(p_2) = dec(p')$.
    (ii) $p_1 \xrightarrow{M_1}_s p'_1$, $p_2 \xrightarrow{M_2}_s p'_2$, $p' = p'_1 \,|\, p'_2$, $nct(M_1, p_2)$, $nct(M_1, p'_2)$, $nct(M_2, p_1)$ and $nct(M_2, p'_1)$, so that rule (M-Par^s) is applicable. In this case, by induction, $dec(p_1) \xrightarrow{M_1}_s dec(p'_1)$ and $dec(p_2) \xrightarrow{M_2}_s dec(p'_2)$ are in $cSMG(Net(p))$. By an argument similar to the above, the satisfaction of the contextual checks ensures that $dec(p_2) \not\xrightarrow{\{(D_1,\emptyset)\}}_s$ and $dec(p'_2) \not\xrightarrow{\{(D_1,\emptyset)\}}_s$ for all $(D_1,\emptyset) \in M_1$, as well as $dec(p_1) \not\xrightarrow{\{(D_2,\emptyset)\}}_s$ and $dec(p'_1) \not\xrightarrow{\{(D_2,\emptyset)\}}_s$ for all $(D_2,\emptyset) \in M_2$. This means that the steps G_1 and G_2 underlying M_1 and M_2, respectively, can be joined together by the definition of step for NP/T nets. Hence, in $cSMG(Net(p))$ there is also $dec(p) = dec(p_1) \oplus dec(p_2) \xrightarrow{M_1 \oplus M_2}_s dec(p'_1) \oplus dec(p_2) = dec(p')$.
    (iii) $p_1 \xrightarrow{\{(\sigma_1,I)\}}_s p'_1$, $p_2 \xrightarrow{\{(\overline{\gamma},\emptyset)\}}_s p'_2$, $p' = p'_1 \,|\, p'_2$, p_2 is sequential, $Sync(\sigma_1, \overline{\gamma}, \sigma)$ and $M = \{(\sigma, I)\}$, so that rule (S-Com^s) is applicable. By induction, we have that

$dec(p_1) \xrightarrow{\{(\sigma_1,I)\}}_s dec(p_1')$ and $dec(p_2) \xrightarrow{\{(\overline{\gamma},\emptyset)\}}_s dec(p_2')$ are in $cSMG(Net(p))$. By Definition 3.27, a transition $t_1 = ({}^\bullet t_1, I, \sigma_1, t_1^\bullet)$ such that $dec(p_1)[\{t_1\}\rangle dec(p_1')$ exists, and also a transition $t_2 = ({}^\bullet t_2, \emptyset, \overline{\gamma}, t_2^\bullet)$ such that $dec(p_2)[\{t_2\}\rangle dec(p_2')$; since $dec(p_2) = \{p_2\}$, it follows that ${}^\bullet t_2 = \{p_2\}$ and $t_2^\bullet = dec(p_2')$. By rule (s-com), the net transition $t = ({}^\bullet t_1 \oplus {}^\bullet t_2, I, \sigma,\ t_1^\bullet \oplus t_2^\bullet)$ is also derivable,[10] so that $dec(p_1) \oplus dec(p_2)[\{t\}\rangle dec(p_1') \oplus dec(p_2')$. Therefore, as required, $dec(p) = dec(p_1) \oplus dec(p_2) \xrightarrow{M}_s dec(p_1') \oplus dec(p_2') = dec(p')$.

(iv) $p \equiv q$, $q \xrightarrow{M}_s q'$ and $q' \equiv p'$, so that rule ($Cong^s$) is applicable. By induction, we know that $dec(q) \xrightarrow{M}_s dec(q')$ and so the thesis follows because $dec(p) = dec(q)$ and $dec(p') = dec(q')$.

- Proof of 2: The following five cases are possible:

  (i) $dec(p_1) \xrightarrow{M}_s dec(p_1')$ in $cSMG(Net(p))$ and, by induction, $p_1 \xrightarrow{M}_s p_1'$. By Definition 3.27, two cases are possible: either $dec(p_1) = dec(p_1') = m_1$, $p_1 = p_1'$, $M \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$ and $M((D,\emptyset)) \leq m_1(D)$ for all $D \in \mathscr{T}cons$; or there exists a step G such that $dec(p_1)[G\rangle dec(p_1')$, $M = M_1 \oplus M_2$, $M_1 = {}^\circ l(G)$, $M_2 \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$, $m_2 = dec(p_1) \ominus {}^\bullet G$ and $M_2((D,\emptyset)) \leq m_2(D)$ for all $D \in \mathscr{T}cons$. In the former case, as all the labels in M are of the form $(D,\emptyset)$ and $I(M) = \emptyset$ so $nct(M,p_2)$ vacuously holds, the step transition $p \xrightarrow{M}_s p$ is derivable by rule (Par^s). In the latter case, since $m[G\rangle m'$, with $m = dec(p)$ and $m' = dec(p')$, it is required that for all $D \in {}^\circ G$, $m(D) = {}^\bullet G(D)$; consequently, $dec(p_2)(D) = 0$ for all $D \in {}^\circ G$; hence, $nct(M,p_2)$ holds also in this case. Therefore, the step transition $p_1 \,|\, p_2 \xrightarrow{M}_s p_1' \,|\, p_2$ is derivable by rule (Par^s).

  (ii) $dec(p_2) \xrightarrow{M}_s dec(p_2')$ in $cSMG(Net(p))$ and, by induction $p_2 \xrightarrow{M}_s p_2'$. This is the symmetric case of the above, hence omitted.

  (iii) $dec(p_1) \xrightarrow{M_1}_s dec(p_1')$, $dec(p_2) \xrightarrow{M_2}_s dec(p_2')$, $M = M_1 \oplus M_2$ and, by induction, $p_1 \xrightarrow{M_1}_s p_1'$ and $p_2 \xrightarrow{M_2}_s p_2'$. By Definition 3.27, four subcases are possible:

  1. $dec(p_1) = dec(p_1') = m_1$, $p_1 = p_1'$, $M_1 \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$ and, for all $D \in \mathscr{T}cons$, $M_1((D,\emptyset)) \leq m_1(D)$; and $dec(p_2) = dec(p_2') = m_2$, $p_2 = p_2'$, $M_2 \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$ and $M_2((D,\emptyset)) \leq m_2(D)$ for all $D \in \mathscr{T}cons$.
     In this case, since $I(M_1) = I(M_2) = \emptyset$, $nct(M_1,p_2)$ and $nct(M_2,p_1)$ hold. Therefore, $p_1 \,|\, p_2 \xrightarrow{M_1 \oplus M_2}_s p_1 \,|\, p_2$ is derivable by rule (M-Par^s).
  2. $dec(p_1) = dec(p_1') = m_1$, $p_1 = p_1'$, $M_1 \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$ and, for all $D \in \mathscr{T}cons$, $M_1((D,\emptyset)) \leq m_1(D)$; and there exists a step G_2 such that $dec(p_2)[G_2\rangle dec(p_2')$, $M_2 = M_2' \oplus M_2''$, $M_2' = {}^\circ l(G_2)$, $M_2'' \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$, $m_2 = dec(p_2) \ominus {}^\bullet G_2$ and $M_2''((D,\emptyset)) \leq m_2(D)$ for all $D \in \mathscr{T}cons$; and for all $D \in {}^\circ G_2$, $m(D) = {}^\bullet G_2(D)$, where $m = dec(p)$.
     In this case, as $I(M_1) = \emptyset$, it follows that $nct(M_1,p_2)$ and $nct(M_1,p_2')$ hold. Moreover, as for all $D \in {}^\circ G_2$, $m(D) = {}^\bullet G_2(D)$, where $m = dec(p)$, it fol-

[10] Note that $ad({}^\bullet t_1 \oplus {}^\bullet t_2)$ holds, because ${}^\bullet t_1 \oplus {}^\bullet t_2 \subseteq dec(p_1) \oplus dec(p_2)$, which is an admissible marking.

lows that $m_1(D) = 0$ for all $D \in {}^\circ G_2$, and so $nct(M_2, p_1)$ holds. Therefore, $p_1 \mid p_2 \xrightarrow{M_1 \oplus M_2}_s p_1 \mid p_2'$ is derivable by rule (M-Par^s).

3. there exists a step G_1 such that $dec(p_1)[G_1\rangle dec(p_1')$, $M_1 = M_1' \oplus M_1''$, $M_1' = {}^\circ l(G_1)$, $M_1'' \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$, $m_1 = dec(p_1) \ominus {}^\bullet G_1$ and $M_1''((D,\emptyset)) \leq m_1(D)$ for all $D \in \mathscr{T}cons$; and for all $D \in {}^\circ G_1$, $m(D) = {}^\bullet G_1(D)$, where $m = dec(p)$; and $dec(p_2) = dec(p_2') = m_2$, $p_2 = p_2'$, $M_2 \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$ and $M_2((D,\emptyset)) \leq m_2(D)$ for all $D \in \mathscr{T}cons$.
   This case is symmetric to the above, hence omitted.
4. there exists a step G_1 such that $dec(p_1)[G_1\rangle dec(p_1')$, $M_1 = M_1' \oplus M_1''$, $M_1' = {}^\circ l(G_1)$, $M_1'' \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$, $m_1 = dec(p_1) \ominus {}^\bullet G_1$ and $M_1''((D,\emptyset)) \leq m_1(D)$ for all $D \in \mathscr{T}cons$; and for all $D \in {}^\circ G_1$, $m(D) = {}^\bullet G_1(D)$, where $m = dec(p)$; and there exists a step G_2 such that $dec(p_2)[G_2\rangle dec(p_2')$, $M_2 = M_2' \oplus M_2''$, $M_2' = {}^\circ l(G_2)$, $M_2'' \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$, $m_2 = dec(p_2) \ominus {}^\bullet G_2$ and $M_2''((D,\emptyset)) \leq m_2(D)$ for all $D \in \mathscr{T}cons$; and for all $D \in {}^\circ G_2$, $m(D) = {}^\bullet G_2(D)$, where $m = dec(p)$; and additionally, $dom(G_1^\bullet) \cap {}^\circ G_2 = \emptyset$ and $dom(G_2^\bullet) \cap {}^\circ G_1 = \emptyset$.
   In this case, as for all $D \in {}^\circ G_2$, $m(D) = {}^\bullet G_2(D)$, where $m = dec(p)$, it follows that $m_1(D) = 0$ for all $D \in {}^\circ G_2$, and so $nct(M_2, p_1)$ holds. Moreover, as $dom(G_1^\bullet) \cap {}^\circ G_2 = \emptyset$, the marking $dec(p_1') = m_1' = (m_1 \ominus {}^\bullet G_1) \oplus G_1^\bullet$ is such that $m_1'(D) = 0$ for all $D \in {}^\circ G_2$, so that $nct(M_2, p_1')$ holds, too. Similarly, as for all $D \in {}^\circ G_1$, $m(D) = {}^\bullet G_1(D)$, where $m = dec(p)$, it follows that $m_2(D) = 0$ for all $D \in {}^\circ G_1$, and so $nct(M_1, p_2)$ holds. And, as $dom(G_2^\bullet) \cap {}^\circ G_1 = \emptyset$, the marking $dec(p_2') = m_2' = (m_2 \ominus {}^\bullet G_2) \oplus G_2^\bullet$ is such that $m_2'(D) = 0$ for all $D \in {}^\circ G_1$, so that $nct(M_1, p_2')$ holds, too. Therefore, rule (M-Par^s) can be applied to derive $p_1 \mid p_2 \xrightarrow{M_1 \oplus M_2}_s p_1' \mid p_2'$, as required.

(iv) $p_1 \mid p_2 \equiv r_1 \mid r_2$, $dec(r_1) \xrightarrow{\{(\sigma_1,I)\}}_s dec(r_1')$, $dec(r_2) \xrightarrow{\{(\overline{\gamma},\emptyset)\}}_s dec(r_2')$, r_2 is sequential, $Sync(\sigma_1, \overline{\gamma}, \sigma)$, $M = \{(\sigma, I)\}$ and $p_1' \mid p_2' \equiv r_1' \mid r_2'$. By induction, we can assume that $r_1 \xrightarrow{\{(\sigma_1,I)\}}_s r_1'$ and $r_2 \xrightarrow{\{(\overline{\gamma},\emptyset)\}}_s r_2'$ are derivable. Therefore, by rule (S-Com^s), $r_1 \mid r_2 \xrightarrow{\{(\sigma,I)\}}_s r_1' \mid r_2'$ is derivable, too; hence, by rule (Cong^s), also $p_1 \mid p_2 \xrightarrow{M}_s p_1' \mid p_2'$, as required.

(v) $p_1 \mid p_2 \equiv r_1 \mid r_2$, $dec(r_1) \xrightarrow{M_1}_s dec(r_1')$, $dec(r_2) \xrightarrow{M_2}_s dec(r_2')$, $M = M_1 \oplus M_2$ and, by induction, $r_1 \xrightarrow{M_1}_s r_1'$ and $r_2 \xrightarrow{M_2}_s r_2'$ are derivable, with $p_1' \mid p_2' \equiv r_1' \mid r_2'$. This case is similar to (iii), since $dec(p_1 \mid p_2) = dec(r_1 \mid r_2)$, and so omitted.

- If $p = (\nu a)q$, then $dec(p) = dec(q)\{a'/a\}$.

  - Proof of 1: By additional induction on the proof of $p \xrightarrow{M}_s p'$. Two cases are possible:

    (i) Rule (S-Res^s) is the last rule used and the premise is $q \xrightarrow{M'}_s q'$, with $p' = (\nu a)q'$, $M = M'\{a'/a\}$ and $a \notin n(M')$. By induction, $dec(q) \xrightarrow{M'}_s dec(q')$ is in $cSMG(Net(p))$. By Definition 3.27, this is possible if either $dec(q) = dec(q') = m'$, $q = q'$, $M' \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$ and $M'((D,\emptyset)) \leq m'(D)$ for *all* $D \in \mathscr{T}cons$*; or there exists a step* G' *such that* $m'[G'\rangle m''$, $M' = M'_1 \oplus M'_2$, $M'_1 = {}^{\circ}l(G')$, $M'_2 \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$, $m_2 = m' \ominus {}^{\bullet}G'$ *and* $M'_2((D,\emptyset)) \leq m_2(D)$ *for all* $D \in \mathscr{T}cons$.

In the former case, the application of the substitution $\{a'/a\}$ *to* $dec(q)$ *relabels accordingly all the constant names and the negated constant names occurring in* $dec(q)$ *so that* $dec(p) = dec(q)\{a'/a\} \xrightarrow{M'\{a'/a\}}_s dec(q)\{a'/a\} = dec(p)$*, as required. In the latter case, as for any* $t' \in G'$*, transition* $t'\{a'/a\} = ({}^{\bullet}t'\{a'/a\}, {}^{\circ}t'\{a'/a\}) \xrightarrow{\sigma\{a'/a\}} t'^{\bullet}\{a'/a\}$ *is derivable by Proposition 8.20 — where* $\sigma\{a'/a\} = \sigma$ *as* $a \notin n(M')$*, if* $\sigma \notin \mathscr{T}cons$*, and* $D\{a'/a\} = D_{\{a'/a\}}$ *— it follows that* $dec(p) = dec(q)\{a'/a\}[G'\{a'/a\}\rangle dec(q')\{a'/a\} = dec((\nu a)q')$ *is also a step, with* ${}^{\circ}l(G'\{a'/a\}) = M'_1\{a'/a\}$*. Moreover, all the labels in* M'_2 *are turned into labels in* $M'_2\{a'/a\}$*, by an argument similar to the above. Hence,* $M = M'_1\{a'/a\} \oplus M'_2\{a'/a\}$ *and*

$dec(p) = dec(q)\{a'/a\} \xrightarrow{M}_s dec(q')\{a'/a\} = dec(p')$*, as required.*

(ii) *Rule (*$Cong^s$*) is the last rule used: this case is similar to the analogous ones above.*

- *Proof of* 2*: If* $dec(p) \xrightarrow{M}_s m$ *in* $cSMG(Net(p))$*, then this can be due only to one of the following two cases.*

  (i) $dec(p) = dec(p') = m$, $M \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$ *and* $M((D,\emptyset)) \leq m(D)$ *for all* $D \in \mathscr{T}cons$*. In this case,* $dec(q) \xrightarrow{M'}_s dec(q)$ *is also derivable, where* $M = M'\{a'/a\}$*. By induction,* $q \xrightarrow{M'}_s q$ *is a derivable step transition, and so, by rule (S-*Res^s*), also* $p \xrightarrow{M}_s p$ *is derivable.*

  (ii) *There exists a step* G *such that* $dec(p) = m[G\rangle m'$, $M = M_1 \oplus M_2$, $M_1 = {}^{\circ}l(G)$, $M_2 \subseteq \mathscr{M}_{fin}(\mathscr{T}cons \times \{\emptyset\})$, $m_2 = m \ominus {}^{\bullet}G$ *and* $M_2((D,\emptyset)) \leq m_2(D)$ *for all* $D \in \mathscr{T}cons$*. In this case,* G *is of the form* $\overline{G}\{a'/a\}$*, by Proposition 8.20; it follows that* $dec(q)[\overline{G}\rangle\overline{m}$*, where* $m = \overline{m}\{a'/a\}$, $M'_1 = {}^{\circ}l(\overline{G})$ *and* $M_1 = M'_1\{a'/a\}$ *because no net transition in* G *is labeled with the restricted action* a'*, and so no action in* $\overline{G}$ *is labeled with action* a*. Moreover,* $M_2 = M'_2\{a'/a\}$ *and so* $dec(q) \xrightarrow{M'_1 \oplus M'_2}_s \overline{m}$*. By induction, there exists* q' *such that* $q \xrightarrow{M'_1 \oplus M'_2}_s q'$*, with* $dec(q') = \overline{m}$*. By rule (S-*Res^s*), also transition* $(\nu a)q \xrightarrow{M}_s (\nu a)q'$ *is derivable, with* $dec((\nu a)q') = dec(q')\{a'/a\} = \overline{m}\{a'/a\} = m$*, as required.* □

Theorem 8.9. *For any* $p \in \mathscr{P}_{NPL}$, $p \sim_{step} dec(p)$*.*

Proof. The relation $R = \{(p, dec(p)) \mid p \in \mathscr{P}_{NPL}\}$ *is a bisimulation between the concrete step transition system, rooted in p, originating from the step semantics of Section 8.3, and the concrete step marking graph* $cSMG(Net(p))$*. The fact that relation R is a bisimulation derives from Proposition 8.26.* □

$$
\begin{aligned}
[\![\mathbf{0}]\!]_I &= (\emptyset,\emptyset,\emptyset,\emptyset)\\
[\![\mu.p]\!]_I &= (S,A,T,\{\mu.p\}) \ \text{ given } [\![p]\!]_I = (S',A',T',dec(p)) \ \text{ and where}\\
&\qquad S = \{\mu.p\} \cup S',\ t = (\{\mu.p\},\emptyset,\mu,dec(p))\\
&\qquad T = (\{t\} \cup T')^{\otimes} \quad A = l(T)\\
[\![C]\!]_I &= (\{C\},\emptyset,\emptyset,\{C\}) \ \text{ if } C \in I\\
[\![C]\!]_I &= (S,A,T,\{C\}) \quad \text{ if } C \notin I, \text{ given } C \doteq p \text{ and } [\![p]\!]_{I\cup\{C\}} = (S',A',T',dec(p))\\
&\qquad S = \{C\} \cup S'', \text{ where}\\
&\qquad S'' = \begin{cases} S' & \exists t \in T'.t^{\bullet}(p) > 0 \ \vee \ p = \mathbf{0},\\ S' \setminus \{p\} & \text{otherwise} \end{cases}\\
&\qquad A = A',\ T = T'' \cup T_1 \text{ where}\\
&\qquad T'' = \begin{cases} T' & \exists t \in T'.t^{\bullet}(p) > 0 \ \vee \ p = \mathbf{0},\\ T' \setminus \{t \in T' \mid {}^{\bullet}t(p) > 0\} & \text{otherwise} \end{cases}\\
&\qquad T_C = \{(n \cdot C \oplus m_1, J, \sigma, m_2) \mid (n \cdot p \oplus m_1, J, \sigma, m_2) \in T', n \geq 1\}\\
&\qquad T_1 = \begin{cases} \{(n \cdot C \oplus m_1, J, \sigma, m_2) \in T_C \mid m_1(p) = 0\} & \text{if } p \notin S,\\ T_C & \text{otherwise} \end{cases}
\end{aligned}
$$

Table 8.8 Denotational net semantics — part I

8.7 Denotational Net Semantics

The extension to NPL of the denotational semantics we have defined for FNM in the previous chapter is not too complex: for all the FNM operators, we have only to cope with the neg-set of transitions generated by the strong prefixing operator, and so the unary operator of *synchronous closure* $-^{\otimes}$ is to be adapted accordingly. The real difference is in the denotational semantics of the strong prefixing operator $\underline{\neg D}.p$ for a negated name $\neg D$; in this case, the net associated with the testable place D is to be included in the resulting net, as D is statically reachable from the initial marking $\{\underline{\neg D}.p\}$. Moreover, the set of testable places is not explicitly dealt with in the following definitions, as it can be simply derived by intersecting the set S of places with the set $\mathscr{T}cons$ of testable constants.

As for FNM, the correspondence with the operational semantics is strictly preserved for restriction-free processes: the operational net $Net(t)$ is exactly the denotational net $[\![t]\!]_{\emptyset}$ we are going to define. On the contrary, for a restricted process $p = (\nu L)t$, we have the weaker result that $Net(p)$ is exactly the *statically reachable subnet* of $[\![p]\!]_{\emptyset}$.

Table 8.8 shows the denotational semantics for the empty process $\mathbf{0}$, for the action prefixing operator and for the constant. It is essentially the same semantics given for FNM, adapted to take care of the neg-set of transitions. About the semantics of the

$$[\![\underline{a}.p]\!]_I = (S,A,T,\{\underline{a}.p\}) \quad \text{given } [\![p]\!]_I = (S',A',T',dec(p)) \text{ and where}$$

$$S = \{\underline{a}.p\} \cup S'', \text{ where}$$

$$S'' = \begin{cases} S' & \exists t \in T'.t^{\bullet}(p) > 0 \vee p = \mathbf{0}, \\ S' \setminus \{p\} & \text{otherwise} \end{cases}$$

$$A = l(T),\ T = (T_1 \cup T'')^{\otimes} \text{ where}$$

$$T_1 = \{(\{\underline{a}.p\}, J, a \diamond \sigma, m) \mid (\{p\}, J, \sigma, m) \in T'\}$$

$$T'' = \begin{cases} T' & \exists t \in T'.t^{\bullet}(p) > 0 \vee p = \mathbf{0}, \\ T' \setminus \{t \in T' \mid {}^{\bullet}t(p) > 0\} & \text{otherwise} \end{cases}$$

$$[\![\underline{\neg D}.p]\!]_I = (S,A,T,\{\underline{\neg D}.p\}) \quad \text{given } [\![p]\!]_I = (S',A',T',dec(p)), [\![D]\!]_I = (\overline{S},\overline{A},\overline{T},\{D\}) \text{ and where}$$

$$S = \{\underline{\neg D}.p\} \cup \overline{S} \cup S'', \text{ where}$$

$$S'' = \begin{cases} S' & \exists t \in T'.t^{\bullet}(p) > 0 \vee p = \mathbf{0}, \\ S' \setminus \{p\} & \text{otherwise} \end{cases}$$

$$A = l(T),\ T = (T_1 \cup T'' \cup \overline{T})^{\otimes} \text{ where}$$

$$T_1 = \{(\{\underline{\neg D}.p\}, J \cup \{D\}, \sigma, m) \mid (\{p\}, J, \sigma, m) \in T'\}$$

$$T'' = \begin{cases} T' & \exists t \in T'.t^{\bullet}(p) > 0 \vee p = \mathbf{0}, \\ T' \setminus \{t \in T' \mid {}^{\bullet}t(p) > 0\} & \text{otherwise} \end{cases}$$

Table 8.9 Denotational net semantics — part II

action prefixing operator, the only non-trivial aspect is that the newly added transition $t = (\{\mu.p\}, \emptyset, \mu, dec(p))$ may be able to synchronize with the transitions in T', possibly triggering other synchronizations involving more transitions in T'. The set of all such transitions is denoted by $(\{t\} \cup T')^{\otimes}$, where the auxiliary operation $-^{\otimes}$ of *synchronous closure* of a set of transitions is defined as follows.

Definition 8.6. (Synchronous closure) Given a finite set $T \subseteq T_{NPL}$ of transitions, we define its *synchronous closure* $T^{\otimes}$ as the least set such that

$$T^{\otimes} = T \cup \{({}^{\bullet}t_1 \oplus {}^{\bullet}t_2, {}^{\circ}t_1, \sigma, t_1^{\bullet} \oplus t_2^{\bullet}) \mid \exists t_1, t_2 \in T^{\otimes} \text{ such that}$$
$$t_1 = ({}^{\bullet}t_1, {}^{\circ}t_1, \sigma_1, t_1^{\bullet}), t_2 = ({}^{\bullet}t_2, \emptyset, \overline{\gamma}, t_2^{\bullet}) \text{ and } Sync(\sigma_1, \overline{\gamma}, \sigma)\}. \ \square$$

As shown in Section 7.7, the set $T^{\otimes}$ can be computed inductively by adding, step by step, new transitions to the base, finite set T. It is possible to prove that, given a finite set $T \subseteq T_{NPL}$, its synchronous closure $T^{\otimes}$ is finite.

The denotational definition for the constant C, with $C \doteq p$ and $C \notin I$, is the same as for FNM: note that, in the definition of the set T_1, the neg-set J of transition $(n \cdot C \oplus m_1, J, \sigma, m_2)$ in T_1 is the same neg-set of $(n \cdot p \oplus m_1, J, \sigma, m_2)$ in T'.

Table 8.9 shows the denotational definitions for the strong prefixing operator. When the strong prefix is an input, this semantics is the same given for FNM,

$$[\![p_1 \,|\, p_2]\!]_I = (S,A,T,m_0) \ \text{ given } [\![p_i]\!]_I = (S_i,A_i,T_i,m_i) \ \text{ for } i=1,2, \text{ and where}$$

$$S = S_1 \cup S_2,\ m_0 = m_1 \oplus m_2,\ T = (T_1 \cup T_2)^{\otimes},\ A = l(T)$$

$$[\![p_1 + p_2]\!]_I = (S,A,T,\{p_1+p_2\}) \ \text{ given } [\![p_i]\!]_I = (S_i,A_i,T_i,dec(p_i)) \ \text{ for } i=1,2, \text{ and where}$$

$$S = \{p_1+p_2\} \cup S'_1 \cup S'_2, \text{ with, for } i=1,2,$$

$$S'_i = \begin{cases} S_i & \exists t \in T_i \text{ such that } t^{\bullet}(p_i) > 0 \ \vee \ p_i = \mathbf{0}, \\ S_i \setminus \{p_i\} & \text{otherwise} \end{cases}$$

$$A = l(T),\ T = (T'_1 \cup T''_1 \cup T'_2 \cup T''_2)^{\otimes} \text{ with, for } i=1,2,$$

$$T'_i = \begin{cases} T_i & \exists t \in T_i \,.\, t^{\bullet}(p_i) > 0 \ \vee \ p_i = \mathbf{0}, \\ T_i \setminus \{t \in T_i \mid {}^{\bullet}t(p_i) > 0\} & \text{otherwise} \end{cases}$$

$$T_i^{+} = \{(n \cdot (p_1+p_2) \oplus m_1, J, \sigma, m_2) \mid (n \cdot p_i \oplus m_1, J, \sigma, m_2) \in T_i, n \geq 1\}$$

$$T''_i = \begin{cases} \{(n \cdot (p_1+p_2) \oplus m_1, J, \sigma, m_2) \in T_i^{+} \mid m_1(p_i) = 0\} & \text{if } p_i \notin S \\ T_i^{+} & \text{otherwise} \end{cases}$$

Table 8.10 Denotational net semantics — part III

adapted to consider the neg-set of transitions in the set T_1. The case when the strong prefix is a negated constant name $\neg D$ is much more complex because, besides the set S'' of places statically reachable from p, it also includes the set $\overline{S}$ of places of the net $[\![D]\!]_I$, as D is used by the strong prefixing operator as an inhibiting place and so all of its places are statically reachable from $\{D\}$. The set T_1 is computed by turning each transition $(\{p\}, J, \sigma, m) \in T'$ into the corresponding transition $(\{\underline{\neg D}.p\}, J \cup \{D\}, \sigma, m) \in T_1$, where the neg-set J has been enlarged to include also D. The set T'' is the set of transitions statically reachable from S'' and $\overline{T}$ is the set of transitions statically reachable from $\{D\}$. Therefore, the resulting set T of transitions statically reachable from $\{\underline{\neg D}.p\}$ is the synchronous closure of $T_1 \cup T'' \cup \overline{T}$.

Table 8.10 shows the denotational semantics for the parallel operator and for the choice operator. The definition for parallel composition is exactly the same given for FNM. The denotational semantics of $p_1 + p_2$ is also very similar to that for FNM; the only difference is that the set T''_i, for $i = 1,2$, is to be adapted to consider the neg-set of transitions.

Example 8.11. Continuing Examples 8.3 and 8.8 about the counter,

$$C \doteq \underline{\neg D}.zero.C + inc.(C \,|\, D),$$
$$D \doteq dec.\mathbf{0},$$

let us compute $[\![C]\!]_{\emptyset}$. First, the net $[\![C]\!]_{\{C\}}$ is $(\{C\}, \emptyset, \emptyset, \{C\})$, and the net $[\![D]\!]_{\{C\}}$ is $(\{D\}, \{dec\}, \{(\{D\}, \emptyset, dec, \emptyset)\}, \{D\})$.

Hence, the net $[\![C \,|\, D]\!]_{\{C\}}$ is $(\{C, D\}, \{dec\}, \{(\{D\}, \emptyset, dec, \emptyset)\}, \{C, D\})$.

By action prefixing, we get $[\![inc.(C \,|\, D)]\!]_{\{C\}} = (S_1, A_1, T_1, \{inc.(C \,|\, D)\})$, where

$$\begin{aligned} S_1 &= \{inc.(C\,|\,D), C, D\},\\ A_1 &= \{inc, dec\},\\ T_1 &= \{(\{inc.(C\,|\,D)\}, \emptyset, inc, \{C,D\}), (\{D\}, \emptyset, dec, \emptyset)\}. \end{aligned}$$

Similarly, the net $[\![zero.C]\!]_{\{C\}}$ is $(\{zero.C, C\}, \{zero\}, \{(\{zero.C\}, \emptyset, zero, \{C\})\}, \{zero.C\})$, and $[\![D]\!]_{\{C\}}$ is $(\{D\}, \{dec\}, \{(\{D\}, \emptyset, dec, \emptyset)\}, \{D\})$; the latter net is necessary to compute $[\![\underline{\neg D}.zero.C]\!]_{\{C\}}$ because of the strong prefix $\neg D$. Therefore, $[\![\underline{\neg D}.zero.C]\!]_{\{C\}} = (S_2, A_2, T_2, \{\underline{\neg D}.zero.C\})$, where

$$\begin{aligned} S_2 &= \{\underline{\neg D}.zero.C, C, D\},\\ A_2 &= \{zero, dec\},\\ T_2 &= \{(\{\underline{\neg D}.zero.C\}, \{D\}, zero, \{C\}), (\{D\}, \emptyset, dec, \emptyset)\}. \end{aligned}$$

Now, for $p = \underline{\neg D}.zero.C + inc.(C\,|\,D)$, the net $[\![p]\!]_{\{C\}}$ is $(S', A, T', \{p\})$, where

$$\begin{aligned} S' &= \{p, C, D\},\\ A &= \{inc, dec, zero\},\\ T' &= \{(\{p\}, \emptyset, inc, \{C,D\}), (\{p\}, \{D\}, zero, \{C\}), (\{D\}, \emptyset, dec, \emptyset)\}, \end{aligned}$$

and finally, $[\![C]\!]_\emptyset = (S, A, T, \{C\})$, where

$$\begin{aligned} S &= \{C, D\},\\ T &= \{(\{C\}, \emptyset, inc, \{C,D\}), (\{C\}, \{D\}, zero, \{C\}), (\{D\}, \emptyset, dec, \emptyset)\}. \end{aligned}$$

The resulting NP/T net is depicted in Figure 8.1, and it is exactly the net computed operationally in Example 8.8. □

In order to prove that, for any restriction-free NPL process p, the operational net $Net(p)$ is exactly the same as the denotational net $[\![p]\!]_\emptyset$, we need six auxiliary lemmata, dealing with $Net(p, I)$, i.e., the operational net associated with p, assuming that the constants in I are undefined. These lemmata are similar to the corresponding ones for FNM, hence their proofs are omitted, except for the case of strong prefixing with a negated constant name.

Lemma 8.11. *For any restriction-free process p, let $Net(p, I) = (S', A', T', dec(p))$. It follows that $Net(\mu.p, I) = (S, A, T, \{\mu.p\})$, where S, A and T are defined as in Table 8.8.* □

Lemma 8.12. *For any sequential NPL process p and constant C such that $C \doteq p$, let $Net(p, I \cup \{C\}) = (S', A', T', dec(p))$. It follows that $Net(C, I) = (S, A, T, \{C\})$, where S, A and T are defined as in Table 8.8.* □

Lemma 8.13. *For any sequential NPL process p, let $Net(p, I) = (S', A', T', dec(p))$. It follows that $Net(\underline{a}.p, I) = (S, A, T, \{\underline{a}.p\})$, where S, A and T are defined as in Table 8.9.* □

Lemma 8.14. *For any sequential process p, let $Net(p, I) = (S', A', T', dec(p))$; for any testable constant D, let $Net(D, I) = (\overline{S}, \overline{A}, \overline{T}, \{D\})$. It follows that $Net(\underline{\neg D}.p, I) = (S, A, T, \{\underline{\neg D}.p\})$, where S, A and T are defined as in Table 8.9.*

Proof. *By rule (s-pref$_2$), the only transitions statically enabled at the initial marking* $\{\underline{\neg D}.p\}$ *are those with premise of the form* $(\{p\}, J, \sigma, m)$*. These transitions constitute the set* T_1*, where* D *has been added to the neg-set. Therefore, all the places in* $dom(m)$ *are (statically) reachable from the initial marking* $\{\underline{\neg D}.p\}$ *and so all the places in* S' *are statically reachable as well, except possibly the place* p *itself, as* $Net(p)$ *is statically reduced (definition of the set* S''*). Moreover, as* D *is statically reachable, all the places in the set* $\overline{S}$ *are reachable. Therefore,* $S = \{\underline{\neg D}.p\} \cup \overline{S} \cup S''$*. Now, since* $Net(\underline{\neg D}.p)$ *is statically reduced,* T *is the set of all the transitions that are statically enabled at* S*. Set* T'' *contains all the transitions statically enabled at* S''*, while* $\overline{T}$ *contains all the transitions statically enabled at* $\overline{S}$*. So all the transitions statically enabled at* S *are exactly the set* $(T_1 \cup T'' \cup \overline{T})^{\otimes}$*, as required.* □

Lemma 8.15. *For any restriction-free NPL processes* p_1 *and* p_2*, let* $Net(p_i, I) = (S_i, A_i, T_i, m_i)$*, for* $i = 1,2$*. It follows that* $Net(p_1 \mid p_2, I) = (S, A, T, m_0)$ *where* S, A, T *and* m_0 *are defined as in Table 8.10.* □

Lemma 8.16. *For any sequential processes* p_1 *and* p_2*, let* $Net(p_i, I) = (S_i, A_i, T_i, m_i)$*, for* $i = 1,2$*. It follows that* $Net(p_1 + p_2, I) = (S, A, T, \{p_1 + p_2\})$ *where* S, A *and* T *are defined as in Table 8.10.* □

Now we are ready to state the correspondence theorem, stating that for any restriction-free process p, the net computed by the operational semantics is exactly the same net computed by the denotational semantics.

Theorem 8.10. *For any restriction-free NPL process* t*,* $Net(t) = [\![t]\!]_{\emptyset}$*.*

Proof. *By induction on the definitions of* $[\![t]\!]_I$ *and* $Net(p, I)$*, where the latter is the operational net associated with* p*, assuming that the constants in* I *are undefined. Then, the thesis follows for* $I = \emptyset$*.*

The first base case is for $\mathbf{0}$ *and the thesis is obvious, as, for any* $I \subseteq \mathscr{C}ons$*,* $[\![\mathbf{0}]\!]_I = (\emptyset, \emptyset, \emptyset, \emptyset) = Net(\mathbf{0}, I)$*, because no place/token is available and so no transition is derivable from* $\emptyset$ *by the operational rules in Table 8.7. The second base case is for* C *when* $C \in I$*, which corresponds to the case when* C *is not defined. In this case, for any* $I \subseteq \mathscr{C}ons$ *with* $C \in I$*,* $[\![C]\!]_I = (\{C\}, \emptyset, \emptyset, \{C\}) = Net(C, I)$*, because, being undefined, no transition is derivable from* C *(rule (cons) is not applicable). The six inductive cases are as follows.*

Action prefixing*: If* $t = \mu.p$*, then, by induction, we can assume that* $Net(p, I) = (S', A', T', dec(p)) = [\![p]\!]_I$*. The thesis* $Net(\mu.p, I) = [\![\mu.p]\!]_I$ *follows by Lemma 8.11.*

Constant*: In this case,* $t = C$ *and we assume that* $C \doteq p$ *and* $C \notin I$*. By induction, we have that* $Net(p, I \cup \{C\}) = (S', A', T', dec(p)) = [\![p]\!]_{I \cup \{C\}}$*, so that in* p *constant* C *is undefined. The thesis* $Net(C, I) = [\![C]\!]_I$ *follows by Lemma 8.12.*

Strong prefixing 1*: If* $t = \underline{a}.p$*, then, by induction, we can assume that* $Net(p, I) = (S', A', T', dec(p)) = [\![p]\!]_I$*. The thesis* $Net(\underline{a}.p, I) = [\![\underline{a}.p]\!]_I$ *follows by Lemma 8.13.*

Strong prefixing 2*: If* $t = \underline{\neg D}.p$*, then, by induction, we can assume that* $Net(p, I) = (S', A', T', dec(p)) = [\![p]\!]_I$ *and* $Net(D, I) = (\overline{S}, \overline{A}, \overline{T}, \{D\}) = [\![D]\!]_I$*. Then, by Lemma 8.14, the thesis* $Net(\underline{\neg D}.p, I) = [\![\underline{\neg D}.p]\!]_I$ *follows.*

$$\begin{aligned}
[\![(\nu a)p]\!]_I &= ([\![p]\!]_I\{a'/a\}) \setminus a' = (S\{a'/a\}, (A\{a'/a\}) \setminus a', (T\{a'/a\}) \setminus a', m_0\{a'/a\}) \\
&\qquad \text{given } [\![p]\!]_I = (S, A, T, m_0) \text{ where} \\
S\{a'/a\} &= \{s\{a'/a\} \mid s \in S\} \\
A\{a'/a\} &= \{\sigma\{a'/a\} \mid \sigma \in A\} \\
(A') \setminus a' &= \{\sigma \in A' \mid a' \notin n(\sigma)\} \\
T\{a'/a\} &= \{(m_1\{a'/a\}, J\{a'/a\}, \sigma\{a'/a\}, m_2\{a'/a\}) \mid (m_1, J, \sigma, m_2) \in T\} \\
(T') \setminus a' &= \{(m_1, J, \sigma, m_2) \mid (m_1, J, \sigma, m_2) \in T' \wedge a' \notin n(\sigma)\}
\end{aligned}$$

Table 8.11 Denotational net semantics — part IV

Parallel*: By induction, we can assume that* $Net(p_i, I) = (S_i, A_i, T_i, dec(p_i)) = [\![p_i]\!]_I$ *for* $i = 1, 2$*. The thesis* $Net(p_1 \mid p_2, I) = [\![p_1 \mid p_2]\!]_I$ *follows by Lemma 8.15.*

Choice*: By induction, we can assume that* $Net(p_i, I) = (S_i, A_i, T_i, dec(p_i)) = [\![p_i]\!]_I$ *for* $i = 1, 2$*. The thesis* $Net(p_1 + p_2, I) = [\![p_1 + p_2]\!]_I$ *follows by Lemma 8.16.* □

The denotational semantics for the restriction operator is described in Table 8.11 and is very similar to that for FNM. The net $[\![(\nu a)p]\!]_I$ is essentially computed in two steps starting from the net $[\![p]\!]_I$: first, the substitution $\{a'/a\}$ is applied elementwise to all the places and transitions of the net for p; then, a *pruning* operation $- \setminus a'$ is performed to remove all the relabeled transitions that are now labeled with sequences containing occurrences of the restricted action a' or $\overline{a'}$. Differently from the previous cases, now the places are *extended* sequential NPL processes.

We are going to state that the statically reachable subnet of the denotational semantics net is the operational semantics net for any restricted NPL process.

Lemma 8.17. *For any NPL process p, let* $Net(p) = (S, A, T, m_0)$*. It follows that* $Net((\nu a)p) = Net_s((Net(p)\{a'/a\}) \setminus a')$*, where* $(Net(p)\{a'/a\}) \setminus a' = (S\{a'/a\}, (A\{a'/a\}) \setminus a', (T\{a'/a\}) \setminus a', m_0\{a'/a\})$*, and the operations of substitution are pruning are defined in Table 8.11.*

Proof. By induction on the static reachability relation $\Longrightarrow^*$*, we can prove that the set* $S_{(\nu a)p}$ *of the statically reachable places of* $Net((\nu a)p)$ *is the subset of* $S\{a'/a\}$ *of the places statically reachable from* $m_0\{a'/a\}$ *in* $(Net(p)\{a'/a\}) \setminus a'$*. And vice versa, the places statically reachable from* $m_0\{a'/a\}$ *in* $(Net(p)\{a'/a\}) \setminus a'$ *constitute exactly the set* $S_{(\nu a)p}$*. The details of the proof are very similar to the analogous Lemmata 6.12 and 7.18, and so omitted.* □

Theorem 8.11. *For any NPL process p,* $Net(p) = Net_s([\![p]\!]_\emptyset)$*, i.e.,* $Net(p)$ *is exactly the statically reachable subnet of* $[\![p]\!]_\emptyset$*.*

Proof. An NPL process p is either a restriction-free process t, or there exists an action a and an NPL process p' *such that* $p = (\nu a)p'$*. In the former case,* $Net(t) =$

$[\![t]\!]_\emptyset$ by Theorem 8.10, so that the thesis follows trivially, because $[\![t]\!]_\emptyset$ is statically reduced: $Net_s([\![t]\!]_\emptyset) = [\![t]\!]_\emptyset$.

In the latter case, we can assume, by induction, that $Net(p') = Net_s([\![p']\!]_\emptyset)$. If $Net(p') = (S, A, T, m_0)$, then $Net((\nu a)p') = Net_s((Net(p')\{a'/a\}) \setminus a')$, by Lemma 8.17. By definition in Table 8.11, $[\![(\nu a)p']\!]_\emptyset = ([\![p']\!]_\emptyset\{a'/a\}) \setminus a'$. Moreover, it holds trivially that $Net_s(N(m_0)) = Net_s(Net_s(N(m_0)))$ for any net $N(m_0)$. Therefore,

$$\begin{aligned} Net_s([\![(\nu a)p']\!]_\emptyset) &= Net_s(([\![p']\!]_\emptyset\{a'/a\}) \setminus a') \\ &= Net_s((Net_s([\![p']\!]_\emptyset)\{a'/a\}) \setminus a') \\ &= Net_s((Net(p')\{a'/a\}) \setminus a') \\ &= Net((\nu a)p'). \end{aligned} \qquad \square$$

8.8 RNPL

Regular NPL — RNPL, for short — is the calculus whose terms are generated from actions, process constants and negated constant names as described by the following abstract syntax:

$$\begin{array}{llllllll} s & ::= a.r & | \quad \tau.r & | \; \underline{a}.s & | \; \underline{\neg D}.s & | \; s+s & & \\ q & ::= \; s & | \quad \mathbf{0} & | \; \bar{a}.r & | \; q+q & & & \\ r & ::= \; q & | \quad C & & & & & \textit{sequential terms} \\ t & ::= \; r & | \quad t \,|\, t & & & & & \textit{restriction-free terms} \\ p & ::= \; t & | \; (\nu a)p & & & & & \textit{general terms} \end{array}$$

where $a \in \mathscr{L}$, $\bar{a} \in \overline{\mathscr{L}}$, $\neg D \in \neg\mathscr{T}cons$ and any constant C is equipped with a defining equation in syntactic category q. The only difference with respect to NPL is that the action prefixing operator $\mu.-$ is applied to processes of category r, instead of category t.

With respect to the LTS semantics, RNPL processes generate finite-state abstract LTS, labeled over $\mathscr{A} = \{\tau\} \cup \overline{\mathscr{L}} \cup \mathscr{L}^+$. This fact can be proved by structural induction, with the base case given by sequential processes, as done for RMCS in Section 7.8.

Proposition 8.27. (Finite-state LTS) *For any RNPL process p, the LTS reachable from p, $\mathscr{C}_p = (\mathscr{P}_p, sort(p), \rightarrow_p, p)$, is finite-state.* □

With respect to the net semantics, RNPL is more expressive than RMCS, because it represents, besides all the concurrent FSMs and many finite P/T nets, also many finite NP/T nets. For instance, the net in Figure 3.20(a), originating from the RNPL process p of Example 8.9, is a finite NP/T net and not a P/T net.

Now we want to argue that, for any RNPL process p, $Net(p)$ is a *bounded*, finite NP/T net. By Theorem 8.7, any NPL process p is such that $Net(p)$ is a finite NP/T net; hence, since RNPL is a subcalculus of NPL, $Net(p)$ is a finite NP/T net for any RNPL process p, too. So, it remains to prove that such a net is bounded. By syntactic definition, the basic constituents of any RNPL process $p = (\nu L)(p_1 \mid \dots \mid p_n)$ are the processes $p_1, \dots, p_n$ in syntactic category r. Of course, the net transitions that any sequential p_i may generate have a singleton pre-set and a post-set which is a singleton at most. By rule (s-com), two net transitions t_1 and t_2 can be synchronized, producing a transition t with pre-set ${}^\bullet t = {}^\bullet t_1 \oplus {}^\bullet t_2$ and post-set $t^\bullet = t_1^\bullet \oplus t_2^\bullet$, so that $|{}^\bullet t| = 2$ and $|t^\bullet| \leq 2$; of course, t can be used as a premise for rule (s-com) to generate a new transition with pre-set of size three and a post-set of size not larger than three; however, the number of applications of rule (s-com) is bounded by the length of the longest atomic sequence that any p_i can generate. Hence, if j is the length of the longest sequence of all the p_i's, the generable net transitions have a pre-set of size $j+1$ and a post-set of size $j+1$ at most. Summing up, for any m and any t, if $m[t\rangle m'$, then $|m'| \leq |m|$, because t does not produce more tokens than it consumes. Therefore, the following theorem holds.

Theorem 8.12. *For any process p of RNPL, $Net(p)$ is a k-bounded, finite NP/T net, where $k = |dec(p)|$.* □

However, RNPL is not able to express all the possible bounded, finite NP/T nets. It is enough to consider a finite NP/T net with a transition of the form $(\{s_1\}, \emptyset, a, \{s_2, s_3\})$, which produces more tokens than it consumes.

Chapter 9
Generalizations and Variant Semantics

Abstract A few possible extensions and variant semantics are discussed. They are related to (i) representing all Petri nets labeled over the set *Act* of structured actions and co-actions; (ii) allowing the body of recursively defined constants to be non-sequential; (iii) generalizing the approach to the case when the restriction operator can occur inside the body of recursively defined constants. Finally, asynchronous communication is discussed and an open problem for future research is suggested.

9.1 Communicating Petri Nets

Following the tradition in the field of automata theory [HMU01], also in the classic definition of Petri nets (see, e.g., [Pet81, DR98, Rei13]), transitions are labeled with actions taken from a set A of *unstructured* actions; hence, in the thesis of the representability theorems, our assumption that the label set A is a subset of $\mathscr{L} \cup \{\tau\}$ is in analogy with this tradition.

If we want to be more generous and to consider Petri nets labeled over the set $Act = \mathscr{L} \cup \overline{\mathscr{L}} \cup \{\tau\}$ of *structured* actions and co-actions, we obtain the so-called *communicating* Petri nets, which have actually been used as the semantic model for FNC, FNM and also NPL, when the modeled process is not output-closed. However, the extension of the six representability theorems to communicating Petri nets may be non-trivial.

On the one hand, the constructions given for SFM, CFM and BPP can be trivially extended, as the parallel composition operator is either absent or does not allow synchronization. For instance, let us consider the concurrent FSM net $N_1(m_0)$ in Figure 9.1(a), where $m_0 = s_1 \oplus s_2$. We can extract from this net its associated CFM process, namely $p = \mathscr{T}_{CFM}(N_1(m_0)) = C_1 \,|\, C_2$, where $C_1 \doteq a.\mathbf{0}$ and $C_2 \doteq \overline{a}.\mathbf{0}$, whose associated net $Net(p)$ is isomorphic to $N_1(s_1 \oplus s_2)$, as expected.

On the other hand, if we extract from the net $N_1(m_0)$ its associated FNC process, namely $q = \mathscr{T}_{FNC}(N_1(m_0)) = (\nu L)(C_1 \,|\, C_2)$, where $L = \{x_1^1, x_2^1, x_1^2, x_2^2\}$, $C_1 \doteq a.\mathbf{0}$ and $C_2 \doteq \overline{a}.\mathbf{0}$, then its associated net $Net(q)$ is isomorphic to $N_2(s_3 \oplus s_4)$ in Figure

R. Gorrieri, *Process Algebras for Petri Nets*, Monographs in Theoretical
Computer Science. An EATCS Series, DOI 10.1007/978-3-319-55559-1_9

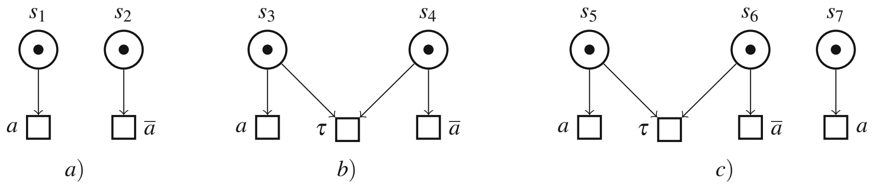


Fig. 9.1 Some simple communicating nets

9.1(b), because now parallel composition enables the synchronization between a and $\overline{a}$. This is an unwanted result.

As a matter of fact, for the process algebras we have studied where parallel composition allows for synchronization, namely FNC, FNM and NPL, the situation is a bit delicate. Let us consider the finite CCS net $N_3(m_0)$ in Figure 9.1(c), where $m_0 = s_5 \oplus s_6 \oplus s_7$. In this case, if we extract the FNC process $p = \mathscr{T}_{FNC}(N_3(m_0))$ from this net according to Definition 6.12, we get

$$p = (\nu x_5^1, x_6^1, x_7^1, x_5^2, x_6^2, x_7^2, x_5^3, x_6^3, x_7^3, x_5^4, x_6^4, x_7^4)(C_5 \,|\, C_6 \,|\, C_7),$$

where $C_5 \doteq a.\mathbf{0} + x_6^2.\mathbf{0}$, $C_6 \doteq \overline{x}_6^2.\mathbf{0} + \overline{a}.\mathbf{0}$ and $C_7 \doteq a.\mathbf{0}$. However, the net $Net(p)$ is not isomorphic to the net $N_3(m_0)$ in Figure 9.1(c), because it generates an additional τ-labeled transition with pre-set $s_6 \oplus s_7$, due to the synchronization of action $\overline{a}$ of C_6 and its complementary action a of C_7.[1]

A possible way out is to extend these calculi with the additional, *external* (i.e., to be used at the top level only) operator of *relabeling* $p[b/a]$, as defined in [GV15] (originally, in a more general form, in [Mil89]), which relabels each occurrence of action a, performed by p, as b. (As for syntactic substitution, the relabeling operator can be generalized to allow for multiple relabelings: $p[b_1/a_1, \ldots, b_n/a_n]$.) The following SOS operational rule

$$(\mathrm{Rel})\ \frac{p \xrightarrow{\sigma} p'}{p[b/a] \xrightarrow{\sigma[b/a]} p'[b/a]}$$

describes the intended LTS semantics of this operator, where the label $\sigma[b/a]$ is defined as follows: $a[b/a] = b$, $\overline{a}[b/a] = \overline{b}$, $c[b/a] = c$, if $c \neq a, \overline{a}$, and $(\alpha\sigma)[b/a] = \alpha[b/a]\sigma[b/a]$.

The problem of defining an operational *unsafe* net semantics for this operator seems hard.[2] On the contrary, a denotational net semantics for this operator is quite easy (see Table 9.1): it is enough to relabel the transitions, which makes sense if

[1] Note that the synchronization between actions a of C_5 and $\overline{a}$ of C_6 generates the same τ-labeled transition with pre-set $s_5 \oplus s_6$ due to the synchronization on the bound action x_6^2, and so it does no harm.

[2] However, note that a 1-safe operational net semantics for the relabeling operator can easily be given, following the approach of Degano, De Nicola and Montanari [DDM88].

$$
\begin{aligned}
[\![p[b/a]]\!]_I &= [\![p]\!]_I[b/a] = (S, A[b/a], T[b/a], m_0)\\
&\qquad\qquad \text{given } [\![p]\!]_I = (S, A, T, m_0) \text{ where}\\
A[b/a] &= \{\sigma[b/a] \mid \sigma \in A\}\\
T[b/a] &= \{(m_1, \sigma[b/a], m_2) \mid (m_1, \sigma, m_2) \in T\}
\end{aligned}
$$

Table 9.1 Denotational net semantics for the relabeling operator

we assume that this operator is used syntactically at the top level only. Then, the procedure is as follows:

- First, take a communicating Petri net $N(m_0)$ labeled with actions taken from Act, which can be a finite CCS net (for FNC), or a finite P/T net (for FNM) or a finite NP/T net (for NPL).
- Then, relabel each transition t_j of the net N labeled with an input action a_j to a new input action a_j^j, yielding a new renamed net $N'(m_0)$. In this way, the set of labels of the net $N'(m_0)$ does not contain any pair of complementary actions/co-actions.
- Then, compute the associated process $p(N'(m_0))$, which is $\mathscr{T}_{FNC}(N'(m_0))$, or $\mathscr{T}_{FNM}(N'(m_0))$, or $\mathscr{T}_{NPL}(N'(m_0))$, according to the chosen type of net and the process algebra of interest. Note that the process $p(N'(m_0))$ does not contain any pair of *free* complementary actions/co-actions, so that $[\![p(N'(m_0))]\!]_\emptyset$ is a net where no additional synchronization net transitions are introduced.
- Then, consider the process $q = p(N'(m_0))[a_1/a_1^1, \ldots, a_k/a_k^k]$ and compute its associated net $[\![q]\!]_\emptyset$, according to the denotational net semantics for the chosen calculus enriched with the relabeling operator; the result is to relabel each transition of $[\![p(N'(m_0))]\!]_\emptyset$ labeled with an input action a_j^j with its original name a_j, for $j = 1, \ldots, k$.

It is not difficult to realize that the statically reachable subnet $Net_s([\![q]\!]_\emptyset)$ of the denotational net $[\![q]\!]_\emptyset$ is isomorphic to the original net $N(m_0)$. For instance, let us consider FNC. Theorem 6.8 ensures that $Net(p(N'(m_0)))$ is isomorphic to $N'(m_0)$, while Theorem 6.11 ensures that $Net(p(N'(m_0)))$ is exactly the statically reachable subnet of $[\![p(N'(m_0))]\!]_\emptyset$; hence, $Net_s([\![p(N'(m_0))]\!]_\emptyset)$ is isomorphic to $N'(m_0)$. Therefore, $Net_s([\![q]\!]_\emptyset)$ must be isomorphic to $N(m_0)$, as the additional step of relabeling reintroduces the original names of the involved input transitions.

9.2 Variant Net Semantics

A peculiar feature which is common to all the calculi studied in this book is that the body p defining a constant C must be a *sequential* process, because, intuitively, a constant is intended to be the name of a sequential process. This aspect of the language definition cannot be changed for NPL, because the negated constant name $\neg D$ is used to test the absence of a *sequential* process, named D. However, for the other calculi, namely CFM, BPP, FNC and FNM, we may be more generous and allow the body p to be any *parallel* process. For instance, the following definition for constant C could be accepted in the enriched version of these calculi:

$$C \doteq a.\mathbf{0} \,|\, b.C.$$

The technical development would not change so much. From a syntactic point of view, we are forced to explicitly define constant *guardedness*, as done for instance in [GV15], and to require that a term q is a process only if all the constants in $Const(q)$ are guarded; in fact, in the current version of these calculi, constant guardedness is ensured by syntactic construction. Moreover, the LTS interleaving semantics for this enriched version of the calculi CFM, BPP and FNC would remain the same, while for the enriched version of FNM it would be necessary to include one additional axiom, generating the structural congruence $\equiv$, of the form: $C \equiv p$ if $C \doteq p$; the effect of this extension is that rule (Cons) can now be omitted (see [Gor16] or Chapter 6 of [GV15] for details). Also the step semantics would require a slight adaptation for rule (Cons^s), whose transition label is not required to be a singleton; for instance, the step transition $C \xrightarrow{\{a,b\}}_s \mathbf{0} \,|\, C$ would be derivable for the constant C defined above.

Also the net semantics would require some changes, the most notable one being that the decomposition of a constant C would be the result of decomposing its body p; formally:

$$dec(C) = dec(p) \quad \text{if } C \doteq p.$$

For instance, for the definition above, $dec(C) = \{a.\mathbf{0}, b.C\}$. This has the effect that a constant C is not a place; as a consequence, there is no need for the specific net transition operational rule (cons) for constants. The finiteness theorems (for any process p, the net $Net(p)$ is a finite net) for the various enriched calculi would still be correct. However, the representability theorems would not always be correct. In fact, as a constant is not a place, it follows that $C_1 \doteq p$ and $C_2 \doteq p$ are mapped via dec to the same multiset $dec(p)$, and so some undesirable fusion of places may occur, as illustrated in the following example about CFM.

Example 9.1. Let us consider the concurrent FSM net $N(m_0) = (\{s_1, s_2, s_3, s_4\}, \{a\}, \{(s_1, a, s_3), (s_2, a, s_4)\}, m_0)$, where $m_0 = \{s_1, s_2\}$, which has four places and two transitions. The CFM term $p = \mathscr{T}_{CFM}(N(m_0))$ would be $C_1 \,|\, C_2$, where

$$\begin{array}{llll} C_1 \doteq a.C_3 + \mathbf{0}, & & C_2 \doteq \mathbf{0} + a.C_4, \\ C_3 \doteq \mathbf{0} + \mathbf{0}, & & C_4 \doteq \mathbf{0} + \mathbf{0}, \end{array}$$

but now $Net(p)$ is the net $(\{a.C_3 + \mathbf{0}, \mathbf{0} + a.C_4, \mathbf{0} + \mathbf{0}\}, \{a\}, \{(a.C_3 + \mathbf{0}, a, \mathbf{0} + \mathbf{0}), (\mathbf{0} + a.C_4, a, \mathbf{0} + \mathbf{0})\}, \{a.C_3 + \mathbf{0}, \mathbf{0} + a.C_4\})$, which has three places only, as the two distinct places s_3 and s_4 are now mapped to the same place $\mathbf{0} + \mathbf{0}$. □

A solution to this drawback is possible for FNC and FNM [Gor16], at the price of introducing some additional new bound names: for each place s_i, the associated constant C_i has a defining equation of the form

$$C_i \doteq c_i^1 + \cdots + c_i^k + y_i.\mathbf{0}$$

where each summand c_i^j, for $j = 1, \ldots, k$ (where k is the number of transitions in N), is defined as usual, but the new bound name y_i is used to keep distinct each constant body, as clarified in the following example.

Example 9.2. Let us consider again the net $N(m_0) = (\{s_1, s_2, s_3, s_4\}, \{a\}, \{(s_1, a, s_3), (s_2, a, s_4)\}, m_0)$, where $m_0 = \{s_1, s_2\}$, which has four places and two transitions. The FNC term $q = \mathscr{T}_{FNC}(N(m_0))$ would be $(\nu L)(C_1 \,|\, C_2)$, where $L = \{x_1^1, x_2^1, x_1^2, x_2^2\} \cup \{y_1, \ldots, y_4\}$ and

$$\begin{array}{llll} C_1 \doteq a.C_3 + \mathbf{0} + y_1, & & C_2 \doteq \mathbf{0} + a.C_4 + y_2, \\ C_3 \doteq \mathbf{0} + \mathbf{0} + y_3, & & C_4 \doteq \mathbf{0} + \mathbf{0} + y_4, \end{array}$$

and now $Net(q)$ is the net $(\{a.C_3 + \mathbf{0} + y_1, \mathbf{0} + a.C_4 + y_2, \mathbf{0} + \mathbf{0} + y_3, \mathbf{0} + \mathbf{0} + y_4\}, \{a\}, \{(a.C_3 + \mathbf{0}, a, \mathbf{0} + \mathbf{0} + y_3), (\mathbf{0} + a.C_4, a, \mathbf{0} + \mathbf{0} + y_4)\}, \{a.C_3 + \mathbf{0} + y_1, \mathbf{0} + a.C_4 + y_2\})$, which is indeed isomorphic to $N(m_0)$. □

Summing up, we think that our choice of requiring that the body of a constant definition be sequential is not only intuitively more attractive, but it gives also rise to a mathematically better theory.

9.3 General Restriction

The restriction operator in CCS [Mil89], in Multi-CCS [GV15] and in the π-calculus [MPW92] is not forced to be used at the top level only, as in FNC and its extensions. In particular, it is possible that this operator occurs within the body of a recursively defined constant. To fix the problem, consider the CCS dialect such that $\mathscr{L}$ is finite (but large enough), the sum operator is guarded and the body of any constant definition is sequential, as assumed for all the calculi studied in this book. Note that in this CCS dialect the prefixing operator $\mu.-$ is applied to any general term and not only to a restriction-free term, as in FNC. This CCS dialect is Turing-complete when at least eight actions are available[3] (namely, $dec_1, dec_2, inc_1, inc_2, zero_1, zero_2, a, b$); as a matter of fact, the modeling of the counter machines in NPL described in Section 8.2.1 can be easily adapted to CCS by replacing the NPL counter of Example 8.3 with the following CCS counter [Tau89]:

[3] It is an open problem to find the least number of constants to get Turing-completeness in CCS, when simulating a universal Turing machine (UTM for short) with a 2-counter machine [Min67]. However, it is not difficult to show that a direct construction of a UTM with n states and m symbols, based on a tape simulated by two stacks (see Example 3.24 in [GV15] for the definition of a stack in CCS), would require $4m + 2 + n$ constants and $4m + 4$ actions [Gor17].

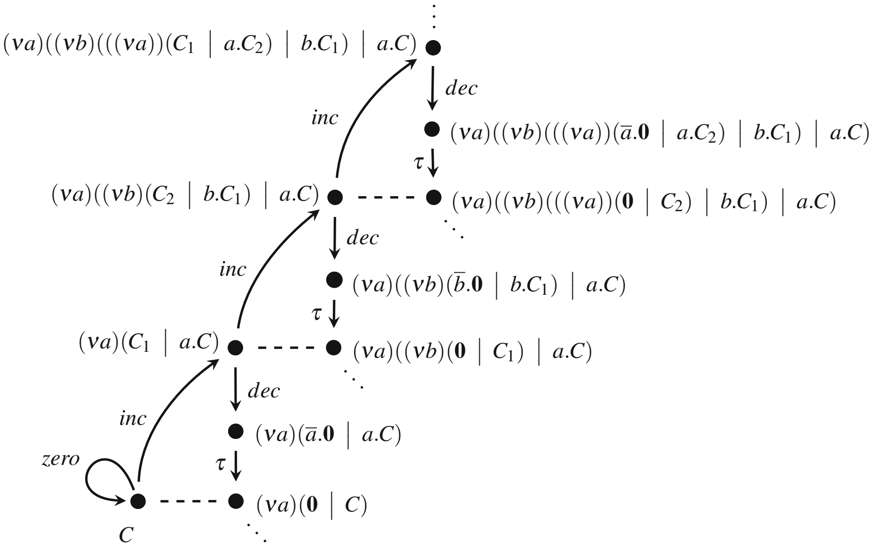

$(\nu a)((\nu b)(((\nu a))(C_1 \mid a.C_2) \mid b.C_1) \mid a.C)$
dec
inc
$(\nu a)((\nu b)(((\nu a))(\bar{a}.\mathbf{0} \mid a.C_2) \mid b.C_1) \mid a.C)$
τ
$(\nu a)((\nu b)(C_2 \mid b.C_1) \mid a.C)$
$(\nu a)((\nu b)(((\nu a))(\mathbf{0} \mid C_2) \mid b.C_1) \mid a.C)$
dec
inc
$(\nu a)((\nu b)(\bar{b}.\mathbf{0} \mid b.C_1) \mid a.C)$
τ
$(\nu a)(C_1 \mid a.C)$
$(\nu a)((\nu b)(\mathbf{0} \mid C_1) \mid a.C)$
dec
inc
$(\nu a)(\bar{a}.\mathbf{0} \mid a.C)$
zero
τ
$(\nu a)(\mathbf{0} \mid C)$
C

Fig. 9.2 The initial part of the infinite LTS of the counter C. (A dashed line connects weakly bisimilar states.)

$$C \doteq zero.C + inc.((\nu a)(C_1 \,|\, a.C)),$$
$$C_1 \doteq dec.\bar{a}.\mathbf{0} + inc.((\nu b)(C_2 \,|\, b.C_1)),$$
$$C_2 \doteq dec.\bar{b}.\mathbf{0} + inc.((\nu a)(C_1 \,|\, a.C_2)),$$

which exploits three recursively defined constants only, and where restriction occurs in the body of all of them. The LTS for C has infinitely many states; the initial fragment of this LTS is outlined in Figure 9.2.

Even if, from a syntactic point of view, the set of bound names of C is finite, namely $bn(C) = \{a, b\}$, the number of different names that such a process may use *semantically* may be unbounded. In fact, the use of nested, alternated occurrences of the restriction operator allows for an unbounded generation of *new* names. To explain the issue, consider the process

$$p = (\nu a)((\nu b)(((\nu a)((\nu b)(C_2 \,|\, b.C_1) \,|\, a.C_2) \,|\, b.C_1)) \,|\, a.C)$$

which is reachable from C by performing action *inc* four times. From a semantical point of view, p is using four different bound names: the rightmost occurrence of a prefix a (in $a.C$) is subject to the leftmost occurrence of the operator (νa), while the leftmost occurrence of a (in $a.C_2$) is subject to the rightmost occurrence of (νa), hence the two occurrences of a are denoting two *different* semantic actions, even if they are both called a. And similarly for the occurrences of prefixes b and operators (νb). Indeed, from a semantical point of view, C may generate unboundedly many different *semantic* bound names, but these are obtained by means of two syntactic bound names only, namely a and b, by a clever alternation of their use.

Therefore, the net semantics we have defined for FNC is to be changed, because we cannot simply assume that for each syntactic bound action a there exists exactly one corresponding restricted action a', because now the *syntactic* bound names are not in bijective correspondence with the *semantic* bound names. However, the extension is not too difficult. It is enough to use a countable set $\mathscr{N} = \{\eta_i \mid i \in \mathbb{N}\}$ of auxiliary *restricted* actions to build the set of extended processes $\mathscr{P}^{\mathscr{N}}$. At a first approximation, the decomposition function dec is to be updated, but only for the restriction operator, as follows:

$$dec((\nu a)q) = dec(q\{\eta_i/a\}) \qquad \eta_i \in \mathscr{N} \text{ is a } \textit{new} \text{ restricted action,}$$

assuming that a mechanism for the generation of a new name is available, whenever necessary. For instance, consider the processes $p = (\nu a)(a.\mathbf{0} \,|\, b.\mathbf{0})$ and $q = (\nu a)(\overline{a}.\mathbf{0} \,|\, b.\mathbf{0})$. Then,

$$\begin{aligned} dec(p\,|\,q) &= dec(p) \oplus dec(q) = dec(a.\mathbf{0}\,|\,b.\mathbf{0})\{\eta_i/a\} \oplus dec(\overline{a}.\mathbf{0}\,|\,b.\mathbf{0})\{\eta_j/a\} \\ &= \eta_i.\mathbf{0} \oplus 2 \cdot b.\mathbf{0} \oplus \overline{\eta}_j.\mathbf{0} \end{aligned}$$

where $i \neq j$. In general, we may need infinitely many different restricted names, as in the case of the counter process C above, when the set of semantic bound names is infinite; in such a case, the resulting P/T net is infinite as well. This basic idea has been applied to giving a net semantics to the π-calculus in [BG09], by using nets with inhibitor arcs; this semantics generates finite nets for so-called *finite-net* (closed) processes, which are exactly those processes where restriction can occur at the top level only, in particular, not in the body of any recursively defined constant.

As observed in [Mor11], this basic idea may be inaccurate in some cases. For instance, consider the process $p = B|B$, where $B \doteq a.(\nu b)(b.c.\mathbf{0} + \overline{b}.\mathbf{0})$. The expected semantics is that p performs two a-labeled transitions and then deadlocks. By applying the decomposition function to p, we would get $dec(p) = \{B, B\}$. By axiom (pref), the net transition $\{B\} \xrightarrow{a} dec((\nu b)(b.c.\mathbf{0} + \overline{b}.\mathbf{0}))$ is derivable, where $dec((\nu b)(b.c.\mathbf{0} + \overline{b}.\mathbf{0})) = \eta_i.c.\mathbf{0} + \overline{\eta}_i.\mathbf{0}$ for some new restricted name η_i. Now, since we have two tokens in place B, after performing the two a-labeled transitions, we have two tokens in place $\eta_i.c.\mathbf{0} + \overline{\eta}_i.\mathbf{0}$, so that now a synchronization is possible, and then a c-labeled transition is executable: this is in contrast with the intended behavior of p. The problem is due to the fact that the two instances of B should be differentiated: as the set of its bound names is not empty, each instance should use a different set of restricted names.

To overcome this problem, [Mor11] suggests an explicit mechanism for generating new names, based on a suitable indexing of the set $\mathscr{N}$ of restricted actions, and decorating each extended sequential process s such that $bn(s) \neq \emptyset$, with a pair of natural numbers $[e, i]$, where i denotes the first available index for a new restricted action, and e the exponent of 2 to be used as the step for computing the next available index (i.e., the next index is $i + 2^e$). Hence, each place in S_{CCS} is a pair (s, δ), where s is an extended sequential process and δ is a decoration, which is empty, denoted by ε, when $bn(s) = \emptyset$. The details of the construction — in the simplified

$$
\begin{aligned}
d(\mathbf{0},e,i,I) &= \varepsilon \\
d(\mu.p,e,i,I) &= d(p,e,i,I) \\
d(p_1+p_2,e,i,I) &= \begin{cases} d(p_1,e+1,i,I) & \text{if } d(p_1,e+1,i,I)\neq\varepsilon \\ d(p_2,e+1,i+2^e,I) & \text{otherwise} \end{cases} \\
d(p_1\,|\,p_2,e,i,I) &= \begin{cases} d(p_1,e+1,i,I) & \text{if } d(p_1,e+1,i,I)\neq\varepsilon \\ d(p_2,e+1,i+2^e,I) & \text{otherwise} \end{cases} \\
d((\nu a)p,e,i,I) &= [e,i] \\
d(C,e,i,I) &= \begin{cases} \varepsilon & \text{if } C\in I \\ d(p,e,i,I\cup\{C\}) & \text{otherwise, with } C\doteq p \end{cases}
\end{aligned}
$$

Table 9.2 Decoration function

$$
\begin{aligned}
dec(\mathbf{0},e,i) &= \emptyset \\
dec(\mu.p,e,i) &= \{(\mu.p,d(\mu.p,e,i,\emptyset))\} \\
dec(p_1+p_2,e,i) &= \{(p_1+p_2,d(p_1+p_2,e,i,\emptyset))\} \\
dec(p_1\,|\,p_2,e,i) &= dec(p_1,e+1,i)\oplus dec(p_2,e+1,i+2^e) \\
dec((\nu a)p,e,i) &= dec(p\{\eta_i/a\},e,i+2^e) \\
dec(C,e,i) &= \{(C,d(C,e,i,\emptyset))\}
\end{aligned}
$$

Table 9.3 Decomposition function

setting of our CCS dialect, without any attempt to optimize the net semantics — are as follows. Formally, the decoration function

$d : \mathscr{P}^{\mathscr{N}} \times \mathbb{N}\times\mathbb{N}\times\mathscr{P}_{fin}(\mathscr{C}ons) \to \{\varepsilon\}\cup\mathbb{N}\times\mathbb{N}$

is defined in Table 9.2, where the third parameter i gives the first available index, and the second parameter e is used to compute 2^e, which specifies the length of the step to the next index. It exploits, as fourth parameter, a set of constant names, in order to avoid looping, as the decoration of a constant C is computed by taking the decoration of its body p, computed w.r.t. the enriched set of constants $I\cup\{C\}$. The only case where the decoration is not ε is for $d((\nu a)p,e,i,I)=[e,i]$. In all the other cases, either the result is ε (for $\mathbf{0}$ and for C, when $C\in I$), or it is computed inductively on the structure of the process. Note that for the choice operator and for parallel composition, the inductive cases $d(p_1,e+1,i,I)$ and $d(p_2,e+1,i+2^e,I)$ are computed w.r.t. a step 2^{e+1}, starting from i for p_1 and $i+2^e$ for p_2. The reason for this will be clarified later.

The new decomposition function $dec : \mathscr{P}^{\mathscr{N}} \times\mathbb{N}\times\mathbb{N}\to\mathscr{M}_{fin}(S_{CCS})$ is defined in Table 9.3. As expected, the decomposition of $\mathbf{0}$ is the empty multiset. The de-

$$\text{(pref)}\ \frac{}{dec(\mu.p,e,i)\xrightarrow{\mu} dec(p,e,i)} \qquad \text{(cons)}\ \frac{dec(p,e,i)\xrightarrow{\mu} m}{dec(C,e,i)\xrightarrow{\mu} m}\ \ C \doteq p$$

$$\text{(sum}_1\text{)}\ \frac{dec(p_1,e+1,i)\xrightarrow{\mu} m}{dec(p_1+p_2,e,i)\xrightarrow{\mu} m} \qquad \text{(sum}_2\text{)}\ \frac{dec(p_2,e+1,i+2^e)\xrightarrow{\mu} m}{dec(p_1+p_2,e,i)\xrightarrow{\mu} m}$$

$$\text{(com)}\ \frac{m_1 \xrightarrow{\gamma} m_1' \ \ m_2 \xrightarrow{\overline{\gamma}} m_2'}{m_1 \oplus m_2 \xrightarrow{\tau} m_1' \oplus m_2'}\ \ ad(m_1 \oplus m_2)$$

Table 9.4 Rules for net transitions

composition of a sequential process s is (s,δ), where the decoration δ is computed by the decoration function d, whose fourth parameter is the empty set; of course, if $bn(s)=\emptyset$, then $\delta=\varepsilon$, otherwise, $\delta=[e',i']$, which represents the correct indexing that is used when the restriction operator will be encountered. The case for the restricted process $(\nu a)p$ shows that the index i is used to select action η_i, to be substituted for the bound name a, and then the decomposition of p continues with the next index $i+2^e$. The case for the parallel process $p_1\,|\,p_2$ shows that the decomposition is done inductively on the processes p_1 and p_2, where the index space is partitioned into two sets: p_1 has all the indexes starting from i with step 2^{e+1}, and p_2 all the indexes from $i+2^e$ with step 2^{e+1}. To clarify this issue, consider $dec((p_1\,|\,p_2)\,|\,p_3,0,0)$; the initial parameters have been chosen to have $\mathbb{N}$ as the initial index space: in fact, the initial available index is 0 and the step is $2^0=1$. Now,

$$dec((p_1\,|\,p_2)\,|\,p_3,0,0) = dec(p_1\,|\,p_2,1,0) \oplus dec(p_3,1,1)$$

and so the initial index for $p_1\,|\,p_2$ is 0 with step $2^1=2$, so that its index space is the set of even numbers; while the index space for p_3 is the set of odd numbers, as the initial index is 1 and the step is 2. Now,

$$dec(p_1\,|\,p_2,1,0) = dec(p_1,2,0) \oplus dec(p_2,2,2)$$

and so the initial index for p_1 is 0 with step $2^2=4$, so that its index space is $J_1=\{4k \mid k\in\mathbb{N}\}$, while the initial index for p_2 is 2 with step 4, so that its index space is $J_2=\{2+4k \mid k\in\mathbb{N}\}$. Note that J_1 and J_2 constitute a partition of the set of even numbers, which have been allotted to $p_1\,|\,p_2$.

Of course, given the more structured nature of the places, also the net transition rules are to be adapted. Table 9.4 describes the rules for our CCS dialect. These rules exploit the parametrized function dec also in the source state, in order to determine the parameters e and i that are to be used to compute the target multiset. Note that the rules (sum$_1$) and (sum$_2$) also perform a partitioning for p_1 and p_2 of the index space of p_1+p_2, similar to what dec performs over parallel processes.

To illustrate how this theory works, let us consider again the process $p=B\,|\,B$, where the constant B is defined as $B \doteq a.(\nu b)(b.c.\mathbf{0}+\overline{b}.\mathbf{0})$. Then,

$dec(B\,|\,B,0,0) = dec(B,1,0) \oplus dec(B,1,1)$

where $dec(B,1,0) = \{(B, d(B,1,0,\emptyset))\}$ and $dec(B,1,1) = \{(B, d(B,1,1,\emptyset))\}$, with $d(B,1,0,\emptyset) = d(a.(\nu b)(b.c.\mathbf{0} + \overline{b}.\mathbf{0}), 1, 0, \{B\}) = d((\nu b)(b.c.\mathbf{0} + \overline{b}.\mathbf{0}), 1, 0, \{B\}) = [1,0]$, and, similarly, $d(B,1,1,\emptyset) = [1,1]$. So, the initial marking is
$m_0 = \{(B,[1,0]),(B,[1,1])\}$
where the two occurrences of the constant B have been differentiated by the decoration. By rule (cons), $dec(B,1,0)$ can do what $dec(a.(\nu b)(b.c.\mathbf{0} + \overline{b}.\mathbf{0}),1,0)$ can do; by axiom (pref), $dec(a.(\nu b)(b.c.\mathbf{0} + \overline{b}.\mathbf{0}),1,0) \xrightarrow{a} dec((\nu b)(b.c.\mathbf{0} + \overline{b}.\mathbf{0}),1,0) = dec((b.c.\mathbf{0} + \overline{b}.\mathbf{0})\{\eta_0/b\},1,2) = \{(\eta_0.c.\mathbf{0} + \overline{\eta}_0.\mathbf{0}, \varepsilon)\}$, because $d(\eta_0.c.\mathbf{0} + \overline{\eta}_0.\mathbf{0}, 1, 2, \emptyset) = \varepsilon$. Hence, the transition $dec(B,1,0) \xrightarrow{a} \{(\eta_0.c.\mathbf{0} + \overline{\eta}_0.\mathbf{0}, \varepsilon)\}$ describes the behavior of the first B. Similarly, $dec(B,1,1) \xrightarrow{a} \{(\eta_1.c.\mathbf{0} + \overline{\eta}_1.\mathbf{0}, \varepsilon)\}$. No other transition is derivable, as η_0 and η_1 are restricted actions, which cannot be performed asynchronously, and, being different, they cannot be synchronized. Therefore, the net for $p = B\,|\,B$ constructed with this technique is coherent with the expected semantics.

This indexing technique, sketched here for the case of this CCS dialect, can be easily extended to the analogous Multi-CCS dialect, and also to the corresponding π-calculus dialect, by using nets with inhibitor arcs [BG09]; in all these cases, the net semantics associates a finite Petri net with *finite-net* processes, i.e., processes where restriction cannot occur within the body of recursively defined constants.[4]

9.4 Asynchronous Communication

The process algebras we have studied in this book are based on *synchronous* communication: the partners of the communication are performing their complementary input/output actions at the same time. Hence, the send operation is *blocking*: the sender, after sending the message, can proceed only after the receiver has received the sent message.

However, in distributed systems a weaker form of communication is usually adopted. *Asynchronous* communication is a form of communication where the send operation is non-blocking, i.e., once performed, the sender can proceed without necessarily waiting for the reception of the sent message. Various process algebraic formalizations of this communication mechanism have been suggested in the literature (see, e.g., [BGZ00, BPV08]). However, there is widespread agreement, in the process algebra community, that a very simple formalization, suggested by [HT91, Bou92], is appropriate for the aim. This formalization simply calls for a syntactic restriction: an output prefix $\overline{a}$ cannot have any continuation. To fix the details, let us consider the *asynchronous* FNC (AFNC, for short), whose syntax is outlined in Table 9.5. It is clear that a term of the form $p = \overline{a}.b.\mathbf{0}$ is not derivable by the syntax. However, if we want to express that in p the output $\overline{a}$ is non-blocking, then we can define the AFNC process $\overline{a}.\mathbf{0}\,|\,b.\mathbf{0}$, which intuitively expresses the ex-

[4] To be precise, in the case of the π-calculus, also some additional conditions are necessary [BG09]; a simple, sufficient, additional condition is that the process term p is closed, i.e., $fn(p) = \emptyset$.

$s ::= \mathbf{0} \mid \overline{a}.\mathbf{0} \mid a.t \mid s+s$	
$q ::= s \mid C$	*sequential terms*
$t ::= q \mid t \vert t$	*restriction-free terms*
$p ::= t \mid (\nu a)p$	*general terms*

Table 9.5 Syntax for asynchronous FNC (AFNC)

pected behavior: the order in which the emitted output $\overline{a}$ is ready for synchronization and the input b is performed is arbitrary.

AFNC is a subcalculus of FNC, so that all the results we have proved in Chapter 6 hold also for AFNC. In particular, the net semantics of any AFNC process is a finite CCS net. Interestingly enough, the FNC process $\mathscr{T}_{FNC}(N(m_0))$ associated with a finite CCS net is actually an AFNC process, and so we could state that AFNC, rather than FNC, is the minimal language that can represent all the finite CCS nets. Similar considerations hold for FNM and NPL.

9.5 Other Languages?

Is the hierarchy of languages we have proposed in this book the only possible one for the chosen hierarchy of distributed systems? In other words, once we have fixed a class of models, can we find a different set of process algebraic operators that can represent that class?

Let us fix one class of models: finite P/T Petri nets. Of course, we do not claim that FNM is the only language that can represent all and only the finite P/T Petri nets. However, we would like to point out some features that any other language capable of representing finite P/T nets must have, so that, in the end, FNM seems the only natural candidate. In fact, the parallel operator $-\parallel-$ of a language able to express Petri nets has to be

- *permissive*: in a process $p \parallel q$, the actions p can perform cannot be prevented by q. This requirement is necessary because P/T Petri nets are permissive as well, meaning that if a transition t is enabled at a marking m, then t is also enabled at a marking $m' \supseteq m$; the parallel operator of FNM is permissive, while this is not the case for other parallel operators, such as the CSP one $p \parallel_A q$ [Hoa85].
- Moreover, the parallel operator $-\parallel-$ is to be ACI (*associative, commutative, with an identity*), because the decomposition of a parallel process into a marking has to reflect that a marking is a (finite) *multiset*; also in this case, the parallel operator of FNM is ACI, while this is not the case for other parallel operators, notably the CSP one.
- In addition, the parallel operator should be able to express multi-party synchronization, because a net transition, which may have a pre-set of any size, can

be generated by means of a synchronization among many participants, actually as many as the tokens in its pre-set. The FNM parallel operator can model multi-party synchronization, by means of the interplay with the strong prefixing operator. Other process algebras offer parallel operators with multi-party synchronization capabilities, but in Multi-CCS multi-party synchronization is "programmable", meaning that we can prescribe the order in which the various participants are to interact, independently of the syntactic position they occupy within the global term and without resorting to a global synchronization function, as in the case of some ACP dialects [BBR10].

The multi-party synchronization discipline has been chosen as simple as possible: a sequence can synchronize with one complementary action at a time, in the exact order they occur in the sequence. About sequentialization operators, we note that prefixing cannot be replaced by ACP-like sequential composition, because a language with recursion and sequential composition can express all the context-free languages, while finite P/T nets cannot [Pet81, GV15]; hence, sequential composition is too powerful to express only finite P/T nets. As we have chosen a CCS-like naming convention, the scoping operator, which can occur syntactically only at the top level, is the CCS restriction operator.

Summing up, any other language, if any, able to represent all and only finite P/T Petri nets should possess these necessary features, which, altogether, seem to be exclusive to FNM, or at least very rare in the panorama of process algebras.

9.6 Future Research

To conclude this book, we would like to discuss an open problem whose solution would make the theory presented here really useful in practice, for distributed systems verification.

A *decidable* behavioral equivalence over *reactive systems* supports the verification technique called *equivalence checking*: a complex, detailed model is correct if it is behaviorally equivalent to some simpler model that is self-evidently correct. A language for describing reactive systems opens the way not only to compositional modeling but also to *compositional reasoning*, when the considered behavioral equivalence is a *congruence*. For instance, when checking the equivalence between two composite systems, say $p_1 \,|\, p_2$ and $q_1 \,|\, q_2$, it might be more convenient to check separately whether p_1 is equivalent to q_1, and whether p_2 is equivalent to q_2, instead of considering the two large global state spaces, because when the equivalence is a congruence for parallel composition, then we are sure that $p_1 \,|\, p_2$ and $q_1 \,|\, q_2$ are equivalent, too. *Compositional equivalence checking* on finite-state LTS process algebras, such as RCS (see Section 6.8), is a viable verification technique because the used equivalence (typically, some form of bisimilarity over finite-state LTSs) is *decidable* and also a *congruence* for the operators of RCS.

If we desire a similar compositional verification technique for the process algebras we have introduced in this book, each representing a different class of Petri nets, and so of *distributed systems*, we have to find a behavioral equivalence relation which is decidable over the chosen class of finite Petri nets and a congruence for the operators of the associated process algebra. For instance, for CFM we have that step bisimilarity is a congruence (Theorem 5.3) and decidable over concurrent FSM nets (as they are finite, bounded P/T nets). So, compositional equivalence checking is a viable verification techniques for CFM.

However, the problem is not always easy, in particular for FNC and its conservative extensions FNM and NPL. Let us consider FNM and let us examine the three well-known equivalences over finite P/T nets we have used throughout the book: interleaving bisimilarity, step bisimilarity and net isomorphism. On the one hand, only net isomorphism is decidable for finite P/T nets, while interleaving bisimilarity and step bisimilarity are undecidable [Jan95, Esp98]. On the other hand, net isomorphism is not a congruence for the choice operator $+$, since even LTS isomorphism is not a congruence for $+$ (see Section 4.2.2): $Net(a.(\mathbf{0}+\mathbf{0}))$ is isomorphic to $Net(a.(\mathbf{0}+\mathbf{0}+\mathbf{0}))$, but $Net(a.(\mathbf{0}+\mathbf{0})+a.(\mathbf{0}+\mathbf{0}))$ is not isomorphic to $Net(a.(\mathbf{0}+\mathbf{0}+\mathbf{0})+a.(\mathbf{0}+\mathbf{0}))$. Moreover, interleaving bisimilarity is not a congruence for FNM parallel composition, while step bisimilarity is a congruence for all the operators of FNM. Summing up, none of these three equivalences satisfies both properties: being decidable and a congruence. Of course, if we restrict our attention to RMCS (see Section 7.8) — hence, only to finite, *bounded* nets — the situation is much better and step bisimilarity can be used to this aim, being decidable and a congruence. However, for FNM and general, unbounded, finite P/T nets, it is a challenging open problem to find an equivalence relation which satisfies both properties.

Glossary

Symbol	Description	Where
Lab	set of labels	Definition 2.1
τ	internal, invisible action	Definition 2.1
$\mathscr{L}$	set of actions (inputs)	Section 4.1
$\overline{\mathscr{L}}$	set of co-actions (outputs)	Section 6.1
$Act = \mathscr{L} \cup \overline{\mathscr{L}} \cup \{\tau\}$	set of all actions	Section 6.1
$\mathscr{L}' \cup \overline{\mathscr{L}'}$	set of all restricted actions	Section 6.1.1
$\mathscr{G} = \mathscr{L} \cup \mathscr{L}'$	set of inputs, also restricted	Section 6.1.1
$\overline{\mathscr{G}} = \overline{\mathscr{L}} \cup \overline{\mathscr{L}'}$	set of outputs, also restricted	Section 6.1.1
$Act_\gamma = \mathscr{G} \cup \overline{\mathscr{G}} \cup \{\tau\}$	set of all actions, also restricted	Section 6.1.1
$\xrightarrow{\mu}$	labeled transition in an LTS	Definition 2.2
$\xrightarrow{\sigma}{}^{*}$	reflexive transitive closure of $\xrightarrow{\mu}$	Definition 2.4
$\longrightarrow^{*}$	reachability	Definition 2.4
$\cong$	LTS isomorphism	Definition 2.8
$=_{tr}$	trace equivalence	Definition 2.9
$=_{ctr}$	completed trace equivalence	Definition 2.11
$\lesssim$	simulation preorder	Definition 2.12
$\simeq$	simulation equivalence	Definition 2.12
$\sim$	bisimulation equivalence	Definition 2.12
$\equiv$	structural congruence	Definition 2.14
$\overset{\sigma}{\Longrightarrow}$	labeled weak transition in an LTS	Definition 2.15
$=_{wtr}$	weak trace equivalence	Definition 2.16
$=_{wctr}$	weak completed trace equivalence	Definition 2.17
$\lessapprox$	weak simulation preorder	Definition 2.18
$\approxeq$	weak simulation equivalence	Definition 2.18
$\approx$	weak bisimulation equivalence	Definition 2.19
M	multiset (of labels)	Definition 2.21
$SLab$	step labels	Definition 2.22
$\sim_{step}$	step bisimulation equivalence	Definition 2.24
$o(M)$	observable content of a step label	Table 2.1
$\overset{M}{\Longrightarrow}$	weak step transition	Table 2.2
$\lessapprox_{step}$	weak step simulation preorder	Definition 2.26
$\approx_{step}$	weak step bisimulation equivalence	Definition 2.26
m	multiset (of places)	Definition 3.1
$\oplus$	multiset union	Definition 3.1
$\ominus$	multiset difference	Definition 3.1

R. Gorrieri, *Process Algebras for Petri Nets*, Monographs in Theoretical
Computer Science. An EATCS Series, DOI 10.1007/978-3-319-55559-1

Symbol	Description	Where
${}^{\bullet}t$	pre-set of a net transition	Definition 3.2
$t^{\bullet}$	post-set of a net transition	Definition 3.2
$l(t)$	labeling of a net transition	Definition 3.2
${}^{\circ}t$	neg-set of a net transition	Definition 3.22
$m[t\rangle$	t enabled at m	Definition 3.4
$Net_d(N(m_0))$	dynamically reachable subnet	Definition 3.7
$Net_s(N(m_0))$	statically reachable subnet	Definition 3.9
$\cong_r$	(rooted) net isomorphism	Definition 3.15
$IMG(N(m_0))$	interleaving marking graph	Definition 3.16
$L[N(m_0)]$	Petri net language	Definition 3.18
G	multiset (of transitions)	Section 3.4.3
$SMG(N(m_0))$	step marking graph	Definition 3.20
$aIMG(N(m_0))$	abstract IMG for NP/T nets	Definition 3.26
$\sim_{int}^{a}$	abstract interleaving bisimilarity	Definition 3.26
$cIMG(N(m_0))$	concrete IMG for NP/T nets	Definition 3.26
$\sim_{int}^{c}$	concrete interleaving bisimilarity	Definition 3.26
$aSMG(N(m_0))$	abstract SMG for NP/T nets	Definition 3.27
$\sim_{step}^{a}$	abstract step bisimilarity	Definition 3.26
$cSMG(N(m_0))$	concrete SMG for NP/T nets	Definition 3.27
$\sim_{step}^{c}$	concrete step bisimilarity	Definition 3.26
$\mathscr{C}ons$	set of process constants	Section 4.1
$Const(p)$	set of constants used by p	Definition 4.1
$\mathscr{P}_{SFM}$	set of SFM processes	Definition 4.2
$sub(p)$	set of subterms of p	Table 4.1
$\mathscr{C}_{SFM}$	LTS for SFM	Definition 4.4
N_{SFM}	net for SFM	Section 4.3
$dec(p)$	decomposition of p	Table 4.3
$Net(p)$	reachable subnet from $dec(p)$	Definition 4.5
$\mathscr{T}_{SFM}(-)$	translation function for SFM	Definition 4.6
$[\![-]\!]_{\emptyset}$	denotational net semantics	Section 4.5
$\mathscr{P}_{CFM}$	set of CFM processes	Section 5.1
$\mathscr{P}_{CFM}^{seq}$	set of sequential CFM processes	Section 5.1
$sub(p)$	set of sequential subterms of p	Section 5.1
$\mathscr{C}_{CFM}$	LTS for CFM	Section 5.1.1
$\mathscr{C}_{CFM}^{step}$	STS for CFM	Section 5.1.2
N_{CFM}	net for CFM	Section 5.1.3
$dec(p)$	decomposition of p	Table 5.3
$Net(p)$	reachable subnet from $dec(p)$	Definition 5.1
$\mathscr{T}_{CFM}(-)$	translation function for CFM	Definition 5.2
$[\![-]\!]_{\emptyset}$	denotational net semantics	Section 5.1.6
$\mathscr{P}_{BPP}$	set of BPP processes	Section 5.2
$\mathscr{P}_{BPP}^{seq}$	set of sequential BPP processes	Section 5.2
N_{BPP}	net for BPP	Section 5.2.2
$Net(p)$	reachable subnet from $dec(p)$	Definition 5.4
$\mathscr{T}_{BPP}(-)$	translation function for BPP	Definition 5.5
$[\![-]\!]_{\emptyset}$	denotational net semantics	Section 5.2.4

$Const(p)$	set of constants used by p	Table 6.1
$\mathscr{P}_{FNC}$	set of FNC processes	Section 6.1
$\mathscr{P}^{seq}_{FNC}$	set of sequential FNC processes	Section 6.1
$bn(p)$	bound names	Definition 6.1
$fn(p)$	free names	Definition 6.1
$fo(p)$	free outputs	Definition 6.1
$ad(-)$	admissible (term)	Section 6.1.1
$\mathscr{P}^{\gamma,par}_{FNC}$	set of extended, restriction-free FNC procs	Section 6.1.1
$-\{a'/a/\}$	syntactic substitution	Table 6.3
$i(-)$	function from $\mathscr{P}_{FNC}$ to $\mathscr{P}^{\gamma,par}_{FNC}$	Definition 6.5
$sub(p)$	set of sequential subterms of p	Table 6.4
$\mathscr{C}_{FNC}$	LTS for FNC	Section 6.2
$\mathscr{C}^{step}_{FNC}$	STS for FNC	Section 6.3
N_{FNC}	net for FNC	Section 6.4
$dec(p)$	decomposition of p	Table 6.8
$Net(p)$	reachable subnet from $dec(p)$	Definition 6.11
$\mathscr{T}_{FNC}(-)$	translation function for FNC	Definition 6.12
$[\![-]\!]_\emptyset$	denotational net semantics	Section 6.7
$\mathscr{P}_{FNM}$	set of FNM processes	Section 7.1.1
$\mathscr{P}^{seq}_{FNM}$	set of sequential FNM processes	Section 7.1.1
$wf(p)$	well-formed FNM process	Table 7.1
$In(p)$	initials of p	Section 7.1.3
$\mathscr{C}_{FNM}$	LTS for FNM	Section 7.2
$\equiv$	structural congruence for FNM	Table 7.5
$Sync$	synchronization relation for FNM	Table 7.4
$\mathscr{C}^{step}_{FNM}$	STS for FNM	Section 7.3
$MSync$	step synchronization relation	Table 7.11
N_{FNM}	net for FNM	Section 7.4
$Net(p)$	reachable subnet from $dec(p)$	Definition 7.3
$\mathscr{T}_{FNM}(-)$	translation function for FNM	Definition 7.4
$[\![-]\!]_\emptyset$	denotational net semantics	Section 7.7
$\mathscr{T}cons$	set of testable process constants	Section 8.1
$\neg\mathscr{T}cons$	set of negated testable constant names	Section 8.1
$\mathscr{P}_{NPL}$	set of NPL processes	Section 8.1
$\mathscr{P}^{seq}_{NPL}$	set of sequential NPL processes	Section 8.1
$\mathscr{C}^{c}_{NPL}$	concrete LTS for NPL	Section 8.2
$\mathscr{C}^{a}_{NPL}$	abstract LTS for NPL	Section 8.2
$\sim^{c}_{int}$	concrete interleaving bisimilarity	Section 8.2.2
$\sim^{a}_{int}$	abstract interleaving bisimilarity	Section 8.2.2
$\mathscr{C}^{c,step}_{NPL}$	concrete STS for NPL	Section 8.3
$\mathscr{C}^{a,step}_{NPL}$	abstract STS for NPL	Section 8.3
$\sim^{c}_{step}$	concrete step bisimilarity	Definition 8.2
$\sim^{a}_{step}$	abstract step bisimilarity	Definition 8.3
N_{NPL}	net for NPL	Section 8.4
$Net(p)$	reachable subnet from $dec(p)$	Definition 8.4
$\mathscr{T}_{NPL}(-)$	translation function for NPL	Definition 8.5
$[\![-]\!]_\emptyset$	denotational net semantics	Section 8.7

References

[Age75] T. Agerwala, "Towards a Theory for the Analysis and Synthesis of Systems Exhibiting Concurrency", Ph.D. dissertation, Johns Hopkins University, 1975.

[AILS07] L. Aceto, A. Ingólfsdóttir, K. Larsen, J. Srba, *Reactive Systems: Modelling, Specification and Verification*, Cambridge University Press, 2007.

[Bae05] J.C.M. Baeten, "A Brief History of Process Algebra", *Theoretical Computer Science* 51(1/2):129-176, 2005.

[BB87] T. Bolognesi, E. Brinksma, "Introduction to the ISO Specification Language LOTOS", *Computer Networks* 14: 25-59, 1987.

[BBGM14] P. Baldan, F. Bonchi, F. Gadducci, G. Monreale, "Encoding Synchronous Interactions Using Labelled Petri Nets", in Procs. Coordination'14, LNCS 8459, 1-16, Springer-Verlag, 2014.

[BBR10] J.C.M. Baeten, T. Basten, M.A. Reniers, *Process Algebra: Equational Theories of Communicating Processes*, Cambridge Tracts in Theoretical Computer Science 50, Cambridge University Press, 2010.

[BCMS01] O. Burkart, D. Caucal, F. Moller, B. Steffen, "Verification on Infinite Structures", Chapter 9 of [BPS01], 545-623, 2001.

[BDK01] E. Best, R. Devillers, M. Koutny, "A Unified Model for Nets and Process Algebras", Chapter 14 of [BPS01], 873-944, Elsevier, 2001.

[BDK02] E. Best, R. Devillers, M. Koutny, "The Box Algebra = Petri Nets + Process Expressions", *Inf. Comput.*, 178(1):44-100, 2002.

[BDKP91] E. Best, R. Devillers, A. Kiehn, L. Pomello, "Concurrent Bisimulations in Petri Nets", *Acta Inf.* 28(3): 231-264, 1991.

[BG89] A. Brogi, R. Gorrieri, "A Distributed, Net Oriented Semantics for Delta Prolog", in *Theory and Practice of Software Development - Colloquium on Trees in Algebra and Programming* (TAPSOFT 89), LNCS 351, 162-177, Springer-Verlag, 1989.

[BG02] M. Bravetti, R. Gorrieri, "Deciding and Axiomatizing Weak ST Bisimulation for a Process Algebra with Recursion and Action Refinement", *ACM Trans. Comput. Log.* 3(4): 465-520, 2002.

[BG09] N. Busi, R. Gorrieri, "Distributed Semantics for the π-calculus Based on Petri Nets with Inhibitor Arcs", *Journal of Logic and Algebraic Programming* 78(3):138-162, 2009.

[BGJ10] S. Böhm, S. Göller, P. Jančar, "Bisimilarity of One-Counter Processes Is PSPACE-Complete", in Procs. CONCUR'10, LNCS 6269, 177-191, Springer-Verlag, 2010.

[BGZ98] N. Busi, R. Gorrieri and G. Zavattaro, "A Process Algebraic View of Linda Coordination Primitives", *Theoretical Computer Science* 192(2):167-199, 1998.

[BGZ00] N. Busi, R. Gorrieri and G. Zavattaro, "Comparing Three Semantics for Linda-like Languages", *Theoretical Computer Science* 240(1):49-90, 2000.

[BGZ09] N. Busi, M. Gabbrielli, G. Zavattaro, "On the Expressive Power of Recursion, Replication and Iteration in Process Calculi", *Mathematical Structures in Computer Science* 19(6):1191-1222, 2009.

R. Gorrieri, *Process Algebras for Petri Nets*, Monographs in Theoretical
Computer Science. An EATCS Series, DOI 10.1007/978-3-319-55559-1

[BK84] J.A. Bergstra, J.W. Klop, "Process Algebra for Synchronous Communication", *Information and Control* 60:109-137, 1984.

[BKL83] L. Babai, W.M. Kantor, E.M. Luks, "Computational Complexity and the Classification of Finite Simple Groups", in Procs. 24th Annual Symposium of Foundations of Computer Science (FOCS), 162-171, IEEE CS-Press, 1983.

[BLV95] T. Bolognesi, J. van de Lagemaat, C. Vissers (eds.), *LOTOSphere: Software Development with LOTOS*, Springer-Verlag, 1995.

[BMM06] R. Bruni, H.C. Melgratti, U. Montanari, "Event Structure Semantics for Nominal Calculi", in Procs. *CONCUR 2006*, LNCS 4137, 295-309, Springer-Verlag, 2006.

[BMS15] R. Bruni, U. Montanari, M. Sammartino, "Revisiting Causality, Coalgebraically", *Acta Inf.* 52(1): 5-33, 2015.

[Bou92] G. Boudol, "Asynchrony and the π-Calculus" (note), Rapport de Recherche 1702, INRIA, Sophia-Antipolis, 1992.

[BPS01] J.A. Bergstra, A. Ponse, S.A. Smolka (eds.), *Handbook of Process Algebra*, Elsevier, 2001.

[BPV08] R. Beauxis, C. Palamidessi, F.D. Valencia, "On the Asynchronous Nature of the Asynchronous π-Calculus", in [DDM08], 473-492, 2008.

[BP99] N. Busi, G. M. Pinna, "Process Semantics for Place/Transition Nets with Inhibitor and Read Arcs", *Fundamenta Informaticae* 40(2-3):165-197, 1999.

[BP00] N. Busi, G. M. Pinna, "Comparing Truly Concurrent Semantics for Contextual Place/Transition Nets", *Fundamenta Informaticae* 44(3):209-244, 2000.

[BRR89] J. W. de Bakker, W. P. de Roever, G. Rozenberg (Eds.), "Partial Ordering Descriptions and Observations of Nondeterministic Concurrent Systems", in *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, School/Workshop, Noordwijkerhout, The Netherlands, LNCS 354, Springer-Verlag, 1989.

[Bus02] N. Busi, "Analysis Issues in Petri Nets with Inhibitor Arcs", *Theoretical Computer Science* 275(1-2):127-177, 2002.

[BW90] J.C.M. Baeten, W.P. Weijland, *Process Algebra*, Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, 1990.

[CEP95] A. Cheng, J. Esparza, J. Palsberg, "Complexity Results for 1-safe Nets", *Theoretical Computer Science* 147:117-136, 1995.

[Chr93] S. Christensen, "Decidability and Decomposition in Process Algebra", Ph.D. Thesis, University of Edinburgh, 1993.

[CHL11] W. Czerwinski, P. Hofman, S. Lasota, "Decidability of Branching Bisimulation on Normed Commutative Context-free Processes", in Procs. CONCUR 2011, LNCS 6901, 528-542, Springer-Verlag, 2011.

[CHM93] S. Christensen, Y. Hirshfeld, F. Moller, "Bisimulation Equivalence is Decidable for Basic Parallel Processes", in Proc. CONCUR'93, LNCS 715, 143-157, Springe-Verlag, 1993.

[DD89] Ph. Darondeau, P. Degano, "Causal Trees", in Proc. *ICALP'89*, LNCS 372, 234-248, Springer-Verlag, 1989.

[DD90] Ph. Darondeau, P. Degano, "Causal Trees: Interleaving + Causality", in [Gue90], 239-255, 1990.

[DDM88] P. Degano, R. De Nicola, U. Montanari, "A Distributed Operational Semantics for CCS Based on C/E Systems", *Acta Informatica* 26(1-2):59-91, 1988.

[DDM89] P. Degano, R. De Nicola, U. Montanari, "Partial Ordering Descriptions and Observations of Nondeterministic Concurrent Systems", in [BRR89], 438-466,1989.

[DDM08] P. Degano, R. De Nicola, J. Meseguer (eds.), *Concurrency, Graphs and Models, Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, LNCS 5065, Springer-Verlag, 2008.

[DE95] J. Desel, J. Esparza, *Free Choice Petri Nets*, Cambridge University Press, 1995.

[DGM88] P. Degano, R. Gorrieri, S. Marchetti, "An Exercise in Concurrency: A CSP Process as a Condition/Event System", *Advances in Petri Nets 1988*, LNCS 340, 85-105, Springer-Verlag, 1988.

[DH99] W. Damm, D. Harel, “LSCs: Breathing Life into Message Sequence Charts”, in Procs. IFIF TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS), Florence, Italy, IFIP Conference Proceedings 139, Kluwer, 293-311, 1999.

[Dic13] L. E. Dickson, “Finiteness of the Odd Perfect and Primitive Abundant Numbers with n Distinct Prime Factors” *Amer. Journal Math.* 35(4): 413-422, 1913.

[Dij71] E.W. Dijkstra, “Hierarchical Ordering of Sequential Processes”, *Acta Informatica* 1(2): 115-138, 1971.

[DM87] P. Degano, U. Montanari, “Concurrent Histories: A Basis for Observing Distributed Systems”, *J. Comput. Syst. Sci.* 34(2/3): 422-461, 1987.

[DR98] J. Desel, W. Reisig, “Place/Transition Petri Nets”, in [RR98a], 122-173, 1998.

[DRR04] J. Desel, W. Reisig, G. Rozenberg, (Eds.), *Lectures on Concurrency and Petri Nets, Advances in Petri Nets*, LNCS 3098, Springer-Verlag, 2004.

[EN94] J. Esparza, M. Nielsen, “Decidability Issues for Petri Nets: A Survey”, *Bulletin of the EATCS* 52:244-262, 1994.

[Esp95] J. Esparza, “Petri Nets, Commutative Context-Free Grammars, and Basic Parallel Processes”, in Procs. 10th Intl. Symposium on Fundamentals of Computation Theory (FCT’95), LNCS 965, 221-232, Springer-Verlag, 1995.

[Esp98] J. Esparza, “Decidability and Complexity of Petri Net Problems: An Introduction” in [RR98a], 374-428, 1998.

[ER64] C. Elgot, A. Robinson, “Random-access Stored-program Machines, an Approach to Programming Languages”, *Journal of the ACM* 11(4):365-399, 1964.

[FA73] M. Flynn, T. Agerwala, “Comments on Capabilities, Limitations and ‘Correctness’ of Petri Nets”, *Comp. Arch. News* 4(2):81-86, 1973.

[FH99] S. Fröschle, T. Hildebrandt, “On Plain and Hereditary History-preserving Bisimulation”, in Procs. MFCS’99, LNCS 1672, 354-365, Springer-Verlag, 1999.

[FJLS10] S.B. Fröschle, P. Jančar, S. Lasota, Z. Sawa, “Non-interleaving Bisimulation Equivalences on Basic Parallel Processes”, *Inf. Comput.* 208(1): 42-62, 2010.

[Flo62] R.W. Floyd, “Algorithm 97: Shortest Path”, *Comm. of the ACM* 5 (6):345, 1962.

[GG89] R. J. van Glabbeek, U. Goltz, “Equivalence Notions for Concurrent Systems and Refinement of Actions”, in Proc. MFCS’89. LNCS 379, 237-248, Springer-Verlag, 1989.

[Gla01] R.J. van Glabbeek, “The Linear Time - Branching Time Spectrum I”, Chapter 1 of [BPS01], 3-99, 2001.

[Gla93] R.J. van Glabbeek, “The Linear Time - Branching Time Spectrum II”, in Procs. CONCUR ’93, LNCS 715, 66-81, Springer-Verlag, 1993.

[GL95] R. Gorrieri, C. Laneve, “Split and ST Bisimulation Semantics”, *Inf. Comput.* 118(2): 272-288, 1995.

[GM90a] R. Gorrieri, U. Montanari, “Towards Hierarchical Specification of Systems: A Proof System for Strong Prefixing”, *Int. Journal of Foundations of Computer Science*, 1(3):277-293, 1990.

[GM90b] R. Gorrieri, U. Montanari, “SCONE: A Simple Calculus of Nets”, in Proc. *CONCUR’90*, LNCS 458, 2-30, Springer-Verlag, 1990.

[GM95] R. Gorrieri, U. Montanari, “On the Implementation of Concurrent Calculi in Net Calculi: Two Case Studies”, *Theoretical Computer Science* 141(1-2):195-252, 1995.

[GMM90] R. Gorrieri, S. Marchetti, U. Montanari, “A^{2}CCS: Atomic Actions for CCS”, *Theoretical Computer Science* 72(2-3):203-223, 1990.

[Gol88] U. Goltz, “On Representing CCS Programs by Finite Petri Nets”, in Proc. *MFCS’88*, LNCS 324, 339-350, Springer-Verlag, 1988.

[Gol90] U. Goltz, “CCS and Petri nets”, in [Gue90], 334-357, 1990.

[Gor16] R. Gorrieri, “Language Representability of Finite Place/Transition Petri Nets”, *Vietnam Journal of Computer Science* 3(1):15-34, 2016.

[Gor17] R. Gorrieri, “CCS(25,12) is Turing-complete”, submitted for publication.

[GR94] U. Goltz, A. Rensink, “Finite Petri Nets as Models for Recursive Causal Behaviour”, *Theoretical Computer Science* 124(1):169-179, 1994.

[Gue90] I. Guessarian (ed.), *Semantics of Systems of Concurrent Processes*, LITP Spring School on Theoretical Computer Science, La Roche Posay, France, LNCS 469, Springer-Verlag, 1990.

[GV87] R.J. van Glabbeek, F. Vaandrager, "Petri Net Models for Algebraic Theories of Concurrency", in Proc. *PARLE'87*, LNCS 259, 224-242, Springer-Verlag, 1987.

[GV11] R. Gorrieri, C. Versari, "An Operational Petri Net Semantics for A^{2}CCS", *Fundamenta Informaticae* 109(2):135-160, 2011.

[GV15] R. Gorrieri, C. Versari, *Introduction to Concurrency Theory: Transition Systems and CCS*, EATCS Texts in Computer Science, Springer-Verlag, 2015.

[Hack74] M. Hack, "The Recursive Equivalence of the Reachability Problem and the Liveness Problem for Petri Nets and Vector Addition Systems", in Procs. 15th Ann. Symp. on Switching and Automata Theory, IEEE, 1974.

[Hack75] M. Hack, "Decision Problems for Petri Nets and Vector Addition Systems", Technical Memo 59, Project MAC, MIT, 1975. (Early version appeared as Computation Structure Group Memo 95, Project MAC, MIT, 1974.)

[Hack76a] M. Hack, "Petri Net Languages", Technical Report 159, MIT, 1976.

[Hack76b] M. Hack, "Decidability questions for Petri nets", Ph.D. thesis, MIT, 1976.

[Har87] D. Harel, "Statecharts: A Visual Formalism for Complex Systems", *Sci. Comput. Program.* 8(3): 231-274, 1987.

[Hir93] Y. Hirshfeld, "Petri Nets and the Equivalence Problem", in Procs. 7th Workshop on *Computer Science Logic* (CSL'93), LNCS 832,165-174, Springer-Verlag, 1993.

[HMU01] J.E. Hopcroft, R. Motwani, J.D. Ullman, *Introduction to Automata Theory, Languages and Computation*, 2nd ed., Addison-Wesley, 2001.

[Hoa78] C.A.R. Hoare, "Communicating Sequential Processes", *Communications of the ACM* 21(8):666-677, 1978.

[Hoa85] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall International Series in Computer Science, 1985.

[HPSS87] D. Harel, A. Pnueli, J.P. Schmidt, R. Sherman, "On the Formal Semantics of Statecharts" (Extended Abstract), in Procs. of the Symposium on Logic in Computer Science (LICS '87), Ithaca, New York, IEEE Computer Society Press, 54-64, 1987.

[HRS76] H.B. Hunt, D.J. Rosenkrantz, T.G. Szymanski, "On the Equivalence, Containment, and Covering Problems for the Regular and Context-free Languages", *Journal of Computer and System Sciences* 12:222-268, 1976.

[HT91] K. Honda and M. Tokoro, "An object calculus for asynchronous communication", In Procs. of ECOOP'91, LNCS 512, 133-147. Springer-Verlag, 1991.

[Huy83] D.T. Huynh, "Commutative Grammars: The Complexity of Uniform Word Problems", *Information and Control* 57:21-39, 1983.

[Jan95] P. Jančar, "Undecidability of Bisimilarity for Petri Nets and Some Related Problems", *Theoretical Computer Science* 148(2):281-301, 1995.

[Jan03] P. Jančar, "Strong Bisimilarity on Basic Parallel Processes Is PSPACE-complete", in Procs of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03), 218-227, IEEE Computer Society Press, 2003.

[JE96] P. Jančar, J. Esparza, "Deciding Finiteness of Petri Nets up to Bisimulation", In Proceedings of 23rd International Colloquium on Automata, Languages, and Programming (ICALP'96), LNCS 1099, 478-489, Springer-Verlag, 1996.

[JK95] R. Janicki, M. Koutny, "Semantics of Inhibitor Nets", *Information and Computation* 123:1-16, 1995.

[JKM01] P. Jančar, A. Kučera, and R. Mayr, "Deciding Bisimulation-like Equivalences with Finite-state Processes", *Theoretical Computer Science* 258(1-2):409-433, 2001.

[JLL77] N.D. Jones, L.H. Landweber, Y.E. Lien, "Complexity of Some Problems in Petri Nets", *Theoretical Computer Science* 4:277-299, 1977.

[JM95] P. Jančar, F. Moller, "Checking Regular Properties of Petri Nets", In Proceedings of the 6th International Conference on Concurrency Theory (CONCUR'95), LNCS 962, 348-362, Springer-Verlag, 1995.

[JM96] L. Jategaonkar, A. Meyer, "Deciding True Concurrency Equivalences on Finite Safe Nets", *Theor. Comput. Sci.* 154(1): 107-143, 1996.

[JNW96] A. Joyal, M. Nielsen, G. Winskel, "Bisimulation from Open Maps", *Information and Computation* 127:164-185, 1996.

[Kah74] Gilles Kahn, "The Semantics of Simple Language for Parallel Programming", in Procs. of IFIP Congress 74, Stockholm, Sweden, 1974, North-Holland, 471-475, 1974.

[Kel76] R. Keller, "Formal Verification of Parallel Programs", *Comm. of the ACM* 19(7):561-572, 1976.

[KM69] R.M. Karp, R.E. Miller, "Parallel Program Schemata", *Journal of Computer and System Sciences* 3(2):147-195, 1969.

[KM02] A. Kučera, R. Mayr, "Weak Bisimilarity Between Finite-state Systems and BPA or Normed BPP Is Decidable in Polynomial Time", *Theor. Comput. Sci.* 270(1-2): 677-700, 2002.

[Kön36] D. König, *Theorie der endlichen und unendlichen Graphen: Kombinatorische Topologie der Streckenkomplexe.* Akad-Verlag, 1936.

[Kos82] S.R. Kosaraju, "Decidability of Reachability in Vector Addition Systems", in Procs. 6th ACM STOC, 267-281, ACM Press, 1982.

[Kot05] M. Kot. "Regularity of BPP Is PSPACE-complete", in Procs. 3rd Ph.D. Workshop of Faculty of Electrical Engineering and Computer Science (WOFEX'05), pages 393-398, Ostrava, 2005.

[KS05] M. Kot, Z. Sawa, "Bisimulation Equivalence of a BPP and a Finite-state System Can Be Decided in Polynomial Time", in Procs. 6th International Workshop on Verification of Infinite-State Systems (INFINITY'04), volume 138 of ENTCS, pages 49-60, Elsevier Science Publishers, 2005.

[Lam92] J.L. Lambert, "A Structure to Decide Reachability in Petri Nets", *Theoretical Computer Science* 99:79-104, 1992.

[Ler10] J. Leroux, "The General Vector Addition System Reachability Problem by Presburger Inductive Invariants", *Logical Methods in Computer Science* 6(3:22):1-25, 2010.

[Ler11] J. Leroux, "Vector Addition System Reachability Problem: A Short Self-contained Proof", in Proc. 38th Symposium on Principles of Programming Languages (POPL'11), 307-316, ACM Press, 2011.

[Lip76] R. Lipton, "The Reachability Problem Requires Exponential Space", Tech. Rep. 63, Yale University, 1976.

[LR81] D.J. Lehmann, M.O. Rabin, "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem", In Procs. POPL'81, 133-138, ACM Press, 1981. A revised version appeared as [RL94].

[May81] E. W. Mayr, "An Algorithm for the General Petri Net Reachability Problem", in Proc. 13th Annual ACM Symp. on Theory of Comp., (STOC'81), 238-246, ACM Press, 1981.

[May84] E.W. Mayr, "An Algorithm for the General Petri Net Reachability Problem", *SIAM J. Comput.* 13:441-460, 1984.

[Mey09] R. Meyer. "A Theory of Structural Stationarity in the π-Calculus", *Acta Informatica* 46(2):87-137, 2009.

[Mil80] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science 92, Springer-Verlag, 1980.

[Mil85] G.J. Milne, "CIRCAL and the Representation of Communication, Concurrency, and Time", *ACM Trans. Program. Lang. Syst.* 7(2): 270-298, 1985.

[Mil89] R. Milner, *Communication and Concurrency*, Prentice-Hall, 1989.

[Mil99] R. Milner. *Communicating and Mobile Systems: The π-calculus*, Cambridge University Press, 1999.

[Min67] M.L. Minsky, *Computation: Finite and Infinite Machines*, Prentice-Hall, Upper Saddle River, NJ, USA, 1967.

[MM90] J. Meseguer, U. Montanari, "Petri Nets Are Monoids", *Information and Computation* 88(2):105-155, 1990.

[Mor11] M. Morara, *Una Semantica Distribuita per il Multi-CCS Utilizzando Reti di Petri*, Master's thesis (in Italian), University of Bologna, 2011.

[MPW92] R. Milner, J. Parrow, D. Walker, “A Calculus of Mobile Processes”, *Information and Computation* 100(1), 1-77, 1992.

[MR95] U. Montanari, F. Rossi, “Contextual Nets”, *Acta Informatica* 32:545-596, 1995.

[MY94] U. Montanari, D. Yankelevich, “Combining CCS and Petri Nets Via Structural Axioms”, *Fundam. Inform.* 20(1/2/3):193-229, 1994.

[NPW81] M. Nielsen, G.D. Plotkin, G. Winskel, “Petri Nets, Event Structures and Domains, Part I”, *Theor. Comput. Sci.* 13: 85-108, 1981.

[NT84] M. Nielsen, P.S. Thiagarajan, “Degrees of Non-determinism and Concurrency: A Petri Net View”, in Procs. of the Fourth Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS’84), LNCS 181, 89-117, Springer-Verlag, 1984.

[Old91] E.R. Olderog, *Nets, Terms and Formulas*, Cambridge Tracts in Theoretical Computer Science 23, Cambridge University Press, 1991.

[Par01] J. Parrow, “An Introduction to the π-calculus”, Chapter 8 of [BPS01], 479-543, 2001.

[Park81] D.M.R. Park, “Concurrency and Automata on Infinite Sequences”, In Proc. 5th GI-Conference on Theoretical Computer Science, LNCS 104, 167-183, Springer-Verlag, 1981.

[Pet81] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981.

[Petri62] C.A. Petri, *Kommunikation mit Automaten*, Ph.D. Dissertation, University of Bonn, 1962.

[Plo04a] G.D. Plotkin, “The Origins of Structural Operational Semantics”, *J. Logic and Algebraic Programming* 60-61: 3-15, 2004.

[Plo04b] G.D. Plotkin, “A Structural Approach to Operational Semantics”, *J. Logic and Algebraic Programming* 60-61: 17-139, 2004. Revised version of the original Technical Report DAIMI FN-19, Aarhus University, 1981.

[PRS92] L. Pomello, G. Rozenberg, C. Simone, “A Survey of Equivalence Notions for Net Based Systems”, in *Advances in Petri Nets: The DEMON Project*, LNCS 609, 410-472, Springer-Verlag, 1992.

[PT87] R. Paige, R.E. Tarjan, “Three Partition Refinement Algorithms”, *SIAM Journal of Computing* 16(6):973-989, 1987.

[Rac78] C. Rackoff, “The Covering and Boundedness Problem for Vector Addition Systems”, *Theoretical Computer Science* 6:223-231, 1978.

[RE98] G. Rozenberg, J. Engelfriet, “Elementary Net Systems”, in [RR98a], 12-121, 1998.

[Rei85] W. Reisig, *Petri Nets: An Introduction*, EATCS Monographs in Theoretical Computer Science, Springer-Verlag, 1985.

[Rei98] W. Reisig, *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets*, Springer-Verlag, 1998.

[Rei13] W. Reisig, *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*, Springer-Verlag, 2013.

[Reu88] C. Reutenauer, *The Mathematics of Petri Nets*, Masson, 1988.

[RGG] E. Rudolph, P. Graubmann and J. Grabowski, “Tutorial on Message Sequence Charts”, available at http://www.sdl-forum.org/MSC/msctutorial.pdf

[RL94] M. O. Rabin, D. Lehmann, “The Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem”, in: A. W. Roscoe (Ed.), *A Classical Mind: Essays in Honour of C.A.R. Hoare*, Prentice Hall, 1994, Chapter 20, 333-352. An extended abstract appeared as [LR81].

[Ros98] A.W. Roscoe, *The Theory and Practice of Concurrency*, Prentice-Hall, 1998.

[RR98a] W. Reisig, G. Rozenberg (eds.), *Lectures on Petri Nets I: Basic Models*, Lecture Notes in Computer Science 1491, Springer-Verlag, 1998.

[RR98b] W. Reisig, G. Rozenberg (eds.), *Lectures on Petri Nets II: Applications*, Lecture Notes in Computer Science 1492, Springer-Verlag, 1998.

[RT88] A. Rabinovitch, B. Trakhtenbrot, “Behaviour Structures and Nets”, *Funadmenta Informaticae* 11:357-404, 1988.

[San12] D. Sangiorgi, *An Introduction to Bisimulation and Coinduction*, Cambridge University Press, 2012.

[SM73] L.J. Stockmeyer, A.R. Meyer, “Word Problems Requiring Exponential Time”, In Procs. 5th Annual ACM Symposium on Theory of Computing (STOC’73), 1-9, ACM Press, 1973.

[ST77] G.S. Sacerdote, R.L. Tenney, "The Decidability of the Reachability Problem for Vector Addition Systems", in Proc. 9th Annual Symposium of Theory of Computing (STOC'77), ACM Press, 61-76, 1977.

[SW01] D. Sangiorgi, D. Walker, *The π-calculus: A Theory of Mobile Processes*, Cambridge University Press, 2001.

[TA15] W.J. Thong, M. A., Ameedeen, "A Survey of Petri Net Tools", in *Advanced Computer and Communication Engineering Technology*, Lecture Notes in Electrical Engineering 315, 537-551, Springer-Verlag, 2015.

[Tau89] D. Taubner, *Finite Representations of CCS and TCSP Programs by Automata and Petri Nets*, LNCS 369, Springer-Verlag, 1989.

[Tool] Complete Overview of Petri Nets Tools Database, https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/complete_db.html

[Tur36] A.M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem". Proceedings of the London Mathematical Society, Series 2, 42, 230-265, 1936.

[VBG09] C. Versari, N. Busi, R. Gorrieri. "An Expressiveness Study of Priority in Process Calculi", *Mathematical Structures in Computer Science* 19(6):1161-1189, 2009.

[Vog91] W. Vogler, "Deciding History-preserving Bisimulation", in Proc. ICALP'91, LNCS 510, 495-505, Springer-Verlag, 1991.

[Vog97] W. Vogler, "Partial Order Semantics and Read Arcs", in Proc. MFCS'97, LNCS 1295, 508-517, Springer-Verlag, 1997.

[Win87] G. Winskel, "Event Structures", *Advances in Petri Nets, Part II*, Proceedings of an Advanced Course, Bad Honnef, 1986, LNCS 255, 325-392, Springer-Verlag, 1987.

[Win88] G. Winskel, "An Introduction to Event Structures", in [BRR89], 364-397, 1988.

Index

R. Gorrieri, *Process Algebras for Petri Nets*, Monographs in Theoretical Computer Science. An EATCS Series, DOI 10.1007/978-3-319-55559-1