Partial-Order Process Algebra*
(and its Relation to Petri Nets)

J.C.M. Baeten[1] and T. Basten[2]**

[1] Dept. of Computing Science, Eindhoven University of Technology, The Netherlands
josb@win.tue.nl
[2] Dept. of Electrical Engineering, Eindhoven University of Technology, The Netherlands
tbasten@ics.ele.tue.nl

Abstract

To date, many different formalisms exist for describing and analyzing the behavior of concurrent systems. Petri nets and process algebras are two well-known classes of such formalisms. Petri-net theory is well suited for reasoning about concurrent systems in a partial-order framework; it handles causal relationships between actions of concurrent systems in an explicit way. Process algebras, on the other hand, often provide a total-order framework, which means that information about causalities is not always accurate. This chapter illustrates how to develop a partial-order process algebra in the style of ACP. It is shown how to extend such an algebraic theory with a causality mechanism inspired by Petri-net theory. In addition, the chapter clarifies the concepts of interleaving and non-interleaving process algebra; total-order semantics for concurrent systems are often incorrectly referred to as interleaving semantics.

Key words: process algebra – Petri nets – concurrency – partial-order theory – (non-)interleaving process algebra – causality – communication

Contents

*This report will appear as a chapter in J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, Elsevier Science, Amsterdam, The Netherlands.

**Part of this work was done while the author was employed at the Department of Computing Science, Eindhoven University of Technology, The Netherlands.

1 Introduction

The behavior of parallel and distributed systems, often called *concurrent systems*, is a popular topic in the literature on (theoretical) computing science. Numerous formal languages for describing and analyzing the behavior of concurrent systems have been developed. The meaning of expressions in such a formal language is often captured in a so-called formal *semantics*. The objective of a formal semantics is to create a precise and unambiguous framework for reasoning about concurrent systems. Along with the development of the large number of formal languages for describing concurrent systems, an almost equally large number of different semantics has been proposed. The existence of so many different semantics for concurrent systems has created a whole new area of research named *comparative concurrency semantics* [42]. The primary purpose of comparative concurrency semantics is the classification of semantics for concurrent systems in a meaningful way.

On a high level of abstraction, the behavior of a concurrent system is often represented by the actions that the system can perform and the ordering of these actions. Such an abstract view of the behavior of a concurrent system is called a *process*. The semantics of a formal language for describing the behavior of concurrent systems defines a process for each expression in the formal language. Two expressions in a formal language describe the same system if and only if they correspond to equivalent processes in the semantics, where the equivalence of processes is determined by a so-called *semantic equivalence*.

The quest of developing formalisms and semantics that are well suited for describing and analyzing the behavior of concurrent systems is characterized by a number of ongoing discussions on classifications of semantic equivalences. One discussion is centered around linear-time semantics versus branching-time semantics. In a linear-time semantics, two processes that agree on the ordering of actions are considered equivalent. However, such processes may differ in their branching structure, where the branching structure of a process is determined by the moments that choices between alternative branches of behavior are made. A branching-time semantics distinguishes processes with the same ordering of actions but different branching structures.

Another discussion focuses on total-order semantics versus partial-order semantics. In a total-order semantics, actions of a process are always totally ordered, whereas in a partial-order semantics, actions may occur simultaneously or causally independent of each other. Partial-order semantics are often referred to as *true concurrency* semantics, because they are well suited to express concurrency of actions. A typical characteristic of a total-order semantics is that concurrency of actions is equivalent to non-determinism: A process that performs two actions in parallel is equivalent to a process that chooses non-deterministically between the two possible total orderings of the two actions. Thus, a total-order semantics abstracts from the causal dependencies between actions.

Total-order semantics are often confused with *interleaving* semantics. As pointed out in [2], the term "interleaving" originates from one specific class of formal languages for describing concurrent systems, namely process algebras. Process-algebraic theories have in common that processes are represented by terms constructed from action constants and operators such as choice (alternative composition), sequential composition, and parallel composition (merge operator, interleaving operator). A set of axioms or equational laws specifies which processes must be considered equal. Most process-algebraic theories contain some form of *expansion theorem*. An expansion theorem states that parallel composition can be expressed equivalently in terms of choice and sequential composition. A process-algebraic theory with an expansion theorem is called an *interleaving theory*; the semantics of such an algebraic theory is called an *interleaving semantics* or an *interleaving process algebra*. An algebraic theory without an expansion theorem is said to be *non-interleaving*. The semantics of such a theory is a *non-interleaving semantics* or a *non-interleaving process algebra*. As mentioned, most process-algebraic theories are interleaving theories. Very often, such an interleaving theory has a total-order semantics, which causes the confusion between the terms "total-order" and "interleaving." However, in [2], it is shown that it is possible to develop both process-algebraic theories with an interleaving, partial-order semantics and algebraic theories with a non-interleaving, total-order semantics. These examples show that the characterizations interleaving versus non-interleaving and total-order versus partial-order for process algebras are orthogonal. The above discussion also makes clear that, for semantics of non-algebraic languages for describing concurrent systems, the characterization interleaving versus non-interleaving is not meaningful.

This chapter can be situated in the field of comparative concurrency semantics. Its main topic is to illustrate a way to develop a process-algebraic theory with a partial-order semantics. The starting point is an algebraic theory in the style of the Algebra of Communicating Processes (ACP) [13]. The basic semantic equivalence that is used throughout the chapter is bisimilarity [50]. Bisimilarity is often used to provide process-algebraic theories with a semantics that, in the terminology of this chapter, can be characterized as a branching-time, interleaving, total-order semantics. An interesting aspect of bisimilarity is that it can be turned into a semantic equivalence called *step* bisimilarity [48] that provides the basis for a partial-order view on the behavior of concurrent systems. By means of step bisimilarity, it is possible to obtain a process-algebraic theory with a branching-time, interleaving, *partial*-order semantics in a relatively straightforward way. Furthermore, we show how to obtain a *non-interleaving* variant of such a process algebra. This chapter does not discuss variations of process-algebraic theories in the linear-time/branching-time spectrum. The interested reader is referred to [26, 27, 28].

Another important theme of this chapter is the study of concepts known from Petri-net theory [52] in the process-algebraic framework developed in this chapter. The Petri-net formalism is a well-known theory for describing and analyzing concurrent systems. Petri nets have been used both as a language for describing concurrent systems and as a semantic framework for providing other languages for describing concurrent systems with a formal semantics. The Petri-net formalism is particularly well suited for creating a partial-order framework for reasoning about the behavior of concurrent systems. The non-interleaving, partial-order process algebra of Section 10 of this chapter is an algebra that incorporates a number of the most important concepts of the Petri-net formalism. In particular, it adopts the standard Petri-net mechanism for handling causalities.

The chapter is written in the style of a tutorial. This means that it is self-contained, focuses on concepts, and contains many (small) examples and detailed explanations. The goal is not to develop a complete framework with a formal description language and semantics that is applicable to the development of large, complex concurrent systems. In addition, we do not claim that the approach of this chapter is the only way to obtain a process-algebraic theory with a partial-order semantics. It is our aim to provide a conceptual understanding of several important concepts that play a role in describing and analyzing the behavior of concurrent systems. A better understanding of these concepts can be useful in the development of formalisms that are sufficiently powerful to support the development of large and complex systems. The chapter combines and extends some of the ideas and results that appeared earlier in [2], [4], and [8, Chapter 3].

The remainder of this chapter is organized as follows. Section 2 introduces some notation for bags, which are omnipresent in the remainder of this chapter, and a convenient notation for quantifiers. In Section 3, the basic semantic framework used throughout this chapter is defined. The framework of labeled transition systems is used to formalize the notion of a process and bisimilarity of processes. It is shown how labeled transition systems can be used to obtain both a total-order view of concurrent systems and a partial-order view, where the latter is based on the notion of step bisimilarity. Section 4 introduces a class of Petri nets called labeled P/T nets. The framework of labeled transition systems is used to define both a total-order semantics and a step semantics for labeled P/T nets. Section 5 provides an introduction to standard ACP-style process algebra. As in Section 4, labeled transition systems form the basis for both a total-order and a partial-order framework. In Section 6, the algebraic framework of Section 5 is extended with a class of algebraic operators, called causal state operators. This class of operators is inspired by the Petri-net approach to defining the behavior of concurrent systems. It is shown that the resulting algebraic framework is sufficiently powerful to capture the semantics of labeled P/T nets in algebraic terms. Section 7 contains a brief intermezzo on algebraic renaming and communication functions, which is useful in the remaining sections. Section 8 describes two approaches to modularizing P/T nets. A modular P/T net models a system component that may interact with its environment via a well defined interface. It is explained how modular P/T nets in combination with the algebraic framework of Section 6 can be used to develop a compositional formalism for modeling and analyzing concurrent systems. The resulting formalism is a step towards a framework supporting the development of complex concurrent systems. In Section 9, the algebraic framework of Section 6 is extended with a class of so-called cause-addition operators. Cause-addition operators allow for the explicit specification of causalities in algebraic expressions. This class of operators is inspired by the way causalities are handled in Petri-net theory. The relation between cause addition and sequential composition, which is the most important operator for specifying causal orderings in process algebra, is studied. Section 10 studies a process algebra that incorporates several of the important characteristics of Petri-net theory. It is a non-interleaving, partial-order process algebra that includes the classes of causal state operators and cause-addition operators. In the context of this algebra, the relation between the causality mechanisms of standard ACP-style process algebra and Petri-net theory is investigated. Finally, Section 11 summarizes the most important conclusions of this chapter. It also provides some pointers to related work and it identifies some interesting topics for future study.

2 Notation

Bags In this chapter, bags are defined as finite multi-sets of elements from some alphabet A. A bag over alphabet A can be considered as a function from A to the natural numbers $\mathbb{N}$ such that only a finite number of elements from A is assigned a non-zero function value. For some bag X over alphabet A and $a \in A$, $X(a)$ denotes the number of occurrences of a in X, often called the cardinality of a in X. The set of all bags over A is denoted $\mathcal{B}(A)$. Note that any finite set of elements from A also denotes a unique bag over A, namely the function yielding 1 for every element in the set and 0 otherwise. The empty bag, which is the function yielding 0 for any element in A, is denoted $\mathbf{0}$. For the explicit enumeration of a bag, a notation similar to the notation for sets is used, but using square brackets instead of curly brackets and using superscripts to denote the cardinality of the elements. For example, $[a^2 \mid P(a)]$ contains two elements a for every a such that $P(a)$ holds, where P is some predicate on symbols of the alphabet under consideration. To denote individual elements of a bag, the same symbol "$\in$" is used as for sets: for any bag X over alphabet A and element $a \in A$, $a \in X$ if and only if $X(a) > 0$. The sum of two bags X and Y, denoted $X \uplus Y$, is defined as $[a^n \mid a \in A \wedge n = X(a) + Y(a)]$. The difference of X and Y, denoted $X - Y$, is defined as $[a^n \mid a \in A \wedge n = (X(a) - Y(a)) \max 0]$. The binding of sum and difference is left-associative. The restriction of X to some domain $D \subseteq A$, denoted $X \restriction D$, is defined as $[a^{X(a)} \mid a \in D]$. Restriction binds stronger than sum and difference. The notion of subbags is defined as expected: Bag X is a subbag of Y, denoted $X \leq Y$, if and only if for all $a \in A$, $X(a) \leq Y(a)$.

Quantifiers For quantifiers, we use the convenient notation of [23] in which the bound variables are represented explicitly. A quantifier is a commutative and associative operator with a unit element. Examples of well-known quantifiers are the logical quantifiers $\forall$ and $\exists$. Given a quantifier $\mathcal{Q}$ and a non-empty list of variables $\bar{x}$, a quantification over $\bar{x}$ is written in the following format: $(\mathcal{Q}\,\bar{x} : D(\bar{x}) : E(\bar{x}))$, where D is a predicate that specifies the *domain* of values over which the variables in $\bar{x}$ range and where $E(\bar{x})$ is the quantified *expression*. Note that also the explicit enumeration of sets and bags could be written in quantifier notation. However, for sets we adhere to the standard notation with curly brackets and for bags we use the notational conventions introduced in the previous paragraph.

Another notational convention concerning quantifiers is the following. Any *binary* commutative and associative operator $\oplus$ with a unit element can be generalized to a quantifier, also denoted $\oplus$, in a straightforward way. Consider, for example, the binary $+$ on natural numbers. The sum of all natural numbers less than 10 can be written as follows: $(+\,x : x \in \mathbb{N} \wedge x < 10 : x)$.

3 Process Theory

A natural and straightforward way to describe the behavior of a concurrent system is a *labeled transition system*. A labeled transition system consists of a set of *states* plus a *transition relation* on states. Each transition is labeled with an *action*. An action may be any kind of activity performed by a concurrent system, such as a computation local to some component of the system, a procedure call, the sending of a message, or the receipt of a message. However, for modeling purposes, the details of actions are often not important. The set of states in a labeled transition system is an abstraction of all possible states of a concurrent system. The transition relation describes the change in the state of a concurrent system when some action, represented by the label of the transition, is performed.

The framework of labeled transition systems introduced in this section is used throughout the remainder of this chapter to provide the Petri-net formalism of the next section and the various process-algebraic theories of the other sections with a formal semantics. Labeled transition systems can be used to define both a total-order and a partial-order semantics for concurrent systems. The difference is made by choosing an appropriate definition for the actions of a labeled transition system. In a total-order view, actions are assumed to be atomic entities without internal structure. A system can perform only a single atomic action at a time. Thus, the actions are totally ordered. In a partial-order view, a concurrent system can perform causally independent atomic actions simultaneously. Atomic actions are no longer totally ordered. It is even possible that a system performs the same atomic action several times in parallel, because also different occurrences of the same action may be causally independent. Thus, in a partial-order view on the behavior of concurrent systems, actions are *bags* of atomic actions. In this case, actions are often called *multi-actions* or *steps*. A partial-order semantics of concurrent systems that is based on the notion of steps is often called a *step semantics*.

The basic notion in the framework of labeled transition systems used in this chapter is the so-called *process space*. A process space describes a *set of processes*. A process space is a labeled transition system as described above extended with a termination predicate on states. Each state in a process space can be interpreted as the initial state of a process. A *process* is a labeled transition system extended with a termination predicate and a distinguished initial state. The termination predicate of a process defines in what states the process *can terminate successfully*. If a process is in a state where it cannot perform any actions or terminate successfully, then it is said to be in a *deadlock*. The possibility to distinguish between successful termination and deadlock is often useful.

A predicate on the elements of some set is represented as a subset of the set: It holds for elements in the subset and it does not hold for elements outside the subset.

Definition 3.1. (Process space) A *process space* is a quadruple $(\mathcal{P}, \mathcal{A}, \longrightarrow, \downarrow)$, where $\mathcal{P}$ is a set of process states, $\mathcal{A}$ is a set of actions, $\_ \xrightarrow{\_} \_ \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ is a ternary transition relation, and $\downarrow \_ \subseteq \mathcal{P}$ is a termination predicate.

Let $(\mathcal{P}, \mathcal{A}, \longrightarrow, \downarrow)$ be some process space. Each state p in $\mathcal{P}$ uniquely determines a process that consists of all states reachable from p.

Definition 3.2. (Reachability) The *reachability relation* $\stackrel{*}{\Longrightarrow} \subseteq \mathcal{P} \times \mathcal{P}$ is defined as the smallest relation satisfying, for any $p, p', p'' \in \mathcal{P}$ and $\alpha \in \mathcal{A}$,

$p \stackrel{*}{\Longrightarrow} p$ and

$(p \stackrel{*}{\Longrightarrow} p' \wedge p' \stackrel{\alpha}{\longrightarrow} p'') \Rightarrow p \stackrel{*}{\Longrightarrow} p''$.

Process state p' is said to be *reachable* from state p if and only if $p \stackrel{*}{\Longrightarrow} p'$. The set of all states reachable from p is denoted $p*$.

Definition 3.3. (Process) Let p be a state in $\mathcal{P}$. The *process* defined by p is the tuple $(p, p*, \mathcal{A}, \longrightarrow \cap (p* \times \mathcal{A} \times p*), \downarrow \cap p*)$. Process state p is called the *initial state* of the process.

In the remainder, processes are identified with their initial states. Sometimes, it is intuitive to think about elements of $\mathcal{P}$ as states, whereas sometimes it is more natural to see them as processes.

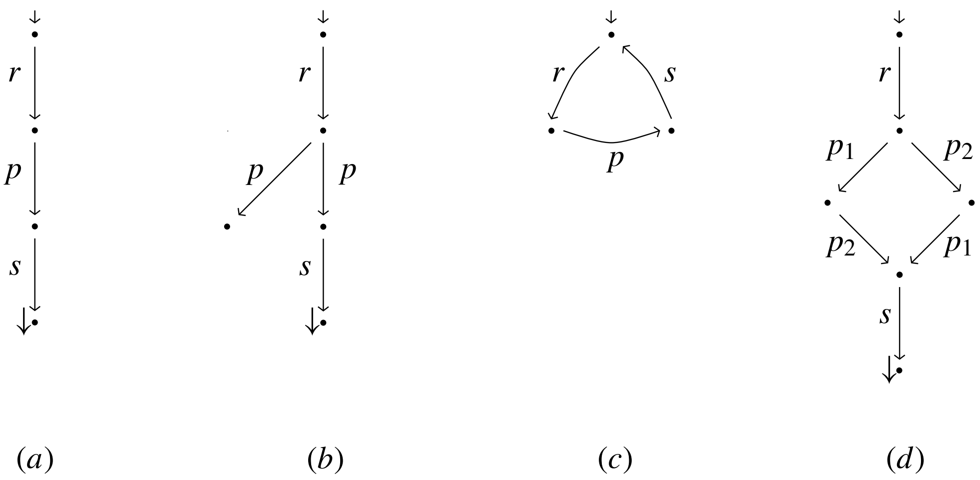


Figure 3.4: Some simple examples of processes.

Example 3.5. Let $(\mathcal{P}, \mathcal{A}, \longrightarrow, \downarrow)$ be some process space, where the set of actions equals a set of atomic actions A that includes the atomic actions p, p_1, p_2, r, and s. Figure 3.4 shows some examples of processes.

Process states are depicted as dots. The initial state of a process is marked with a small incoming arrow. The process in 3.4(a) is a simple sequential process. It represents, for example, a system that receives a message from its environment, processes this message, sends the processed message to its environment, and terminates successfully. The process in Figure 3.4(b) is almost the same, except that the processing action may result in a deadlock. The third process is a variant that iterates the behavior of the system in (a) without a possibility of termination. Finally, the process in Figure 3.4(d) represents a system that performs two processing actions in any order on the incoming message before it returns the processed message to its environment. In a total-order view on concurrent systems, one could also say that the system performs the two processing actions in parallel. There is no way to distinguish concurrency from non-determinism. A total-order semantics abstracts from the causal dependencies between the occurrences of actions.

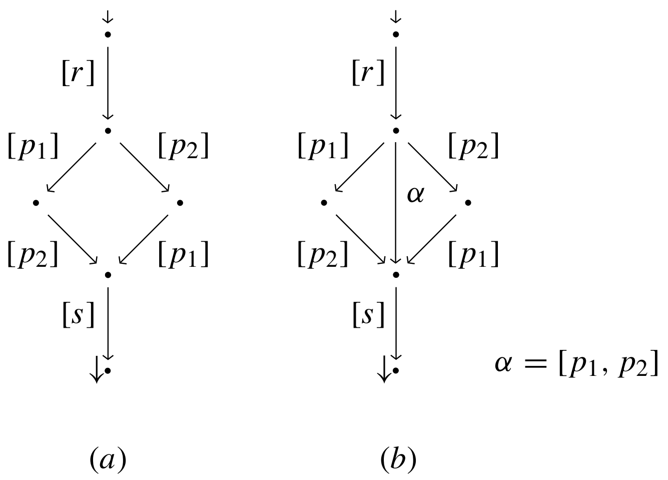


Figure 3.6: Concurrency versus non-determinism in a step semantics.

Example 3.7. Consider again a process space $(\mathcal{P}, \mathcal{A}, \longrightarrow, \downarrow)$. Let A be the same set of atomic actions as introduced in the previous example. Assume that the set of actions $\mathcal{A}$ is defined as $\mathcal{B}(A)$, the set of bags over A. Figure 3.6(a) shows the process corresponding to the process in Figure 3.4(d). It represents a system that receives a message, performs two processing actions in any order, and returns the processed message to the environment. The occurrences of the two processing actions are causally *dependent*. The process of Figure 3.6(a) does *not* model a system that may perform two causally *independent* processing actions. Such a system is represented by the process depicted in Figure 3.6(b). It performs the two processing actions in any order *or* simultaneously. Thus, in a step semantics it is possible to distinguish concurrency from non-determinism.

The above example claims that the two processes depicted in Figure 3.6 are different. This raises the question when precisely processes are equivalent or not. To answer this question, the following definition introduces the so-called bisimilarity relation on processes, which is the basic semantic equivalence used throughout this chapter. It states that two processes are equivalent if and only if they can copy each others behavior and if they have the same termination options. The notion of bisimilarity was originally introduced in [50]. In [26, 27], bisimilarity is studied in more detail and a comparison with other semantic equivalences is made.

Definition 3.8. (Bisimilarity) A binary relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is called a *bisimulation* if and only if, for any $p, p', q, q' \in \mathcal{P}$ and $\alpha \in \mathcal{A}$,

i) $p\mathcal{R}q \wedge p \xrightarrow{\alpha} p' \;\Rightarrow\; (\exists q' : q' \in \mathcal{P} : q \xrightarrow{\alpha} q' \wedge p'\mathcal{R}q')$, *ii*) $p\mathcal{R}q \wedge q \xrightarrow{\alpha} q' \;\Rightarrow\; (\exists\, p' : p' \in \mathcal{P} : p \xrightarrow{\alpha} p' \wedge p'\mathcal{R}q')$, and

iii) $p\mathcal{R}q \;\Rightarrow\; (\downarrow p \Leftrightarrow \downarrow q)$.

Two processes p and q are called *bisimilar*, denoted $p \sim q$, if and only if there exists a bisimulation $\mathcal{R}$ such that $p\mathcal{R}q$.

Property 3.9. *Bisimilarity, $\sim$, is an equivalence relation.*

Proof. Reflexivity of bisimilarity follows from the fact that the identity relation on processes is a bisimulation. Symmetry follows immediately from the definition of bisimilarity. Transitivity follows from the fact that the relation composition of two bisimulations is again a bisimulation. □

An important property of bisimilarity is that it preserves moments of choice in processes as well as deadlocks.

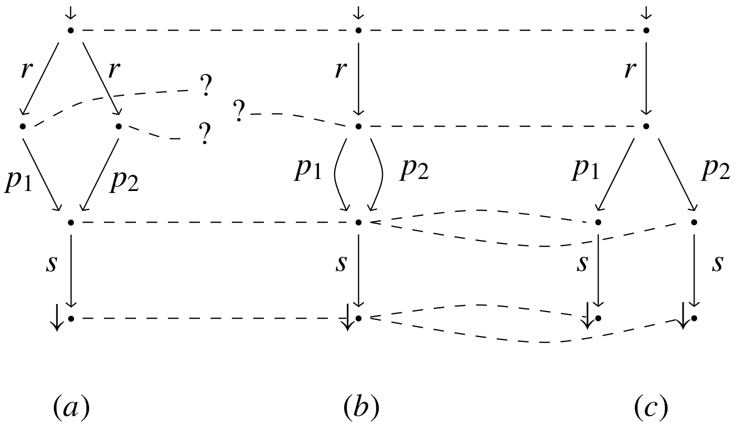


Figure 3.10: Bisimilar or not?

Example 3.11. As before, let $(\mathcal{P}, \mathcal{A}, \longrightarrow, \downarrow)$ be some process space, with $\mathcal{A}$ defined as the set of atomic actions A of Example 3.5. Consider the three processes depicted in Figure 3.10. Although all three are different, they appear to behave similarly. In all three cases, the process performs one out of two possible processing actions on the incoming message. In the process of Figure 3.10(a), the choice is made immediately upon receipt of the message. In process (b), the choice is made *after* the message has been received. The dashed lines depict an attempt to construct a bisimulation. It is not difficult to see that it is impossible to define a bisimulation between the two processes. As mentioned above, bisimilarity preserves moments of choice. Thus, the two processes of Figure 3.10(a) and (b) are not bisimilar. Bisimilarity provides the basis for a *branching-time* semantics for concurrent systems. The processes depicted in Figure 3.10(b) and (c), on the other hand, are bisimilar. The dashed lines depict what pairs of process states construct a bisimulation. It is straightforward to check that it satisfies the conditions of Definition 3.8. The two processes have the same moments of choice. The only difference is that in process (c) the two branches do not join, whereas they do come together in the process depicted in (b).

Example 3.12. Consider again Example 3.5. None of the processes in Figure 3.4 is bisimilar to any of the others. The fact that the two processes in Figure 3.4(a) and (b) are different conforms to the claim that bisimilarity preserves deadlocks.

Consider Example 3.7. It is not possible to construct a bisimulation between the two processes in Figure 3.6. This confirms the remark made in Example 3.7 that in a step semantics it is possible to distinguish concurrency from non-determinism.

If bisimilarity is defined on a process space where the actions are bags over some set of atomic actions, then the resulting equivalence is often called *step bisimilarity*. The concept of step bisimilarity dates back at least to [48].

4 Labeled Place/Transition Nets

Petri-net theory is one of the most well-known examples of a partial-order theory for modeling and analyzing concurrent systems. Petri nets were introduced in 1962 by Carl Adam Petri [52]. Since then, Petri-net theory has been extended in many ways and applied to many kinds of problems. Its popularity is due to both its easy-to-understand graphical representation of nets and its potential as a technique for formally analyzing concurrent systems. This section introduces so-called labeled Place/Transition nets, which is a class of Petri nets illustrative for any other Petri-net formalism.

Let U be some universe of identifiers; let L be some set of labels.

Definition 4.1. (Labeled P/T net) An L-labeled Place/Transition net, or simply labeled P/T net, is a tuple (P, T, F, W, ℓ) where

i) $P \subseteq U$ is a non-empty, finite set of *places*;

ii) $T \subseteq U$ is a non-empty, finite set of *transitions* such that $P \cap T = \emptyset$;

iii) $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the *flow relation*;

iv) $W : F \to \mathbb{N}$ is a *weight function*;

v) $\ell : T \to L$ is a *labeling function*.

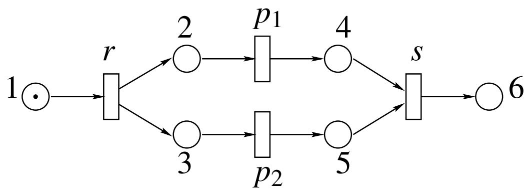


Figure 4.2: An example of a labeled P/T net.

Example 4.3. As mentioned, labeled P/T nets also have a graphical representation. Figure 4.2 shows a labeled P/T net. Places are depicted by circles, transitions by rectangles, and the flow relation by arcs. The weight function is represented by labeling arcs with their weights; weights equal to one are omitted. Attached to each place is its identifier. Attached to each transition is its label. Transition identifiers are usually omitted. They are only included if it is necessary to distinguish different transitions with the same label. The small black dot residing in place 1 is called a token. Tokens are introduced below.

Let (P, T, F, W, ℓ) be a labeled P/T net. Elements of $P \cup T$ are referred to as *nodes*. A node $x \in P \cup T$ is called an *input node* of another node $y \in P \cup T$ if and only if there exists a directed arc from x to y; that is, if and only if xFy. Node x is called an *output node* of y if and only if there exists a directed arc from y to x. If an input or output node x is a place in P, it is called an input place or output place; if it is a transition, it is called an input or an output transition. Two auxiliary functions $\boldsymbol{i}, \boldsymbol{o} : (P \cup T) \to \mathcal{B}(P \cup T)$ assign to each node its *bag* of input and output nodes, respectively. For any $x \in P \cup T$, $\boldsymbol{i}x = [y^{W(y,x)} \mid yFx]$ and $\boldsymbol{o}x = [y^{W(x,y)} \mid xFy]$.

Example 4.4. Consider again the P/T net in Figure 4.2. The bag of input places of transition r is the singleton bag containing place 1; that is, $\boldsymbol{i}r = [1]$. The bag of output places of r is $[2, 3]$; $\boldsymbol{o}r = [2, 3]$. The bag of output transitions of place 6 is empty; $\boldsymbol{o}6 = \mathbf{0}$.

A labeled P/T net as defined above is a static structure. Labeled P/T nets also have a *behavior*. The behavior of a net is determined by its structure and its *state*. To express the state of a net, its places may contain *tokens*. In labeled P/T nets, tokens are nothing more than simple markers. (See the graphical representation of a labeled P/T net in Figure 4.2.) The state of a net is defined as the distribution of the tokens over the places. In Petri-net theory, the state of a net is often called the *marking* of the net.

Definition 4.5. (Marked, labeled P/T net) A *marked*, L-labeled P/T net is a pair (N, m), where $N = (P, T, F, W, \ell)$ is an L-labeled P/T net and where m is a bag over P denoting the marking of the net. The set of all marked, L-labeled P/T nets is denoted $\mathcal{N}$.

The behavior of marked, labeled P/T nets is defined by a so-called *firing rule*, which is simply a transition relation defining the change in the state of a marked net when executing an action. To define a firing rule, it is necessary to formalize when a net is allowed to execute a certain action. The following definitions form the basis for a total-order semantics for labeled P/T nets.

Definition 4.6. (Transition enabling) Let (N, m) with $N = (P, T, F, W, \ell)$ be a marked, labeled P/T net in $\mathcal{N}$. A transition $t \in T$ is *enabled*, denoted $(N, m)[t\rangle$, if and only if each of its input places p contains at least as many tokens as the cardinality of p in $\boldsymbol{i}t$. That is, $(N, m)[t\rangle \Leftrightarrow \boldsymbol{i}t \leq m$.

Example 4.7. In the marked net of Figure 4.2, transition r is enabled. None of the other transitions is enabled.

When a transition t of a labeled P/T net is enabled, the net can *fire* this transition. Upon firing, t removes $W(p, t)$ tokens from each of its input places p; it adds $W(t, p)$ tokens to each of its output places p. This means that, upon firing t, the marked net (N, m) changes into another marked net $(N, m - \boldsymbol{i}t \uplus \boldsymbol{o}t)$. When firing t, the labeled P/T net executes an *action*, defined by its label. That is, when firing transition t, the P/T net executes the action $\ell(t)$.

Definition 4.8. (Total-order firing rule) The *total-order firing rule* $\_\,[\_\rangle^t\,\_ \subseteq \mathcal{N} \times L \times \mathcal{N}$ is the smallest relation satisfying for any marked, labeled P/T net (N, m) in $\mathcal{N}$, where $N = (P, T, F, W, \ell)$, and any transition $t \in T$,

$$(N, m)[t\rangle \;\Rightarrow\; (N, m)\,[\ell(t)\rangle^t\,(N, m - \boldsymbol{i}t \uplus \boldsymbol{o}t).$$

Tokens that are removed from the marking when firing a transition are often referred to as *consumed* tokens or the *consumption* of a transition; tokens that are added to the marking are referred to as *produced* tokens or the *production* of a transition.

Example 4.9. In the P/T net of Figure 4.2, firing transition r changes the marking from [1] to [2, 3]. In the new marking, both transitions p_1 and p_2 are enabled. As a result, *either* p_1 can fire yielding marking [3, 4] *or* p_2 can fire yielding marking [2, 5]. After firing one of these two transitions, the other one is the only transition in the net that can still fire. Independent of the order in which p_1 and p_2 are fired the resulting marking is [4, 5]. In this marking, transition s is enabled; firing s yields marking [6].

As explained in the introduction to this chapter, a semantics of a formal language for describing concurrent systems defines a process for each expression in the formal language. The definitions given so far provide the following semantics for the language of labeled P/T nets. The usual Petri-net semantics does not distinguish successful termination and deadlock. Recall the notions of a process space and bisimilarity from Definitions 3.1 and 3.8.

Definition 4.10. (Total-order semantics) The process space $(\mathcal{N}, L, [\,\rangle^t, \emptyset)$, with the set of marked, labeled P/T nets as processes, the set of labels as actions, the total-order firing rule as the transition relation, and the empty termination predicate, combined with bisimilarity as the semantic equivalence defines a (branching-time) total-order semantics for labeled P/T nets.

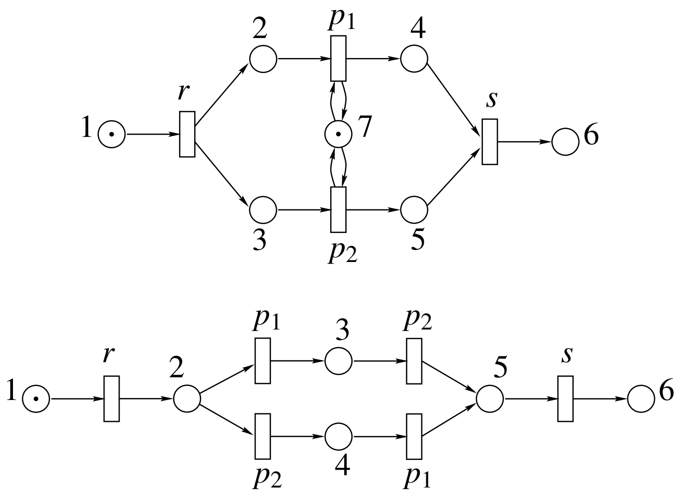


Figure 4.11: Two examples of P/T nets without true concurrency.

Example 4.12. In the semantics of Definition 4.10, the process corresponding to the labeled P/T net of Figure 4.2 is the one depicted in Figure 3.4(d), except that it does not terminate successfully after the execution of action s. (Note that dots in Figure 3.4(d) correspond to marked, labeled P/T nets; labeled arrows represent the total-order firing rule $[\,\rangle^t$.) Figure 4.11 shows two other P/T nets with exactly the same semantics. Intuitively, the P/T net of Figure 4.2 models a system with true concurrency, whereas the two P/T nets of Figure 4.11 do not. In the P/T nets of Figure 4.11, the actions p_1 and p_2 may happen in any order, but not simultaneously. However, as one might expect by now, in the total-order semantics of Definition 4.10, it is not possible to distinguish the three nets.

Note that, if desirable, the process-theoretic framework of the previous section is sufficiently flexible to express that the P/T nets of Figures 4.2 and 4.11 terminate successfully after firing transition s. Let $\downarrow_6$ be the termination predicate defined as follows: $\downarrow_6 = \{(N, m) \in \mathcal{N} \mid 6 \in m\}$. In the process space $(\mathcal{N}, L, [\,\rangle^t, \downarrow_6)$, the semantics of all the three P/T nets of Figures 4.2 and 4.11 is the process depicted in Figure 3.4(d).

The total-order semantics of Definition 4.10 can be generalized to a step semantics for labeled P/T nets in a very natural way. It suffices to allow that multiple transitions can fire at the same time.

A marking of a labeled P/T net enables a *set* of transitions if and only if all transitions in the set are simultaneously enabled. The auxiliary functions $\boldsymbol{i}$ and $\boldsymbol{o}$ are lifted to sets of transitions as follows: For any labeled P/T net $N = (P, T, F, W, \ell)$ and set $S \subseteq T$, $\boldsymbol{i}S = (\uplus t : t \in S : \boldsymbol{i}t)$ and $\boldsymbol{o}S = (\uplus t : t \in S : \boldsymbol{o}t)$.

Definition 4.13. (Set enabling) Let (N, m) be a marked, labeled P/T net in $\mathcal{N}$, where $N = (P, T, F, W, \ell)$. A set $S \subseteq T$ is *enabled*, denoted $(N, m)[S\rangle$, if and only if $\boldsymbol{i}S \leq m$.

Let $N = (P, T, F, W, \ell)$ be an L-labeled P/T net. Any set $S \subseteq T$ that is enabled in some given marking of N determines a possible *step* of N. The transitions in the set S can fire simultaneously; the *bag* of labels of transitions in S forms the step that the net performs when firing S. Note that the set of all subsets of a given set X, also called the powerset of X, is denoted $\mathcal{P}(X)$. The labeling function $\ell : \mathcal{P}(T) \rightarrow \mathcal{B}(L)$ on sets of transitions of N is defined as follows: For any set $S \subseteq T$, $\ell(S) = (\uplus t : t \in S : [\ell(t)])$.

Definition 4.14. (Step firing rule) The *step firing rule* $\_\,[\_\rangle^s\,\_ \subseteq \mathcal{N} \times \mathcal{B}(L) \times \mathcal{N}$ is the smallest relation satisfying for any marked, labeled P/T net (N, m) in $\mathcal{N}$, where $N = (P, T, F, W, \ell)$, and any $S \subseteq T$,

$$(N, m)[S\rangle \;\Rightarrow\; (N, m)\,[\ell(S)\rangle^s\,(N, m - \boldsymbol{i}S \uplus \boldsymbol{o}S).$$

The step firing rule forms the basis for the following step semantics for labeled P/T nets.

Definition 4.15. (Step semantics) The process space $(\mathcal{N}, \mathcal{B}(L), [\,\rangle^s, \emptyset)$, with the bags over L as actions and the step firing rule as the transition relation, combined with bisimilarity as the semantic equivalence defines a step semantics for labeled P/T nets.

Note that the semantic equivalence resulting from Definitions 3.8 (Bisimilarity) and 4.15 (Step semantics) is the step-bisimilarity equivalence introduced in the previous section. Thus, the semantics of Definition 4.15 can be characterized as a branching-time, partial-order semantics.

Example 4.16. Recall the two processes depicted in Figure 3.6. Consider the two variants of these processes that cannot terminate successfully after performing the last action. Assuming the semantics of Definition 4.15 for labeled P/T nets, the process in Figure 3.6(b) is the semantics of the labeled P/T net of Figure 4.2. The other process depicts the semantics of the two P/T nets in Figure 4.11. Thus, in the step semantics of Definition 4.15, it is possible to distinguish concurrency from non-determinism.

This section has given a brief introduction to elementary Petri-net theory. The class of labeled P/T nets has been introduced, as well as two different semantics for this class of nets. Labeled P/T nets handle causal relationships between actions very explicitly via places and tokens and are, therefore, well suited for developing a partial-order framework for reasoning about concurrent systems. The introduction is not exhaustive. In particular, there are several other approaches to defining a partial-order semantics for P/T nets than the one chosen in this section. However, the framework of this section is sufficient for the remainder of this chapter. Good starting points for further reading on Petri nets are [17, 46, 51, 55, 56, 57]. In the remainder of this chapter, we introduce a process-algebraic theory with both a total-order and a step semantics. In addition, we study several concepts from Petri-net theory in the algebraic partial-order framework.

5 Process Algebra

The goal of this section is to illustrate how process algebra in the style of the Algebra of Communicating Processes (ACP) can be used to reason about concurrent systems in both a total-order and a partial-order setting. The theory ACP originates from [13]. Good introductions to ACP-style process algebra can be found in [5, 6]. Other well-known process-algebraic theories are CCS [43, 44] and CSP [35]. Specifications and verifications in ACP-style process algebra are based on an equational style of reasoning, whereas CCS and CSP emphasize model-based reasoning. In the latter case, the starting point of the theory is a semantic framework of processes in the style of Section 3. The goal is to find equational laws that are valid in this semantic framework. In the former case, the starting point is an equational theory, which may have several semantic interpretations. For a detailed comparison of ACP, CCS, and CSP, the reader is referred to [6, Chapter 8]. This section presents the theory ACP and a few of its extensions. It illustrates how the communication mechanism of ACP can be used to define a step semantics for concurrent systems. It is also defined when a process-algebraic theory and its semantics are said to be interleaving. Both the total-order and the partial-order framework developed in this section can be characterized as interleaving frameworks.

5.1 The equational theory

The Algebra of Communicating Processes Any ACP-style process-algebraic theory is essentially an *equational theory*. An equational theory consists of a *signature* and a set of *axioms*. The signature defines the *sorts* of the theory, a set of *variables* for each sort, and the *functions* of the theory. Functions and variables can be used to construct *terms*. Terms not containing any variables are called *closed* terms. The axioms of the theory determine which terms are equal. A process-algebraic theory usually has only a single sort; terms of this sort represent processes. A 0-ary function is often called a *constant*; other functions are often called *operators*.

The signature and axioms of the equational theory ACP(AC, γ) are given in Table 5.1. The theory is parameterized by a set of constants AC, which is a set of actions, and a partial function $\gamma : AC \times AC \to AC$, which is a communication function. The first part of Table 5.1 lists the sorts in the signature of ACP(AC, γ); the second part defines the constants and the operators in the signature. The third entry of Table 5.1 gives the variables and lists the axioms of the equational theory. Note that new variables may be introduced any time when necessary. An informal explanation of the operators and the axioms is given below.

ACP(AC, γ)

$\boldsymbol{P}$;

$AC \subseteq \boldsymbol{P}$; $\quad \delta : \boldsymbol{P}$; $\quad \partial_H : \boldsymbol{P} \to \boldsymbol{P}$; $\quad \_ + \_, \_ \cdot \_, \_ \parallel \_, \_ \mathbin{⌊\!⌊} \_, \_ \mid \_ : \boldsymbol{P} \times \boldsymbol{P} \to \boldsymbol{P}$;

$x, y, z : \boldsymbol{P}$;

$$
\begin{array}{lllll}
x + y = y + x & A1 & x \parallel y = x \mathbin{⌊\!⌊} y + y \mathbin{⌊\!⌊} x + x \mid y & CM1 \\
(x + y) + z = x + (y + z) & A2 & e \mathbin{⌊\!⌊} x = e \cdot x & CM2 \\
x + x = x & A3 & e \cdot x \mathbin{⌊\!⌊} y = e \cdot (x \parallel y) & CM3 \\
(x + y) \cdot z = x \cdot z + y \cdot z & A4 & (x + y) \mathbin{⌊\!⌊} z = x \mathbin{⌊\!⌊} z + y \mathbin{⌊\!⌊} z & CM4 \\
(x \cdot y) \cdot z = x \cdot (y \cdot z) & A5 & e \cdot x \mid f = (e \mid f) \cdot x & CM5 \\
x + \delta = x & A6 & e \mid f \cdot x = (e \mid f) \cdot x & CM6 \\
\delta \cdot x = \delta & A7 & e \cdot x \mid f \cdot y = (e \mid f) \cdot (x \parallel y) & CM7 \\
 & & (x + y) \mid z = x \mid z + y \mid z & CM8 \\
e \mid f = f \mid e & C1 & x \mid (y + z) = x \mid y + x \mid z & CM9 \\
(e \mid f) \mid g = e \mid (f \mid g) & C2 & & \\
\delta \mid e = \delta & C3 & e \notin H \Rightarrow \partial_H(e) = e & D1 \\
 & & e \in H \Rightarrow \partial_H(e) = \delta & D2 \\
\gamma(a, b) \text{ defined} \Rightarrow a \mid b = \gamma(a, b) & CF1 & \partial_H(x + y) = \partial_H(x) + \partial_H(y) & D3 \\
\gamma(a, b) \text{ undefined} \Rightarrow a \mid b = \delta & CF2 & \partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y) & D4
\end{array}
$$

Table 5.1: The equational theory ACP(AC, γ).

As mentioned, AC is a set of actions. Terms of sort $\boldsymbol{P}$ represent processes. Each action is a process, namely the process that can only execute the action and then terminates.

The two basic operators of ACP(AC, γ) are $+$ and $\cdot$, denoting alternative composition or choice and sequential composition, respectively. These two operators are elementary in describing the behavior of sequential processes. Sequential composition binds stronger than choice. Choice and sequential composition are axiomatized by Axioms $A1$ through $A5$. Most of these axioms are self-explanatory. Only Axiom $A4$ might need some explanation. It states the *right distributivity* of sequential composition over choice. The converse, left distributivity, is not an axiom of the theory. As a result, processes with different moments of choice are distinguished.

The constant δ stands for *inaction*, often also called *deadlock*. However, the former name is best suited, as follows from Axiom $A6$. It says that a process which can choose between some behavior x and doing nothing is equivalent to the process that has no choice and can only do x. Hence, in the context of a choice, δ is not a true deadlock. Axiom $A7$ shows that δ is a deadlock in the context of a sequential composition.

Axioms $CM1$ through $CM9$ axiomatize the behavior of concurrent processes. Constants e and f range over $AC \cup \{\delta\}$. Thus, an axiom such as $CM2$ containing the constant e is actually an axiom *scheme*; the equational theory ACP(AC, γ) contains one axiom for each possible constant $e \in AC \cup \{\delta\}$.

The parallel-composition operator $\|$, often called the *merge* operator, denotes the parallel execution of its operands. It is axiomatized using two auxiliary operators, namely $\lfloor\!\lfloor$, called the *left* merge, and $|$, called the *communication* merge. The left merge has the same meaning as the merge except that the left process must perform the first action. The communication merge also denotes the parallel execution of its operands, this time with the restriction that they must synchronize on their first action. Sequential composition binds stronger than merge, left merge, and communication merge, whereas choice binds weaker.

The result of the synchronization of two actions is defined by the communication function γ, as expressed by Axioms $CF1$ and $CF2$. Constants a and b range over AC. Operator $|$ and function γ are named *communication* merge and function, respectively, because the standard interpretation in ACP-style process algebra of the synchronization of two actions is (synchronous) communication. However, in the remainder, it is shown that the communication function can also be used to define a step semantics for processes. This means that the synchronization of actions conforms to simultaneous execution.

Axioms $C1$, $C2$, and $C3$, where e, f, and g range over $AC \cup \{\delta\}$, state that the synchronization of actions is commutative and associative and that actions cannot synchronize with the inaction constant δ. To the experienced reader, it might be clear that the combination of Axioms $C1$, $C2$, $C3$, $CF1$, and $CF2$ can only be consistent if the communication function γ is associative and commutative. This requirement becomes explicit in the next subsection, where a model of the equational theory of this subsection is constructed. (See the proof of Theorem 5.19.)

Finally, the equational theory ACP(AC, γ) contains the so-called *encapsulation* operators. For any subset of actions $H \subseteq AC$, the operator ∂_H renames all actions in H in a process to the constant δ, as expressed by Axioms $D1$ through $D4$. Thus, the execution of actions in H is effectively blocked. Constant e ranges over $AC \cup \{\delta\}$. In ACP-style process algebra, an encapsulation operator is usually used to enforce communication between actions. This can be achieved by encapsulating the actions that must participate in a communication when they occur in isolation.

An equational theory such as ACP(AC, γ) of Table 5.1 provides the basis for reasoning about concurrent processes. The set of axioms of an equational theory defines an equivalence relation on process terms, called *derivability*. For any process terms x and y in some given equational theory X, $X \vdash x = y$ denotes that $x = y$ can be derived from the axioms of X. Derivability in an equational theory is defined as follows. First, the axioms themselves can be derived from the axioms of the theory. Second, since derivability is an equivalence relation, it is reflexive, symmetric, and transitive. Third, if an equation is derivable from the axioms, then also any equation obtained by substituting terms for variables in this equation is derivable. Finally, any equation obtained by replacing a term in an arbitrary context by another derivably equivalent term is also derivable from the theory. The axioms of an equational theory must be chosen in such a way that derivability defines a meaningful equivalence relation on processes. Below, it is explained that in the case of the theory ACP(AC, γ) derivability corresponds to bisimilarity.

Example 5.2. Let A be some set of actions that includes the actions p_1, p_2, r, and s; let γ be some arbitrary communication function. It can be shown that $\mathrm{ACP}(A, \gamma) \vdash r \cdot ((p_1 + p_2) \cdot s) = r \cdot (p_1 \cdot s + p_2 \cdot s)$. The first step is to substitute p_1, p_2, and s for the variables x, y, and z in Axiom $A4$, which shows that $\mathrm{ACP}(A, \gamma) \vdash t_1 = t_2$, where t_1 is the term $(p_1 + p_2) \cdot s$ and t_2 denotes the term $p_1 \cdot s + p_2 \cdot s$. The second and final step consists of an application of the context rule explained above: Replacing term t_1 in $r \cdot t_1$ with the equivalent term t_2 yields the desired result.

The intuitive meaning of the operators of ACP(AC, γ), the axioms, and the induced equivalence relation given above can be analyzed further by defining an operational semantics. However, before doing so, a few extensions of ACP(AC, γ) are discussed that are needed in the remainder.

Renaming The encapsulation operators of ACP(AC, γ) form a subclass of a much larger class of operators, namely the algebraic *renaming operators*. A renaming operator applied to a process term simply renames actions in the term according to a *renaming function*. A renaming function $\boldsymbol{f} : (AC \cup \{\delta\}) \to (AC \cup \{\delta\})$ is any function with the restriction that $\boldsymbol{f}(\delta) = \delta$. This restriction means that the inaction process can be the result of a renaming, but that it cannot be renamed itself. It is included in the domain of renaming functions only for the sake of convenience. The set of all renaming functions is denoted RF. The Algebra of Communicating Processes with renaming, abbreviated ACP + RN, is defined in Table 5.3. It is parameterized with a set of actions AC and a communication function γ. The first entry of Table 5.3 says that the equational theory (ACP + RN)(AC, γ) extends the theory ACP(AC, γ) of Table 5.1. The other entries of Table 5.3 have the same meaning as the corresponding entries in Table 5.1. In Table 5.3, function $\boldsymbol{f}$ ranges over the set of renaming functions RF and constant e ranges over $AC \cup \{\delta\}$.

(ACP + RN)(AC, γ)	
ACP(AC, γ)	
$\rho_f : \boldsymbol{P} \to \boldsymbol{P}$;	
$x, y : \boldsymbol{P}$;	
$\rho_f(e) = \boldsymbol{f}(e)$	$RN1$
$\rho_f(x + y) = \rho_f(x) + \rho_f(y)$	$RN2$
$\rho_f(x \cdot y) = \rho_f(x) \cdot \rho_f(y)$	$RN3$

Table 5.3: The equational theory (ACP + RN)(AC, γ).

As mentioned, the encapsulation operators of ACP(AC, γ) belong to the class of renaming operators. Let $H \subseteq AC$ be some set of actions. Let $\boldsymbol{e}(H) : (AC \cup \{\delta\}) \to (AC \cup \{\delta\})$ be a renaming function in RF defined as follows: $\boldsymbol{e}(H)(\delta) = \delta$, for any $a \in H$, $\boldsymbol{e}(H)(a) = \delta$, and, for any $a \in AC \setminus H$, $\boldsymbol{e}(H)(a) = a$. It is not difficult to see that the encapsulation operator ∂_H and the renaming operator $\rho_{\boldsymbol{e}(H)}$ are identical.

Iteration In practice, many concurrent processes exhibit some kind of recursive behavior. A restricted form of recursive behavior is *iterative* behavior. Table 5.4 defines the Algebra of Communicating Processes with iteration and renaming, abbreviated ACP* + RN.

(ACP* + RN)(AC, γ)	
(ACP + RN)(AC, γ)	
$\_ ^* \_ : \boldsymbol{P} \times \boldsymbol{P} \to \boldsymbol{P}$;	
$x, y, z : \boldsymbol{P}$;	
$x^* y = x \cdot (x^* y) + y$	$BKS1$
$x^* (y \cdot z) = (x^* y) \cdot z$	$BKS2$
$x^* (y \cdot ((x + y)^* z) + z) = (x + y)^* z$	$BKS3$
$\rho_f(x^* y) = \rho_f(x)^* \rho_f(y)$	$BKS4$

Table 5.4: The equational theory (ACP* + RN)(AC, γ).

The *binary Kleene star* * was introduced in ACP-style process algebra in [11]. It is adapted from the original star operator as introduced in [40]. In [11], also the Axioms $BKS1$ through $BKS4$ appear, although $BKS4$ is formulated only for the specific renaming operators encapsulation and abstraction. Abstraction in process algebra is not covered in this chapter. An extensive study of iteration in ACP-style process algebra can be found in [12].

In Table 5.4, function f ranges over the set of renaming functions RF as introduced above. Axiom $BKS1$ is characteristic for iteration. It states that behavior x is iterated an arbitrary number of times before continuing with behavior y. (Axiom $BKS3$ is a sophisticated axiom which is needed to get a complete axiomatization of bisimilarity in a setting much simpler than the equational theory developed in this subsection; see [11, 12] for more details.) The binary Kleene star has the same binding priority as sequential composition.

Recursive equations Since often verifications in an equational theory result in an equation that is recursive in some given process, it is useful to have a means to determine solutions for such recursive equations. Table 5.5 presents the theory $(\mathrm{ACP}^* + \mathrm{RN} + RSP^*)(AC, \gamma)$ which extends $(\mathrm{ACP}^* + \mathrm{RN})(AC, \gamma)$ with a so-called recursion principle. The *Recursive Specification Principle* for the binary Kleene star (RSP^*) is a derivation rule which gives a solution in terms of the binary Kleene star for some restricted set of recursive equations.

$(\mathrm{ACP}^* + \mathrm{RN} + RSP^*)(AC, \gamma)$	
$(\mathrm{ACP}^* + \mathrm{RN})(AC, \gamma)$	
-	
$x, y, z : \boldsymbol{P}$;	
$\dfrac{x = y \cdot x + z}{x = y^* z}$	RSP^*

Table 5.5: The equational theory $(\mathrm{ACP}^* + \mathrm{RN} + RSP^*)(AC, \gamma)$.

Observe that, compared to theory $(\mathrm{ACP}^* + \mathrm{RN})(AC, \gamma)$, equational theory $(\mathrm{ACP}^* + \mathrm{RN} + RSP^*)(AC, \gamma)$ has no new constants or operators. Another interesting observation is that, in theory $(\mathrm{ACP}^* + \mathrm{RN} + RSP^*)(AC, \gamma)$, the axioms $BKS2$, $BKS3$, and $BKS4$ are derivable from the other axioms of the theory. (It is an interesting exercise for the reader to give the three derivations.)

Standard concurrency A final extension of the equational theory developed so far in this subsection is the extension with the so-called *axioms of standard concurrency*. Table 5.6 presents the Algebra of Communicating Processes with iteration, renaming, the Recursive Specification Principle for the binary Kleene star, and standard concurrency, abbreviated $\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC}$. The equational theory $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ has no new constants or operators. The extension solely consists of six new axioms, formulating some desirable properties of the merge, left merge, and communication merge.

A consequence of the extension of the equational theory with the axioms of standard concurrency is that it allows the straightforward formulation of a general so-called *expansion theorem*. This theorem is very useful for simplifying many calculations. It states how a parallel composition can be expanded into an alternative composition. Recall from Section 2 that the commutative and associative binary process-algebraic operator $+$ with unit element δ generalizes to a quantifier in a standard way. Furthermore, note that the binary operators $\|$ and $|$ in the equational theory $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ are also commutative and associative but that they do not have unit elements. Nevertheless, general quantifier notation may be used for these two operators. For example, for some positive natural number n, and operands $x_0, \ldots, x_n$, the quantifier notation $(\| i : 0 \leq i \leq n : x_i)$ is used as a shorthand notation for $x_0 \| \ldots \| x_n$, where redundant brackets are omitted, and the notation $(\| i : i = k : x_i)$, for some k with $0 \leq k \leq n$, denotes x_k.

(ACP* + RN + *RSP** + SC)(*AC*, γ)	
(ACP* + RN + *RSP**)(*AC*, γ)	
-	
$x, y, z : \boldsymbol{P}$;	
$x \mid y = y \mid x$	*SC*1
$x \parallel y = y \parallel x$	*SC*2
$x \mid (y \mid z) = (x \mid y) \mid z$	*SC*3
$(x \mathbin{⌊\!⌊} y) \mathbin{⌊\!⌊} z = x \mathbin{⌊\!⌊} (y \parallel z)$	*SC*4
$(x \mid y) \mathbin{⌊\!⌊} z = x \mid (y \mathbin{⌊\!⌊} z)$	*SC*5
$(x \parallel y) \parallel z = x \parallel (y \parallel z)$	*SC*6

Table 5.6: The equational theory (ACP* + RN + *RSP** + SC)(AC, γ).

Theorem 5.7. (Expansion) *Let $x_0, \ldots, x_n$ be process terms of sort $\boldsymbol{P}$, where n is a positive natural number. Let I be the set $\{0, \ldots, n\}$.*

$$(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma) \vdash$$
$$(\|i : i \in I : x_i) = (+J : \emptyset \subset J \subset I : (|j : j \in J : x_j) \mathbin{⌊\!⌊} (\|i : i \in I \setminus J : x_i)) + (|i : i \in I : x_i).$$

Proof. It is straightforward to prove the desired result by induction on n. □

Example 5.8. Let A be some set of actions that includes the actions a, b, c, and d. Function $\gamma : A \times A \to A$ is the communication function defining that the communication of actions a and b yields action c. In set notation, $\gamma = \{((a, b), c), ((b, a), c)\}$. In its simplest form, the expansion theorem given above corresponds to Axiom *CM*1 of Table 5.1. The following derivation in the equational theory (ACP* + RN + *RSP** + SC)(A, γ) shows the application of the expansion theorem to a parallel composition of three components.

$$a \parallel b \parallel d$$
$$= \quad \{ \text{ Theorem 5.7 (Expansion) } \}$$
$$a \mathbin{⌊\!⌊} (b \parallel d) + b \mathbin{⌊\!⌊} (a \parallel d) + d \mathbin{⌊\!⌊} (a \parallel b) + (a \mid b) \mathbin{⌊\!⌊} d + (a \mid d) \mathbin{⌊\!⌊} b + (b \mid d) \mathbin{⌊\!⌊} a + (a \mid b \mid d)$$
$$= \quad \{ \text{ Exercise for the reader } \}$$
$$a \cdot (b \cdot d + d \cdot b) + b \cdot (a \cdot d + d \cdot a) + d \cdot (a \cdot b + b \cdot a + c) + c \cdot d$$

Note that the above expansion theorem allows the synchronization of an arbitrary number of processes. In the standard literature on process algebra [5, 6], only a simpler version of the expansion theorem is formulated. The simplifying assumption, called the handshaking axiom, is that only pairs of processes can synchronize. In the current setting, it is desirable to allow arbitrary synchronization, because one possible interpretation of synchronization corresponds to concurrency. There is no obvious reason to restrict concurrency to two processes.

Interleaving theory As explained in the introduction, a process-algebraic theory with an expansion theorem is said to be interleaving. In this paragraph, we formalize the notion of an interleaving theory in terms of more elementary concepts. To the best of our knowledge, such a formalization does not yet exist in the literature on concurrency theory. (The definition of a (non-)interleaving process-algebraic theory in [2] is kept informal.) Therefore, the definition given in this paragraph is phrased in general terms such that it is applicable to any ACP-style equational theory. It can also be easily adapted to other process-algebraic frameworks such as CCS and CSP.

Let $X(AC)$ be some ACP-style equational theory which is parameterized with a set of actions AC. Note that it may have more (unspecified) parameters such as a communication function. In addition, assume that $X(AC)$ has a single process sort $\boldsymbol{P}$ and that its signature contains at least the choice operator $+$, the sequential-composition operator $\cdot$, and the parallel-composition operator $\|$. The following format of process terms forms the basis of the formalization of the notion of an interleaving equational theory. Note that this format is a standard format defined in the process-algebraic literature [6].

Definition 5.9. (Head normal form) The set of process terms over the signature of $X(AC)$ in *head normal form*, denoted $\boldsymbol{H}(AC)$, is inductively defined as follows. First, each action constant in AC is an element of $\boldsymbol{H}(AC)$. Second, if the signature of $X(AC)$ contains the inaction constant δ, then also δ is an element of $\boldsymbol{H}(AC)$. Third, for any action a in AC and term x of sort $\boldsymbol{P}$, the term $a \cdot x$ is an element of $\boldsymbol{H}(AC)$. Finally, for any terms y and z in $\boldsymbol{H}(AC)$, the term $y + z$ is an element of $\boldsymbol{H}(AC)$.

An arbitrary term x of sort $\boldsymbol{P}$ is said to *have* a head normal form, denoted $HNF(x)$ if and only if there is a term y in $\boldsymbol{H}(AC)$ such that $X(AC) \vdash x = y$.

Example 5.10. Consider the equational theory ACP(AC, γ) of Table 5.1. As in Example 5.2, let A be some set of actions that includes the actions p_1, p_2, r, and s; let γ be some arbitrary communication function. The closed term $r \cdot ((p_1 + p_2) \cdot s)$ is in head normal form. Term $(p_1 + p_2) \cdot s$ is not in head normal form. However, it has a head normal form, because it is derivably equal to the term $p_1 \cdot s + p_2 \cdot s$, which is in head normal form. The term x of sort $\boldsymbol{P}$, where x is a variable, does not have a head normal form.

An equational theory is said to be interleaving if and only if each parallel composition has a head normal form under the assumption that its operands have head normal forms.

Definition 5.11. (Interleaving theory) The equational theory $X(AC)$ is an *interleaving theory* if and only if, for any terms x and y of sort $\boldsymbol{P}$, $HNF(x)$ and $HNF(y)$ implies $HNF(x \parallel y)$.

All the equational theories given so far in this subsection are interleaving theories. The proof of this result uses two auxiliary properties about head normal forms of compositions using the auxiliary operators left merge and communication merge.

Theorem 5.12. *Let AC be some set of actions; let γ be an arbitrary communication function. The equational theories* ACP(AC, γ), (ACP+RN)(AC, γ), (ACP*+RN)(AC, γ), (ACP*+RN+RSP^*)(AC, γ), *and* (ACP*+ RN + RSP^* + SC)(AC, γ) *are all interleaving theories.*

Proof. The proof is identical for each of the theories. Therefore, let x and y be terms of sort $\boldsymbol{P}$ of any of the above theories such that $HNF(x)$ and $HNF(y)$. It must be proven that $HNF(x \parallel y)$. It follows from Axiom $CM1$ of Table 5.1 that $x \parallel y = x \mathbin{⌊\!⌊} y + y \mathbin{⌊\!⌊} x + x \mid y$. Definition 5.9 implies that it suffices to prove that $HNF(x \mathbin{⌊\!⌊} y)$, $HNF(y \mathbin{⌊\!⌊} x)$, and $HNF(x \mid y)$, which follows immediately from Properties 5.13 and 5.14. $\square$

Property 5.13. *Let AC be some set of actions; let γ be an arbitrary communication function. For any terms x and y of sort $\boldsymbol{P}$ of any of the equational theories* ACP(AC, γ), (ACP+RN)(AC, γ), (ACP*+RN)(AC, γ), (ACP*+ RN + RSP^*)(AC, γ), *and* (ACP*+ RN + RSP^*+ SC)(AC, γ), *$HNF(x)$ implies $HNF(x \mathbin{⌊\!⌊} y)$.*

Proof. Let $\boldsymbol{H}(AC)$ be the set of terms in head normal form of any of the above theories. Let x and y be terms of sort $\boldsymbol{P}$ of the same theory such that $HNF(x)$. It must be proven that $HNF(x \mathbin{⌊\!⌊} y)$. According to Definition 5.9, there must be a term in $\boldsymbol{H}(AC)$ that is derivably equal to x. Thus, it suffices to prove that for any $z \in \boldsymbol{H}(AC)$, $HNF(z \mathbin{⌊\!⌊} y)$. Let z be a term in $\boldsymbol{H}(AC)$. Since the signature of any of the theories in this property contains the inaction constant δ, term z can be in any of four formats. The proof is by induction on the structure of term z. The symbol $\equiv$ denotes syntactical equivalence of terms.

i) Assume that $z \equiv \delta$. It follows that $\delta \mathbin{⌊\!⌊} y \overset{CM2, A7}{=} \delta$. Since $\delta \in \boldsymbol{H}(AC)$, it follows that $HNF(\delta \mathbin{⌊\!⌊} y)$, which completes the proof for the case $z \equiv \delta$.

ii) Assume that $z \equiv a$, for some action a in AC. It follows that $a \mathbin{⫽} y \overset{CM2}{=} a \cdot y$. Since $a \cdot y \in \boldsymbol{H}(AC)$, it follows that $HNF(a \mathbin{⫽} y)$, which completes the proof also in this case.

iii) Assume that $z \equiv a \cdot x_1$, for some action a in AC and some term x_1 of sort $\boldsymbol{P}$. It follows that $a \cdot x_1 \mathbin{⫽} y \overset{CM3}{=} a \cdot (x_1 \parallel y)$. Since $a \cdot (x_1 \parallel y) \in \boldsymbol{H}(AC)$, it follows that $HNF(a \cdot x_1 \mathbin{⫽} y)$.

iv) Finally, assume that $z \equiv z_1 + z_2$, for some terms z_1 and z_2 in $\boldsymbol{H}(AC)$. It follows that $(z_1 + z_2) \mathbin{⫽} y \overset{CM4}{=} z_1 \mathbin{⫽} y + z_2 \mathbin{⫽} y$. By induction, it follows that $HNF(z_1 \mathbin{⫽} y)$ and $HNF(z_2 \mathbin{⫽} y)$, which means that $HNF(z_1 \mathbin{⫽} y + z_2 \mathbin{⫽} y)$. Consequently, $HNF((z_1 + z_2) \mathbin{⫽} y)$, which completes the proof. □

Property 5.14. *Let AC be some set of actions; let γ be an arbitrary communication function. For any terms x and y of sort $\boldsymbol{P}$ of any of the equational theories* $\text{ACP}(AC, \gamma)$, $(\text{ACP}+\text{RN})(AC, \gamma)$, $(\text{ACP}^*+\text{RN})(AC, \gamma)$, $(\text{ACP}^*+\text{RN}+RSP^*)(AC, \gamma)$, *and* $(\text{ACP}^*+\text{RN}+RSP^*+\text{SC})(AC, \gamma)$, *$HNF(x)$ and $HNF(y)$ implies $HNF(x \mid y)$.*

Proof. Let $\boldsymbol{H}(AC)$ be the set of terms in head normal form of any of the above theories. Let x and y be terms of sort $\boldsymbol{P}$ of the same theory such that $HNF(x)$ and $HNF(y)$. It must be proven that $HNF(x \mid y)$. According to Definition 5.9, there must be terms in $\boldsymbol{H}(AC)$ that are derivably equal to x and y, respectively. Thus, it suffices to prove that for any terms x_0 and y_0 in $\boldsymbol{H}(AC)$, $HNF(x_0 \mid y_0)$. Let x_0 and y_0 be terms in $\boldsymbol{H}(AC)$. The proof is by induction on the structure of x_0.

i) Assume that $x_0 \equiv \delta$. It must be shown that $HNF(\delta \mid y_0)$. This result is shown by straightforward induction on the structure of term y_0.

(a) Assume that $y_0 \equiv \delta$ or that $y_0 \equiv a$, for some action a in AC. It follows that $\delta \mid y_0 \overset{C3}{=} \delta$. Thus, $HNF(\delta \mid y_0)$, which completes the proof in this case.

(b) Assume that $y_0 \equiv a \cdot y_1$, for some action a in AC and some term y_1 of sort $\boldsymbol{P}$. It follows that $\delta \mid a \cdot y_1 \overset{CM6,C3,A7}{=} \delta$, which completes the proof also in this case.

(c) Finally, assume that $y_0 \equiv z_1 + z_2$, for some terms z_1 and z_2 in $\boldsymbol{H}(AC)$. It follows that $\delta \mid (z_1 + z_2) \overset{CM9}{=} \delta \mid z_1 + \delta \mid z_2$. By induction, it follows that $HNF(\delta \mid z_1)$ and $HNF(\delta \mid z_2)$, which means that $HNF(\delta \mid (z_1 + z_2))$.

ii) Assume that $x_0 \equiv a$, for some action a in AC. Again, the desired result that $HNF(a \mid y_0)$ is proven by induction on the structure of y_0.

(a) Assume that $y_0 \equiv \delta$. It follows that $a \mid \delta \overset{C1,C3}{=} \delta$. Thus, $HNF(a \mid \delta)$.

(b) Assume that $y_0 \equiv b$, for some action b in AC. It follows from Axioms $CF1$ and $CF2$ and the fact that the result of the communication function is an action in AC that $HNF(a \mid b)$.

(c) Assume that $y_0 \equiv b \cdot y_1$, for some action b in AC and some term y_1 of sort $\boldsymbol{P}$. It follows that $a \mid b \cdot y_1 \overset{CM6}{=} (a \mid b) \cdot y_1$. It follows from the fact that γ yields an action in AC and Axioms $CF1$, $CF2$, and $A7$ that $HNF((a \mid b) \cdot y_1)$.

(d) Finally, assume that $y_0 \equiv z_1 + z_2$, for some terms z_1 and z_2 in $\boldsymbol{H}(AC)$. It follows that $a \mid (z_1 + z_2) \overset{CM9}{=} a \mid z_1 + a \mid z_2$. By induction, it follows that $HNF(a \mid z_1)$ and $HNF(a \mid z_2)$, which means that $HNF(a \mid (z_1 + z_2))$.

iii) Assume that $x_0 \equiv a \cdot x_1$, for some action a in AC and some term x_1 of sort $\boldsymbol{P}$. As in the previous two cases, the desired result is proven by induction on the structure of y_0. The details are very similar to the proof of the previous case and, therefore, left to the reader.

iv) Finally, assume that $x_0 \equiv z_1 + z_2$, for some terms z_1 and z_2 in $\boldsymbol{H}(AC)$. It follows that $(z_1 + z_2) \mid y_0 \overset{CM8}{=} z_1 \mid y_0 + z_2 \mid y_0$. By induction, it follows that $HNF(z_1 \mid y_0)$ and $HNF(z_2 \mid y_0)$, which means that $HNF((z_1 + z_2) \mid y_0)$. □

Example 5.15. Let A be some set of actions that includes the actions a, b, and c; let γ be the communication function $\{((a, b), c), ((b, a), c)\}$. Consider the following derivation (which can be made in any of the equational theories given in this subsection).

$$
\begin{array}{ll}
 & a \parallel b \\
= & \quad \{ \text{ Axioms } CM1, CM2(2\times) \} \\
 & a \cdot b + b \cdot a + a \mid b \\
= & \quad \{ \text{ Definition } \gamma; \text{ Axiom } CF1 \} \\
 & a \cdot b + b \cdot a + c
\end{array}
$$

This derivation is characteristic for an interleaving equational theory. It shows how a parallel composition can be rewritten into an alternative composition of all its interleavings, including the interleaving in which the two actions communicate. Clearly, the last term of this derivation is in head normal form. Consequently, term $a \parallel b$ has a head normal form (as required by Theorem 5.12).

Another derivation that is characteristic for an interleaving equational theory is the derivation shown in Example 5.8 that uses the expansion theorem of Theorem 5.7.

It is an interesting observation that all the equational theories given in this subsection are interleaving theories. The proof of this result does not depend on the expansion theorem given in Theorem 5.7. The axiomatization of the merge operator in terms of the auxiliary operators left merge and communication merge is the fundamental reason that a theory is an interleaving theory as defined above. Note that this observation does not contradict our informal definition that an equational theory is an interleaving theory if and only if it has some form of expansion theorem. It is possible to prove an expansion theorem for any of the theories of this subsection, although the convenient formulation of such an expansion theorem is only possible when the theory contains the axioms of standard concurrency given in Table 5.6.

As already mentioned in the introduction, most process-algebraic theories in the literature are interleaving theories. In particular, the proof of Theorem 5.12 carries over to any of the ACP-style equational theories presented in [5, 6]. This includes equational theories without communication and theories with recursion or abstraction. To prove that equational theories without communication are interleaving, it is even sufficient to use Property 5.13. Finally, also the equational theories in the CCS framework of [43, 44] and the CSP framework of [35] can be characterized as interleaving theories.

Concluding remark As already mentioned, the motivation to develop the theory $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$ of Table 5.6 is to provide the means to reason about the behavior of concurrent processes. However, so far, the meaning of the algebraic constants, operators, and axioms has only been explained informally. In the next subsection, the semantics of $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$ is formalized using the framework of Section 3.

5.2 A semantics for the equational theory

A model As before, assume that AC is a set of action constants and that $\gamma : AC \times AC \to AC$ is a communication function. The semantics of terms in an equational theory is formalized by defining a so-called *model* of the theory also called an *algebra* for the theory. Since the terms in a single-sorted equational theory such as $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$ are supposed to be interpreted as processes, a model of an ACP-style equational theory is also called a *process algebra*.

In general, a model of a single-sorted equational theory X consists of a *domain* of elements plus a number of functions on that domain, called the *signature* of the model. A model $\mathcal{M}$ with domain $\mathcal{D}$ must satisfy the following properties. First, there must exist an interpretation of the functions in the signature of X in terms of the functions in the signature of the model $\mathcal{M}$ that preserves the arity of functions. Second, any equation that is derivable from the axioms of the equational theory must be *valid* in the model, where validity is defined as follows. Let t be a term in the equational theory X; let σ be a mapping from variables in t to elements from domain $\mathcal{D}$, called a variable substitution. The interpretation of t in the model $\mathcal{M}$ under substitution σ, denoted $[\![t]\!]_\sigma$, is obtained by replacing all functions in t by the corresponding functions in $\mathcal{M}$ and by replacing all variables in t by elements of domain $\mathcal{D}$ according to σ. An equation $t_1 = t_2$ in X is *valid* in model $\mathcal{M}$, denoted $\mathcal{M} \models t_1 = t_2$, if and only if, for *all* substitutions σ for the variables in t_1 and t_2, $[\![t_1]\!]_\sigma =_\mathcal{D} [\![t_2]\!]_\sigma$, where $=_\mathcal{D}$ is the identity on domain $\mathcal{D}$. If $\mathcal{M}$ is a model of an equational theory X, it is also said that X is a *sound axiomatization* of $\mathcal{M}$.

Example 5.16. Assume that $\mathcal{M}$ with domain $\mathcal{D}$ is a model of the theory ACP(AC, γ) of Table 5.1. Recall that the functions in the signature of ACP(AC, γ) consist of the inaction constant δ, the action constants in AC, and the operators listed in Table 5.1. Assume that, for any function f in the signature of ACP(AC, γ), $\bar{f}$ denotes the corresponding function in the signature of $\mathcal{M}$. Consider the equation $a + \delta = a$, where a is an action in AC. It follows from Axiom $A6$ that this equation is derivable from theory ACP(AC, γ). Since $\mathcal{M}$ is a model of this theory, $a + \delta = a$ must be valid in $\mathcal{M}$, which means that the equality $\bar{a} \bar{+} \bar{\delta} =_\mathcal{D} \bar{a}$, where $=_\mathcal{D}$ is the identity on domain $\mathcal{D}$, must hold. In fact, Axiom $A6$ itself is derivable from the theory and must, therefore, be valid in $\mathcal{M}$. That is, for any $d \in \mathcal{D}$, the equality $d \bar{+} \bar{\delta} =_\mathcal{D} d$ must hold.

The basis for a model of the equational theory $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ is a process space, as defined in Definition 3.1. This means that a set of processes, a set of actions, a transition relation, and a termination predicate need to be defined.

Let $\mathcal{C}(AC)$ be the set of *closed* $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ terms. The set $\mathcal{C}(AC)$ forms the basis for the set of processes in the process space. Since the equational theory $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ has no means to express the process that can perform no actions, but can only terminate successfully, a special process $\surd$, pronounced "tick," is introduced. Thus, the set of processes in the abovementioned process space is the set $\mathcal{C}(AC) \cup \{\surd\}$. The set AC is the set of actions. The termination predicate is the singleton $\{\surd\}$. That is, process $\surd$ is the only process that can terminate successfully. The transition relation $\_ \xrightarrow{\_} \_ \subseteq (\mathcal{C}(AC) \cup \{\surd\}) \times AC \times (\mathcal{C}(AC) \cup \{\surd\})$ can now be defined as the smallest relation satisfying the derivation rules in Table 5.17. Note that the transition relation has an implicit parameter, namely the communication function γ. It is not difficult to verify that the transition relation conforms to the informal explanation of the operators given in the previous subsection.

The process space $(\mathcal{C}(AC) \cup \{\surd\}, AC, \longrightarrow, \{\surd\})$ can be turned into a model $\mathcal{M}(AC, \gamma)$ of the equational theory $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ as follows.

Recall that bisimilarity, as defined in Definition 3.8, is an equivalence relation on the set of processes $\mathcal{C}(AC) \cup \{\surd\}$. Thus, it is possible to define equivalence classes of processes modulo bisimilarity in the usual way: For any $p \in \mathcal{C}(AC) \cup \{\surd\}$, the equivalence class of p modulo bisimilarity, denoted $[p]_\sim$, is the set $\{q \in \mathcal{C}(AC) \cup \{\surd\} \mid q \sim p\}$. It follows from Definition 3.8 (Bisimilarity) that the special element $[\surd]_\sim$ only contains the process $\surd$. The *domain* of the model under construction is formed by the set of all the equivalence classes of closed terms modulo bisimilarity.[1] The special element $[\surd]_\sim$ is excluded from the domain of model $\mathcal{M}(AC, \gamma)$ for technical reasons: As mentioned, the equational theory $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ has no means to express the process that can only terminate successfully.

It remains to define the constants and operators of $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ on the domain of the model. The interpretation $\bar{c}$ in model $\mathcal{M}(AC, \gamma)$ of some constant c in the signature of $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ is defined as the equivalence class $[c]_\sim$. Note that this definition fulfills the requirement

[1] In standard process-algebraic terminology, the elements in the domain of a model of some ACP-style equational theory are referred to as processes. However, Definition 3.3 defines a process as some kind of labeled transition system. The domain of model $\mathcal{M}(AC, \gamma)$ consists of equivalence classes of such labeled transition systems. In the literature on concurrency theory, the use of the term "process" for both equivalence classes of labeled transition systems and individual representatives of such equivalence classes is common practice and does not lead to confusion.

$a, b : AC; \boldsymbol{f} : RF;\ p, p', q, q' : \mathcal{C}(AC);$

$$a \xrightarrow{a} \surd \qquad \frac{p \xrightarrow{a} p'}{p \cdot q \xrightarrow{a} p' \cdot q} \qquad \frac{p \xrightarrow{a} \surd}{p \cdot q \xrightarrow{a} q}$$

$$\frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'} \qquad \frac{q \xrightarrow{a} q'}{p + q \xrightarrow{a} q'} \qquad \frac{p \xrightarrow{a} \surd}{p + q \xrightarrow{a} \surd} \qquad \frac{q \xrightarrow{a} \surd}{p + q \xrightarrow{a} \surd}$$

$$\frac{p \xrightarrow{a} p'}{p \parallel q \xrightarrow{a} p' \parallel q} \qquad \frac{q \xrightarrow{a} q'}{p \parallel q \xrightarrow{a} p \parallel q'} \qquad \frac{p \xrightarrow{a} \surd}{p \parallel q \xrightarrow{a} q} \qquad \frac{q \xrightarrow{a} \surd}{p \parallel q \xrightarrow{a} p}$$

$$\frac{p \xrightarrow{a} p',\, q \xrightarrow{b} q',\, \gamma(a, b) \text{ def.}}{p \parallel q \xrightarrow{\gamma(a,b)} p' \parallel q'} \qquad \frac{p \xrightarrow{a} \surd,\, q \xrightarrow{b} \surd,\, \gamma(a, b) \text{ def.}}{p \parallel q \xrightarrow{\gamma(a,b)} \surd}$$

$$\frac{p \xrightarrow{a} p',\, q \xrightarrow{b} \surd,\, \gamma(a, b) \text{ def.}}{p \parallel q \xrightarrow{\gamma(a,b)} p'} \qquad \frac{p \xrightarrow{a} \surd,\, q \xrightarrow{b} q',\, \gamma(a, b) \text{ def.}}{p \parallel q \xrightarrow{\gamma(a,b)} q'}$$

$$\frac{p \xrightarrow{a} p'}{p \mathbin{⫽} q \xrightarrow{a} p' \parallel q} \qquad \frac{p \xrightarrow{a} \surd}{p \mathbin{⫽} q \xrightarrow{a} q}$$

$$\frac{p \xrightarrow{a} p',\, q \xrightarrow{b} q',\, \gamma(a, b) \text{ def.}}{p \mid q \xrightarrow{\gamma(a,b)} p' \parallel q'} \qquad \frac{p \xrightarrow{a} \surd,\, q \xrightarrow{b} \surd,\, \gamma(a, b) \text{ def.}}{p \mid q \xrightarrow{\gamma(a,b)} \surd}$$

$$\frac{p \xrightarrow{a} p',\, q \xrightarrow{b} \surd,\, \gamma(a, b) \text{ def.}}{p \mid q \xrightarrow{\gamma(a,b)} p'} \qquad \frac{p \xrightarrow{a} \surd,\, q \xrightarrow{b} q',\, \gamma(a, b) \text{ def.}}{p \mid q \xrightarrow{\gamma(a,b)} q'}$$

$$\frac{p \xrightarrow{a} p', \boldsymbol{f}(a) \neq \delta}{\rho_{\boldsymbol{f}}(p) \xrightarrow{\boldsymbol{f}(a)} \rho_{\boldsymbol{f}}(p')} \qquad \frac{p \xrightarrow{a} \surd, \boldsymbol{f}(a) \neq \delta}{\rho_{\boldsymbol{f}}(p) \xrightarrow{\boldsymbol{f}(a)} \surd}$$

$$\frac{p \xrightarrow{a} p'}{p^* q \xrightarrow{a} p' \cdot (p^* q)} \qquad \frac{q \xrightarrow{a} q'}{p^* q \xrightarrow{a} q'} \qquad \frac{p \xrightarrow{a} \surd}{p^* q \xrightarrow{a} p^* q} \qquad \frac{q \xrightarrow{a} \surd}{p^* q \xrightarrow{a} \surd}$$

Table 5.17: The transition relation for $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$.

that $\bar{c}$ is a 0-ary function on the domain of $\mathcal{M}(AC, \gamma)$. Also for the operators there is a straightforward way to interpret them in the domain of $\mathcal{M}(AC, \gamma)$, provided that bisimilarity is a *congruence* for all the operators of $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$. That is, the following property must be satisfied. Let $\oplus$ be an arbitrary n-ary operator in the signature of $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$, where n is some positive natural number; let $p_1, \ldots, p_n, q_1, \ldots, q_n$ be closed terms in $\mathcal{C}(AC)$ such that $p_1 \sim q_1, \ldots, p_n \sim q_n$. Then, the congruence property requires that $\oplus(p_1, \ldots, p_n) \sim \oplus(q_1, \ldots, q_n)$.

Property 5.18. (Congruence) *Bisimilarity,* $\sim$, *is a congruence for the operators of* $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$.

Proof. The property follows from the format of the derivation rules in Table 5.17. For details, the reader is referred to [5]. Note that the formulation of the derivation rules in Table 5.17 differs slightly from the formulation of such derivation rules in [5]. However, it is not difficult to verify that the two formulations are equivalent. An extensive treatment of the theory underlying the approach to defining the semantics of (algebraic) languages by means of derivation rules such as those of Table 5.17 can be found in [1]. $\square$

Informally, the congruence property says that equivalence classes of processes can be constructed independently of their representatives. Let $p_1, \ldots, p_n$ be closed terms in $\mathcal{C}(AC)$, where n is some positive natural number; for any n-ary operator $\oplus$ in the signature of $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$, function $\bar{\oplus}$ is defined on equivalence classes of closed terms as follows: $\bar{\oplus}([p_1]_\sim, \ldots, [p_n]_\sim) = [\oplus(p_1, \ldots, p_n)]_\sim$.

At this point, the construction of model $\mathcal{M}(AC, \gamma)$ of the equational theory $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$ is complete. The domain consists of the equivalence classes of closed terms in $\mathcal{C}(AC)$ modulo bisimilarity; the interpretation of any of the constants in the signature of $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$ is the corresponding equivalence class; the interpretation of any operator in the signature of the theory is that same operator lifted to equivalence classes of closed terms. Informally, two closed terms in $\mathcal{C}(AC)$ that are derivably equal in the equational theory $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$ yield the same equivalence class when they are interpreted in the model $\mathcal{M}(AC, \gamma)$, which in turn implies that the corresponding processes are bisimilar. Thus, the equational theory $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$ is a sound axiomatization of bisimilarity.

Theorem 5.19. (Soundness) *For any closed terms $p, q \in \mathcal{C}(AC)$,*

$$(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma) \vdash p = q \;\Rightarrow\; \mathcal{M}(AC, \gamma) \models p = q.$$

Proof. Since bisimilarity is a congruence for the operators of $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$, it suffices to show the validity of each of the axioms. Except for the recursion principle RSP^*, it is not difficult to construct a bisimulation for each axiom. Note that the validity of Axioms $C1$, $C2$, $C3$, $CF1$, and $CF2$ can only be proven if it is assumed that the communication function is commutative and associative.

To prove the validity of RSP^*, assume that p, q, and r are closed terms in $\mathcal{C}(AC)$; let $\mathcal{R}$ be a bisimulation between p and $q \cdot p + r$. It can be shown that the following relation, which uses the transitive closure of $\mathcal{R}$, denoted $\mathcal{R}^+$, is a bisimulation between p and $q * r$, thus proving the validity of RSP^*:

$$\{(p, q * r)\} \cup \mathcal{R}^+ \cup \{(s, q * r) \mid s \in \mathcal{C}(AC) \cup \{\surd\} \wedge s\mathcal{R}^+ p\}$$
$$\cup \{(s, t \cdot (q * r)) \mid s \in \mathcal{C}(AC) \cup \{\surd\} \wedge t \in \mathcal{C}(AC) \wedge s\mathcal{R}^+ t \cdot p\}.$$

In [9], a detailed proof is given (in a setting with silent behavior). □

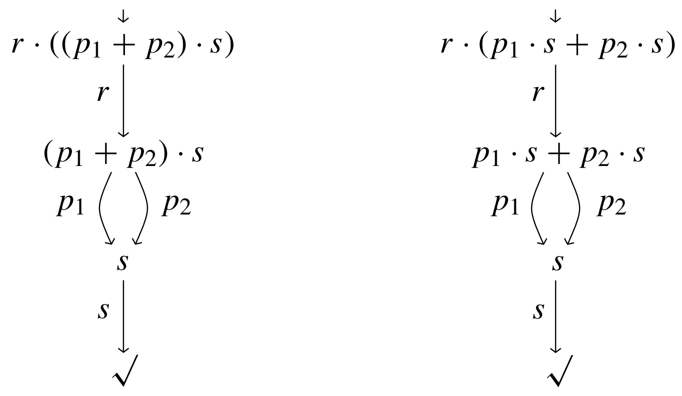


Figure 5.20: Visualizing the semantics of closed $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$ terms.

Example 5.21. Since the basis of the semantics of $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$ is a process space, it is possible to visualize the semantics of closed terms in $\mathcal{C}(AC)$. Consider again Example 5.2. Figure 5.20 depicts the semantics of closed terms $r \cdot ((p_1 + p_2) \cdot s)$ and $r \cdot (p_1 \cdot s + p_2 \cdot s)$. Clearly, these two processes are bisimilar, which conforms to Theorem 5.19 (Soundness) and the conclusion of Example 5.2 that the two closed terms are derivably equal.

Theorem 5.19 is an important result, because it shows that the equational theory (ACP* + RN + RSP^* + SC)(AC, γ) provides a meaningful framework for reasoning about the equality of concurrent processes. It is only possible to prove the equality of two processes if they are bisimilar. This raises the question whether it is always possible to prove the equality of two bisimilar processes in the equational theory (ACP* + RN + RSP^* + SC)(AC, γ). An equational theory is said to be a *complete* axiomatization of some given model if and only if every equality that is valid in the model is also derivable from the axioms of the theory. To the best of our knowledge, it is still an open problem whether the equational theory (ACP* + RN + RSP^* + SC)(AC, γ) is complete for model $\mathcal{M}(AC, \gamma)$. (Note, however, that it follows from the results of [58] that the theory (ACP* + RN + SC)(AC, γ), which is obtained by removing the recursion principle RSP^*, is *not* complete for $\mathcal{M}(AC, \gamma)$.)

Interleaving process algebra In the previous subsection, it has been defined when an equational theory can be characterized as an interleaving theory. As explained in the introduction, a model of an interleaving equational theory is called an interleaving semantics or an interleaving process algebra. However, algebras are well-known mathematical structures (see, for example, [36]) that are not necessarily derived from equational theories as is done in the previous paragraph. Therefore, in this paragraph, we focus on (single-sorted) algebras as mathematical structures in isolation. A single-sorted algebra is simply a structure that has a single domain of elements and a signature of functions of arbitrary arity on that domain. It is not possible to *formally* define when such a single-sorted algebra is a *process* algebra. However, the following informal definition suffices for our purposes. First, the elements of the domain of the algebra should represent processes, where a process is some abstract notion of the behavior of a (concurrent) system. Second, the functions in the signature of the algebra should correspond to meaningful process constants and composition operators on processes. In this paragraph, it is formalized when such a process algebra is an *interleaving* process algebra. In addition, it is shown that the model of the equational theory of Section 5.1 constructed in the previous paragraph is an interleaving process algebra. The formalization of the notion of an interleaving process algebra is based on ACP-style process algebra. However, it can be easily adapted to other process-algebraic frameworks such as CCS and CSP.

Let $\mathcal{A}$ be a process algebra with domain $\mathcal{D}$. The elements of $\mathcal{D}$ are referred to as *processes*. Assume that the signature of $\mathcal{A}$ contains a set of action constants $\mathcal{AC}$. Furthermore, assume that the signature contains at least a (binary) choice operator $\bar{+}$, a sequential composition operator $\bar{\cdot}$, and a parallel-composition operator $\bar{\|}$. In addition, the signature may contain an inaction constant $\bar{\delta}$.

In order to define the notion of an interleaving process algebra, the concept of a process term in head normal form as defined in Definition 5.9 is adapted to the setting of a process algebra. The algebraic framework in this paragraph proves to be simpler than the setting of an equational theory in the previous subsection. The set of processes in a process algebra that have a head normal form can be defined without the use of an auxiliary predicate.

Definition 5.22. (Head normal form) The set of processes over the signature of the algebra $\mathcal{A}$ in *head normal form*, denoted $\mathcal{H}$, is inductively defined as follows. First, $\mathcal{AC} \subseteq \mathcal{H}$. Second, if the signature of $\mathcal{A}$ contains an inaction constant $\bar{\delta}$, then also $\bar{\delta}$ is an element of $\mathcal{H}$. Third, for any action $a \in \mathcal{AC}$ and process $d \in \mathcal{D}$, the process $a \,\bar{\cdot}\, d$ is an element of $\mathcal{H}$. Finally, for any processes $u, v \in \mathcal{H}$, process $u \,\bar{+}\, v$ is an element of $\mathcal{H}$.

A process algebra is said to be interleaving if and only if the set of processes in head normal form is closed under parallel composition.

Definition 5.23. (Interleaving process algebra) The process algebra $\mathcal{A}$ is an *interleaving process algebra* if and only if, for any processes u and v in $\mathcal{H}$, also $u \,\bar{\|}\, v \in \mathcal{H}$.

Let us recall the construction of the model of the equational theory of Section 5.1 in the previous paragraph. Assume that AC is some set of actions; let γ be a communication function. The model $\mathcal{M}(AC, \gamma)$ of theory $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ is based on the set of closed $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ terms, denoted $\mathcal{C}(AC)$. The domain of $\mathcal{M}(AC, \gamma)$ has been defined as the set of equivalence classes of closed terms in $\mathcal{C}(AC)$ modulo bisimilarity. Furthermore, for any action constant $a \in AC$, the interpretation $\bar{a}$ in the model has been defined as $[a]_\sim$; similarly, the interpretation $\bar{\delta}$ of the inaction constant δ has been defined as $[\delta]_\sim$. Finally, for any n-ary operator $\oplus$ in the signature of $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$, where n is some positive natural number, the corresponding function $\bar{\oplus}$ in the signature of $\mathcal{M}(AC, \gamma)$ has been defined as follows: For any closed terms $p_1, \ldots, p_n \in \mathcal{C}(AC)$, $\bar{\oplus}([p_1]_\sim, \ldots, [p_n]_\sim) = [\oplus(p_1, \ldots, p_n)]_\sim$. Clearly, model $\mathcal{M}(AC, \gamma)$ is a process algebra in the general sense defined above.

To prove that the algebra $\mathcal{M}(AC, \gamma)$ is an interleaving process algebra, it is necessary to determine the set of processes in head normal form, as defined by Definition 5.22. The set of action constants $\bar{AC}$ of the algebra $\mathcal{M}(AC, \gamma)$ can be defined in terms of the set of actions AC as follows: $\bar{AC} = \{\bar{a} \mid a \in AC\}$. Furthermore, the signature of the algebra contains, among other functions, the desired choice operator $\bar{+}$ and sequential-composition operator $\bar{\cdot}$. In addition, it contains the inaction constant $\bar{\delta}$. Thus, the set of processes of $\mathcal{M}(AC, \gamma)$ in head normal form, denoted $\mathcal{H}(\bar{AC})$, is defined as in Definition 5.22, where $\bar{AC}$ plays the role of $\mathcal{AC}$. The set of processes in head normal form is parameterized with the set of actions $\bar{AC}$, because $\bar{AC}$ is defined in terms of AC, which is a parameter of model $\mathcal{M}(AC, \gamma)$. Thus, varying the value of parameter AC influences the set of processes in head normal form.

Having defined the set $\mathcal{H}(\bar{AC})$, it is possible to prove that $\mathcal{M}(AC, \gamma)$ is an interleaving process algebra. Note that the signature of $\mathcal{M}(AC, \gamma)$ contains a parallel-composition operator, as required by Definition 5.23 (Interleaving process algebra), namely the operator $\bar{\|}$. Essentially, the desired result follows from the fact that $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ is an interleaving equational theory and an auxiliary property stating that each process of $\mathcal{M}(AC, \gamma)$ in head normal form can be expressed in the signature of the equational theory by a closed term that has a head normal form.

Theorem 5.24. *The process algebra $\mathcal{M}(AC, \gamma)$ is an interleaving process algebra.*

Proof. Let p and q be closed terms in $\mathcal{C}(AC)$ such that $[p]_\sim, [q]_\sim \in \mathcal{H}(\bar{AC})$. It must be shown that $[p]_\sim \mathbin{\bar{\|}} [q]_\sim \in \mathcal{H}(\bar{AC})$. It follows from Property 5.25 given below that there exist closed terms u and v in $\mathcal{C}(AC)$ such that $HNF(u)$, $HNF(v)$, $[u]_\sim = [p]_\sim$, and $[v]_\sim = [q]_\sim$. Hence, it suffices to prove that $[u]_\sim \mathbin{\bar{\|}} [v]_\sim \in \mathcal{H}_\mathcal{M}$. Since $HNF(u)$ and $HNF(v)$, it follows from Theorem 5.12 that $HNF(u \parallel v)$. Definition 5.9 (Head normal form) in the previous subsection implies that there exists a closed term in head normal form $w \in \boldsymbol{H}(AC)$ such that $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma) \vdash u \parallel v = w$. Theorem 5.19 (Soundness) and the definition of validity yield that $[u \parallel v]_\sim = [w]_\sim$. Clearly, it follows from the correspondence between Definitions 5.9 and 5.22 that $[w]_\sim$ is an element of $\mathcal{H}(\bar{AC})$. Since $[w]_\sim = [u \parallel v]_\sim = [u]_\sim \mathbin{\bar{\|}} [v]_\sim$, it is shown that $[u]_\sim \mathbin{\bar{\|}} [v]_\sim$ is an element of $\mathcal{H}(\bar{AC})$, which completes the proof. □

Recall Definition 5.9 from the previous subsection, which formalizes when a closed term defined in the signature of the equational theory $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ has a head normal form. The following property proves the claim made above that any process in the domain of $\mathcal{M}(AC, \gamma)$ that is in head normal form, as defined by Definition 5.22, can be specified in the signature of $(\mathrm{ACP}^* + \mathrm{RN} + RSP^* + \mathrm{SC})(AC, \gamma)$ by means of a closed term that has a head normal form. It also proves the converse, namely that any process in $\mathcal{M}(AC, \gamma)$ that can be specified by means of a closed term that has a head normal form, as defined by Definition 5.9, is indeed a process in head normal form, as defined by Definition 5.22.

Property 5.25. *For any closed term $p \in \mathcal{C}(AC)$,*
$[p]_\sim \in \mathcal{H}(\bar{AC}) \Leftrightarrow (\exists\, q : q \in \mathcal{C}(AC) \wedge HNF(q) : [q]_\sim = [p]_\sim)$.

Proof. Let p be a closed term in $\mathcal{C}(AC)$.

First, assume that $[p]_\sim \in \mathcal{H}(\bar{AC})$. It must be shown that there exists a closed term $q \in \mathcal{C}(AC)$ such that $HNF(q)$ and $[q]_\sim = [p]_\sim$. The proof is by induction on the structure of processes in $\mathcal{H}(\bar{AC})$, as defined by Definition 5.22.

i) Assume that $[p]_\sim = \bar{a}$, for some $a \in AC$. It follows from Definition 5.9 (Head normal form) in the previous subsection that $HNF(a)$. The observation that $\bar{a} = [a]_\sim$ completes the proof in this case.

ii) Assume that $[p]_\sim = \bar{\delta}$. Since $HNF(\delta)$ and $\bar{\delta} = [\delta]_\sim$, term δ satisfies the desired requirements.

iii) Assume that $[p]_\sim = \bar{a} \,\bar{\cdot}\, [q]_\sim$, for some $a \in AC$ and $q \in \mathcal{C}(AC)$. Since $\bar{a} \,\bar{\cdot}\, [q]_\sim = [a \cdot q]_\sim$ and $HNF(a \cdot q)$, the term $a \cdot q$ satisfies the requirements in this case.

iv) Assume that $[p]_\sim = [u]_\sim \,\bar{+}\, [v]_\sim$, for some $u, v \in \mathcal{C}(AC)$ such that $[u]_\sim, [v]_\sim \in \mathcal{H}(\bar{AC})$. (Note that terms u and v are not necessarily elements of $\boldsymbol{H}(AC)$, in which case term $u + v$ would have satisfied the requirements.) By induction, it can be derived that there must exist terms r and s in $\mathcal{C}(AC)$ such that $HNF(r)$, $HNF(s)$, $[r]_\sim = [u]_\sim$, and $[s]_\sim = [v]_\sim$. Hence, it follows from Definition 5.9 (Head normal form) that $HNF(r + s)$. Since, in addition, $[u]_\sim \,\bar{+}\, [v]_\sim = [r]_\sim \,\bar{+}\, [s]_\sim = [r + s]_\sim$, term $r + s$ satisfies the requirements, which completes the first part of the proof.

Second, assume that q is a term in $\mathcal{C}(AC)$ such that $HNF(q)$ and $[q]_\sim = [p]_\sim$. It follows from Definition 5.9 (Head normal form) that there is a term r in $\boldsymbol{H}(AC)$ such that $(\text{ACP}^*{+}\text{RN}{+}RSP^*{+}\text{SC})(AC, \gamma) \vdash q = r$. Theorem 5.19 (Soundness) yields that $[q]_\sim = [r]_\sim$. The correspondence between Definitions 5.9 and 5.22 implies that $[r]_\sim$ is an element of $\mathcal{H}(\bar{AC})$. The observation that $[q]_\sim = [p]_\sim = [r]_\sim$ completes the proof. □

Example 5.26. Consider again Example 5.15. Let A again be the set of actions that includes the actions a, b, and c; let γ be the communication function $\{((a, b), c), ((b, a), c)\}$. Since the terms $a \parallel b$ and $a \cdot b + b \cdot a + c$ are derivably equal in the equational theory $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(A, \gamma)$, it follows from Theorem 5.19 (Soundness) that $[a \parallel b]_\sim = [a \cdot b + b \cdot a + c]_\sim$. That is, the two closed terms specify the same process in the model $\mathcal{M}(A, \gamma)$. (Note that it is also straightforward to obtain this equality by defining a bisimulation between the two terms.) In the algebra $\mathcal{M}(A, \gamma)$, the same process can also be written as $\bar{a} \,\bar{\parallel}\, \bar{b}$ and $\bar{a} \,\bar{\cdot}\, \bar{b} \,\bar{+}\, \bar{b} \,\bar{\cdot}\, \bar{a} \,\bar{+}\, \bar{c}$. Clearly, the latter expression is in head normal form, which means that process $\bar{a} \,\bar{\parallel}\, \bar{b}$ is in head normal form, as required by Theorem 5.24.

An interesting observation is that the proof of Theorem 5.24 does not use any specific characteristics of the equational theory $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$ or its model $\mathcal{M}(AC, \gamma)$ other than the fact that the equational theory is an interleaving theory. Thus, the proof carries over to *any* model of *any* interleaving equational theory. This conforms to our informal definition of an interleaving semantics given in Section 1.

Another interesting observation is that Property 5.25 can be strengthened in case the equational theory under consideration is a *complete* axiomatization of an algebra. In that case, any closed term over the signature of the equational theory that specifies a process in head normal form in the algebra must have a head normal form itself.

Concluding remark Both the equational theory $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$ of Section 5.1 and its model $\mathcal{M}(AC, \gamma)$ given in this subsection are parameterized by a set of actions and a communication function. These parameters provide the flexibility to define, for example, a total-order semantics and a step semantics for concurrent systems, as shown in Sections 5.3 and 5.4, respectively. Since Theorem 5.24 proven above is independent of the choice for the parameters AC and γ, both the resulting process algebras are *interleaving* algebras. Thus, the semantics of Section 5.4 is an *interleaving, partial-order* process algebra.

5.3 A total-order semantics

In this subsection, the equational theory of Section 5.1 and its model of Section 5.2 are turned into a total-order framework for reasoning about the behavior of concurrent systems. The approach is illustrated by means of a few examples. The essence of a total-order semantics for concurrent systems is that no two actions can occur simultaneously. Thus, in this subsection, it is assumed that actions are atomic.

Definition 5.27. (Total-order semantics) Let A be some set of atomic actions. Let $\gamma : A \times A \to A$ be a communication function on atomic actions. The model $\mathcal{M}(A, \gamma)$ defines a (branching-time, interleaving) total-order semantics for $(\mathrm{ACP}^* + \mathrm{RN} + \mathit{RSP}^* + \mathrm{SC})(AC, \gamma)$.

As explained before, a fundamental property of a total-order semantics is that it is impossible to distinguish concurrency and non-determinism. In the setting of the equational theory $(\mathrm{ACP}^*{+}\mathrm{RN}{+}\mathit{RSP}^*{+}\mathrm{SC})(AC, \gamma)$, this means that a (closed) term containing a parallel composition of two component terms must have the same semantics as a term specifying a non-deterministic choice between all possible totally-ordered interleavings of these two component terms.

Example 5.28. To illustrate that the semantics of Definition 5.27 is indeed a total-order semantics, assume that A is some set of atomic actions including the actions p_1, p_2, r, and s. Assume that γ is the communication function that is undefined for all the elements in its domain. That is, there is no communication. Consider the following derivation. Since sequential composition is associative (Axiom $A5$), redundant brackets are omitted.

$$
\begin{array}{ll}
 & r \cdot (p_1 \parallel p_2) \cdot s \\
= & \quad \{ \text{ Axioms } CM1, CM2(2\times) \} \\
 & r \cdot (p_1 \cdot p_2 + p_2 \cdot p_1 + p_1 \mid p_2) \cdot s \\
= & \quad \{ \text{ Definition } \gamma\text{; Axioms } CF2, A6 \} \\
 & r \cdot (p_1 \cdot p_2 + p_2 \cdot p_1) \cdot s
\end{array}
$$

This derivation shows that a term specifying the parallel composition of actions p_1 and p_2 is derivably equal to a term specifying a non-deterministic choice between two totally-ordered alternatives. Theorem 5.19 (Soundness) implies that the two closed terms must correspond to the same process in the semantics of Definition 5.27. Clearly, the semantics of both terms can be visualized by the process depicted in Figure 3.4(d), where dots correspond to closed terms.

In the previous example, communication plays no role. The following example shows a typical use of the communication function in ACP-style process algebra. Communication is used to enforce synchronization between parallel components.

Example 5.29. Let A be some set of atomic actions that includes the actions r_1, r_2, s_2, c_2, and s_3. Let γ be the function $\{((r_2, s_2), c_2), ((s_2, r_2), c_2)\}$. Informally, a process performing an action r_i with $i \in \{1, 2\}$ receives a message of its environment over some port i. If a process performs an action s_i with $i \in \{2, 3\}$, it sends a message to its environment over port i. An action c_2 represents a communication over port 2, which is a synchronization of a send action and a receive action.

A process $B_{1,2}$ that repeatedly receives a message over port 1 and then forwards this message over port 2, thus corresponding to a one-place buffer for messages, can be specified as follows: $B_{1,2} = (r_1 \cdot s_2) * \delta$. It follows from Axioms $BKS1$ and $A6$ that this specification represents a non-terminating iteration. Similarly, process $B_{2,3} = (r_2 \cdot s_3) * \delta$ represents a one-place buffer for messages that receives messages over port 2 and sends messages over port 3.

In a number of steps, it can be shown that the composition of the above two one-place buffers yields a two-place buffer that receives messages from its environment over port 1 and sends messages to its environment over port 3. The composition is a parallel composition with the restriction that the two buffers must communicate over port 2. The following auxiliary derivation shows how a non-terminating iteration can be unfolded one cycle.

$$
\begin{aligned}
& B_{1,2} \\
= \quad & \{ \text{ Substitution } \} \\
& (r_1 \cdot s_2)^* \delta \\
= \quad & \{ \text{ Axiom } BKS1\text{; substitution } \} \\
& (r_1 \cdot s_2) \cdot B_{1,2} + \delta \\
= \quad & \{ \text{ Axioms } A5,\ A6 \ \} \\
& r_1 \cdot s_2 \cdot B_{1,2}
\end{aligned}
$$

The next derivation shows how to calculate the first action of the abovementioned parallel composition of the two one-place buffers. To enforce communication over port 2, isolated occurrences of send and receive actions over port 2 must be encapsulated. Therefore, let $H = \{r_2, s_2\}$.

$$
\begin{aligned}
& \partial_H(B_{1,2} \parallel B_{2,3}) \\
= \quad & \{ \text{ Axiom } CM1 \ \} \\
& \partial_H(B_{1,2} \mathbin{⌊\!⌊} B_{2,3} + B_{2,3} \mathbin{⌊\!⌊} B_{1,2} + B_{1,2} \mid B_{2,3}) \\
= \quad & \{ \text{ Derivation above; Axiom } CM3(2\times)\text{; Axiom } CM7 \ \} \\
& \partial_H(r_1 \cdot (s_2 \cdot B_{1,2} \parallel B_{2,3}) + r_2 \cdot (s_3 \cdot B_{2,3} \parallel B_{1,2}) + (r_1 \mid r_2) \cdot (s_2 \cdot B_{1,2} \parallel s_3 \cdot B_{2,3})) \\
= \quad & \{ \text{ Axiom } CF2\text{; Axioms } D3(2\times),\ D4(3\times),\ D1(2\times),\ D2 \ \} \\
& r_1 \cdot \partial_H(s_2 \cdot B_{1,2} \parallel B_{2,3}) + \delta \cdot \partial_H(B_{1,2} \parallel s_3 \cdot B_{2,3}) + \delta \cdot \partial_H(s_2 \cdot B_{1,2} \parallel s_3 \cdot B_{2,3}) \\
= \quad & \{ \text{ Axioms } A6,\ A7(\text{both } 2\times) \ \} \\
& r_1 \cdot \partial_H(s_2 \cdot B_{1,2} \parallel B_{2,3})
\end{aligned}
$$

Similar derivations yield the following results. The receipt of a message over port 1 is followed by a communication action over port 2, after which, in any order, a new message may be received over port 1 and the first message may be send to the environment over port 3.

$$
\begin{aligned}
\partial_H(s_2 \cdot B_{1,2} \parallel B_{2,3}) &= c_2 \cdot \partial_H(B_{1,2} \parallel s_3 \cdot B_{2,3}) \\
&= c_2 \cdot (r_1 \cdot \partial_H(s_2 \cdot B_{1,2} \parallel s_3 \cdot B_{2,3}) + s_3 \cdot \partial_H(B_{1,2} \parallel B_{2,3})) \\
&= c_2 \cdot (r_1 \cdot s_3 \cdot \partial_H(s_2 \cdot B_{1,2} \parallel B_{2,3}) + s_3 \cdot r_1 \cdot \partial_H(s_2 \cdot B_{1,2} \parallel B_{2,3})) \\
&= c_2 \cdot (r_1 \cdot s_3 + s_3 \cdot r_1) \cdot \partial_H(s_2 \cdot B_{1,2} \parallel B_{2,3}).
\end{aligned}
$$

Summarizing the results obtained so far yields the following two equations.

$$
\begin{aligned}
\partial_H(B_{1,2} \parallel B_{2,3}) &= r_1 \cdot \partial_H(s_2 \cdot B_{1,2} \parallel B_{2,3}) \text{ and} \\
\partial_H(s_2 \cdot B_{1,2} \parallel B_{2,3}) &= c_2 \cdot (r_1 \cdot s_3 + s_3 \cdot r_1) \cdot \partial_H(s_2 \cdot B_{1,2} \parallel B_{2,3}).
\end{aligned}
$$

The second equation defines a non-terminating iteration, as can be shown by applying Axioms $A6$ and RSP^*:

$$\partial_H(s_2 \cdot B_{1,2} \parallel B_{2,3}) = (c_2 \cdot (r_1 \cdot s_3 + s_3 \cdot r_1))^* \delta.$$

Substituting this result in the above equation for $\partial_H(B_{1,2} \parallel B_{2,3})$ yields that

$$\partial_H(B_{1,2} \parallel B_{2,3}) = r_1 \cdot ((c_2 \cdot (r_1 \cdot s_3 + s_3 \cdot r_1))^* \delta).$$

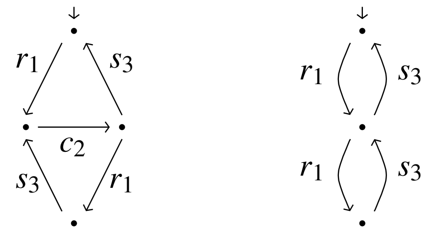
Finally, as in the previous example, it can be shown that the choice in the last result corresponds to a parallel composition.

$$
\begin{aligned}
& r_1 \cdot s_3 + s_3 \cdot r_1 \\
= \quad & \{ \text{ Axiom } A6 \ \} \\
& r_1 \cdot s_3 + s_3 \cdot r_1 + \delta \\
= \quad & \{ \text{ Axiom } CF2\text{; definition } \gamma \ \} \\
& r_1 \cdot s_3 + s_3 \cdot r_1 + r_1 \mid s_3 \\
= \quad & \{ \text{ Axioms } CM2(2\times),\ CM1 \ \} \\
& r_1 \parallel s_3
\end{aligned}
$$

Thus, the final result for the composition of the two one-place buffers is the following.

$$\partial_H(B_{1,2} \parallel B_{2,3}) = r_1 \cdot ((c_2 \cdot (r_1 \parallel s_3))^* \delta).$$

It is not difficult to see that the right-hand side of the above equation specifies the behavior of a two-place buffer. Its semantics is depicted in Figure 5.30(a), where closed terms are represented by dots. Figure 5.30(b) shows the process that results from hiding the internal communication action c_2. Clearly, this process exhibits the behavior that is intuitively expected from a two-place buffer. In the current algebraic framework, it is not possible to hide actions. The extension of ACP-style process algebra with an abstraction operator is described in [6, 14]. It goes beyond the scope of this chapter to treat abstraction in detail.



(a) (b)

Figure 5.30: A two-place buffer in a total-order semantics.

5.4 A step semantics

As in Sections 3 and 4, the basis of a step semantics is to assume that actions are bags of atomic actions. The occurrence of a bag of multiple atomic actions represents the simultaneous execution of its elements. The communication function can be used to specify which actions can be executed in parallel.

Definition 5.31. (Step semantics) Let A be some set of atomic actions. Let $\gamma : \mathcal{B}(A) \times \mathcal{B}(A) \rightarrow \mathcal{B}(A)$ be some communication function. The model $\mathcal{M}(\mathcal{B}(A), \gamma)$ defines a (branching-time, interleaving) step semantics for $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(AC, \gamma)$, *provided* that the communication function γ allows that actions may be executed concurrently.

Example 5.32. The most straightforward definition for the communication function γ in the semantics of Definition 5.31 is to define γ as summation of bags: $\gamma = \uplus$. Assume that A includes the atomic actions p_1, p_2, r, and s. Consider the processes specified by $[r]\cdot([p_1]\cdot[p_2]+[p_2]\cdot[p_1])\cdot[s]$ and $[r]\cdot([p_1]\,\|\,[p_2])\cdot[s]$. In $(\text{ACP}^* + \text{RN} + RSP^* + \text{SC})(\mathcal{B}(A), \uplus)$, it is not possible to prove that these two terms are equal. The corresponding processes in the semantics of Definition 5.31 can be visualized as in Figure 3.6. As explained before, these two processes are not bisimilar.

A disadvantage of choosing bag summation for the communication function is that it is no longer possible to do derivations such as the ones in Example 5.29. Choosing bag summation for the communication function means that any action may occur concurrently with any other action. Bag summation cannot be used to enforce communication between actions or to enforce a total ordering on the execution of certain actions that are in conflict with each other. The latter means that actions can neither communicate or happen concurrently. One solution to combine concurrency with communication and conflicts is to carefully define an appropriate communication function.

Example 5.33. As in Example 5.29, let A be some set of atomic actions including r_1, r_2, s_2, c_2, and s_3. Recall that the indices of these actions correspond to port numbers.

Informally, the communication function can be specified as follows. The only communication is the simultaneous execution of send action s_2 and receive action r_2, which results in the communication action

c_2. Any number of atomic actions that cannot communicate can occur in parallel as long as no two of these actions use the same port. Actions using the same port are in conflict and must, therefore, be totally ordered.

To formalize the definition of the communication function, the auxiliary predicates *conflict* $\subseteq \mathcal{B}(A)$ and *communication* $\subseteq \mathcal{B}(A)$ are defined as follows. For any $\alpha \in \mathcal{B}(A)$,

$$\begin{array}{ll} \mathit{conflict}(\alpha) & \Leftrightarrow (\exists\, a : a \in A : [a^2] \leq \alpha) \vee [r_2, c_2] \leq \alpha \vee [s_2, c_2] \leq \alpha \text{ and} \\ \mathit{communication}(\alpha) & \Leftrightarrow [s_2, r_2] \leq \alpha. \end{array}$$

Using these two predicates, the communication function γ is defined as follows. Note that a bag of atomic actions containing no conflicts can contain at most one pair of atomic actions s_2 and r_2. For any $\alpha, \beta \in \mathcal{B}(A)$,

$$\begin{array}{ll} \neg \mathit{conflict}(\alpha \uplus \beta) \wedge \mathit{communication}(\alpha \uplus \beta) & \Rightarrow \gamma(\alpha, \beta) = (\alpha \uplus \beta) - [s_2, r_2] \uplus [c_2], \\ \neg \mathit{conflict}(\alpha \uplus \beta) \wedge \neg \mathit{communication}(\alpha \uplus \beta) & \Rightarrow \gamma(\alpha, \beta) = \alpha \uplus \beta, \text{ and} \\ \mathit{conflict}(\alpha \uplus \beta) & \Rightarrow \gamma(\alpha, \beta) \text{ is undefined.} \end{array}$$

It is not difficult to verify that this communication function is commutative and associative.

In the framework of this example, the two one-place buffers of Example 5.29 can be specified as follows: $B_{1,2} = ([r_1] \cdot [s_2]) * \delta$ and $B_{2,3} = ([r_2] \cdot [s_3]) * \delta$. Also in the current setting, it is possible to show that the parallel composition of the two buffers while enforcing communication corresponds to a two-place buffer. To enforce communication, actions that contain isolated send or receive actions over port 2 must be encapsulated. Therefore, let $H \subseteq \mathcal{B}(A)$ be defined as the set $\{\alpha \in \mathcal{B}(A) \mid r_2 \in \alpha \vee s_2 \in \alpha\}$. Calculations very similar to the ones in Example 5.29 lead to the following results.

$$\begin{array}{lll} \partial_H(B_{1,2} \parallel B_{2,3}) & = & [r_1] \cdot \partial_H([s_2] \cdot B_{1,2} \parallel B_{2,3}) \\ \partial_H([s_2] \cdot B_{1,2} \parallel B_{2,3}) & = & [c_2] \cdot \partial_H(B_{1,2} \parallel [s_3] \cdot B_{2,3}) \\ & = & [c_2] \cdot \ (\ [r_1] \cdot \partial_H([s_2] \cdot B_{1,2} \parallel [s_3] \cdot B_{2,3}) \\ & & \qquad +[s_3] \cdot \partial_H(B_{1,2} \parallel B_{2,3}) \\ & & \qquad +[r_1, s_3] \cdot \partial_H([s_2] \cdot B_{1,2} \parallel B_{2,3}) \\ & & \qquad) \\ & = & [c_2] \cdot ([r_1] \cdot [s_3] + [s_3] \cdot [r_1] + [r_1, s_3]) \cdot \partial_H([s_2] \cdot B_{1,2} \parallel B_{2,3}) \end{array}$$

The main difference between the above results and the corresponding results in Example 5.29 is that, in the current setting, the actions r_1 and s_3 can occur concurrently, whereas this is not possible in Example 5.29. Example 5.29 shows how a parallel composition is rewritten into a non-deterministic choice of *totally*-ordered interleavings. In the current example, a parallel composition is rewritten into a non-deterministic choice of *partially*-ordered interleavings, namely the interleavings of all the *steps* that the parallel composition can perform. These observations conform to our remark made earlier that both the total-order framework of the previous subsection and the partial-order framework of this subsection are interleaving frameworks.

Note that one of the above results is a recursive equation. Using *RSP**, the axioms for the three merge operators, and the definition of the communication function, the following result can be derived. Again, the calculations are very similar to the ones in Example 5.29.

$$\partial_H(B_{1,2} \parallel B_{2,3}) = [r_1] \cdot (([c_2] \cdot ([r_1] \parallel [s_3])) * \delta).$$

Figure 5.34(a) visualizes the semantics of the two-place buffer. Figure 5.34(b) shows the process that results from hiding the internal communication action $[c_2]$.

Observe that the approach of Example 5.33 can be adapted to obtain a total-order setting by specifying that any two atomic actions are in conflict unless they communicate. The result is that any process can only execute steps consisting of a single atomic action.

The attentive reader might have noticed that in the calculations of Example 5.33 no conflicts between atomic actions occur, as specified by the auxiliary predicate *conflict*. Thus, for the above example, the

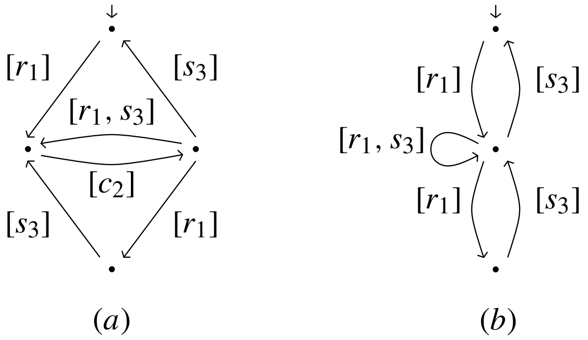


Figure 5.34: A two-place buffer in a step semantics.

communication function can be simplified. Nevertheless, the definition of conflicts can be meaningful, as the following example shows.

Example 5.35. Assume that the set of atomic actions A, the predicates *conflict*, *communication* $\subseteq \mathcal{B}(A)$, the communication function $\gamma : \mathcal{B}(A) \times \mathcal{B}(A) \to \mathcal{B}(A)$, and the set of actions $H \subseteq \mathcal{B}(A)$ are defined as in Example 5.33.

Consider the following two processes: $S = ([r_1] \cdot [s_2] \cdot [r_2])^* \delta$ and $C = (([r_1] \parallel [r_2]) \cdot ([s_2] \mid [s_3]))^* \delta$. Process C, a calculator, models an iterative process of which a cycle starts by receiving two messages over ports 1 and 2. Ports could, for example, be metal wires and messages could be electrical signals. After C has received both signals, it simultaneously sends a result signal over port 3 and a confirmation signal over port 2 to indicate that both input signals have been read. Process S is a simple synchronized buffer that forwards a signal from port 1 to port 2; before it reads its next signal over port 1, it waits for a confirmation over port 2.

Intuitively, the parallel composition of processes S and C, enforcing communication over port 2, should yield an iterative process that in each cycle reads two signals over port 1 and sends a result signal over port 3. Recall that the definition of the predicate *conflict* is such that no two atomic actions using the same port can occur simultaneously. The results below confirm that the composed process starts with a single r_1, which is either performed by S or by C. It does not have the option to read its two inputs simultaneously. After the derivation of the first action, the calculations are very similar to the ones in the previous examples.

$$\begin{aligned}
\partial_H(S \parallel C) &= [r_1] \cdot \partial_H([s_2] \cdot [r_2] \cdot S \parallel C) + [r_1] \cdot \partial_H(S \parallel [r_2] \cdot ([s_2] \mid [s_3]) \cdot C) \\
&= ([r_1] \cdot ([r_1] \cdot [c_2] + [c_2] \cdot [r_1] + [r_1, c_2]) + [r_1] \cdot [r_1] \cdot [c_2]) \cdot \partial_H([r_2] \cdot S \parallel ([s_2] \mid [s_3]) \cdot C) \\
&= ([r_1] \cdot ([r_1] \parallel [c_2]) + [r_1] \cdot [r_1] \cdot [c_2]) \cdot [c_2, s_3] \cdot \partial_H(S \parallel C) \\
&= ([r_1] \cdot [c_2] \parallel [r_1]) \cdot [c_2, s_3] \cdot \partial_H(S \parallel C)
\end{aligned}$$

Thus, it follows from *RSP** that

$$\partial_H(S \parallel C) = (([r_1] \cdot [c_2] \parallel [r_1]) \cdot [c_2, s_3])^* \delta.$$

Ignoring communication actions c_2, the result is indeed a process that, in each cycle, reads two signals over port 1 and sends a result signal over port 3.

The approach to combining communication and concurrency, including conflicts, in the communication function can be generalized to larger examples. However, the resulting communication function may become complex and intuitively difficult to understand. Another solution to combine concurrency with communication in the current framework is to allow arbitrary concurrency between actions, as in Example 5.32, and to use renaming operators to make conflicts and communications explicit. This approach is very similar to the approach to combining concurrency and communication in ACP-style process algebra taken in [4].

Example 5.36. Let A be the set of atomic actions defined in Example 5.33. Assume that the communication function γ on bags of atomic actions equals bag summation $\uplus$. Using the auxiliary predicates *conflict* and *communication* of Example 5.33, the renaming functions $\boldsymbol{cf}, \boldsymbol{cm} : (\mathcal{B}(A) \cup \{\delta\}) \to (\mathcal{B}(A) \cup \{\delta\})$ are defined as follows. For any $\alpha \in \mathcal{B}(A)$,

$$\boldsymbol{cf}(\delta) = \delta,\ \mathit{conflict}(\alpha) \Rightarrow \boldsymbol{cf}(\alpha) = \delta,\ \text{and}\ \neg\mathit{conflict}(\alpha) \Rightarrow \boldsymbol{cf}(\alpha) = \alpha$$

and

$$\boldsymbol{cm}(\delta) = \delta,\ \mathit{communication}(\alpha) \Rightarrow \boldsymbol{cm}(\alpha) = \alpha - [s_2, r_2] \uplus [c_2],\ \text{and}$$
$$\neg\mathit{communication}(\alpha) \Rightarrow \boldsymbol{cm}(\alpha) = \alpha.$$

Given these definitions, it is not difficult to obtain results similar to Examples 5.33 and 5.35. The basic idea is to start with the parallel composition of component processes, resolve conflicts according to the renaming function $\boldsymbol{cf}$, and enforce communication by renaming actions according to renaming function $\boldsymbol{cm}$ and encapsulating actions with isolated atomic actions that should have participated in a communication. Let $H \subseteq \mathcal{B}(A)$ be the set $\{\alpha \in \mathcal{B}(A) \mid r_2 \in \alpha \vee s_2 \in \alpha\}$, as defined in Example 5.33.

$$\partial_H(\rho_{\boldsymbol{cm}}(\rho_{\boldsymbol{cf}}(B_{1,2} \parallel B_{2,3}))) = [r_1] \cdot (([c_2] \cdot ([r_1] \parallel [s_3]))^* \delta) \text{ and}$$
$$\partial_H(\rho_{\boldsymbol{cm}}(\rho_{\boldsymbol{cf}}(S \parallel C))) = (([r_1] \cdot [c_2] \parallel [r_1]) \cdot [c_2, s_3])^* \delta,$$

where $B_{1,2}$ and $B_{2,3}$ are the two one-place buffers defined in Example 5.33 and S and C are the synchronized buffer and the calculator of Example 5.35.

Example 5.36 generalizes to larger examples in a rather straightforward way. In general, the definition of a renaming function removing all conflicts from a process term is not very complicated. The most straightforward way to obtain a renaming function handling an arbitrary number of communications is to define such a function as the composition of a number of auxiliary renaming functions that each rename a single pair of communicating atomic actions. Section 7 explains more about renamings and communication functions in a setting with steps.

Finally, observe that, unlike the approach of Example 5.33, the approach of Example 5.36 cannot be adapted to obtain a total-order setting. In the approach of Example 5.33, conflicts and communications are defined in the communication function. Thus, the communication function can be used to enforce a total ordering of (steps of) single atomic actions. In the approach of Example 5.36, the communication function is fixed to bag summation. A total ordering of atomic actions can only be enforced *explicitly* by means of renaming.

5.5 Concluding remarks

Summarizing the results of this section, it has been shown how to develop an algebraic framework for equational reasoning both in a total-order setting and in a partial-order setting. In addition, the notion of a (non-)interleaving algebraic theory has been formalized. It has been shown that standard ACP-style process algebra always leads to an interleaving framework. An interesting consequence is that the algebraic framework of Section 5.4 can be characterized as an *interleaving partial-order* theory. This observation conforms to our claim made in the introduction that interleaving versus non-interleaving and total-order versus partial-order are complementary characterizations of process-algebraic frameworks.

The basis of the step semantics for concurrent systems presented in Section 5.4 is the communication function of ACP. It can be used to specify which atomic actions are causally independent and, thus, may occur concurrently. Examples 5.33 and 5.35 show how to combine communication and concurrency in the definition of the communication function. Example 5.36 illustrates another approach that makes conflicts and communications explicit by means of renaming operators. The latter approach generalizes more easily to larger examples than the former. In the next section, a third approach to combining concurrency and communication is studied, which is inspired by the theory of P/T nets developed in Section 4. The basic idea is to use the communication function of ACP solely for the purpose of concurrency, as in Examples 5.32 and 5.36, and to introduce a new operator that can be used – to some extent – to resolve conflicts and to enforce (asynchronous) communication between parallel components. This new operator is inspired by the notion of transition firing in the framework of P/T nets.

6 The Causal State Operator

In the P/T-net framework of Section 4, the behavior of a P/T net is determined by its marking. The order in which tokens in the marking are consumed and produced determines the order in which actions are executed. Thus, in the P/T-net framework, the mechanism of consuming and producing tokens is the basic mechanism for introducing causal relationships between actions. In this section, the equational theory of the previous section is extended with an algebraic operator, called the *causal state operator*, that is inspired by the idea of consuming and producing tokens. The causal state operator plays an important role in adopting the causality mechanism of Petri nets into process algebra. In ACP-style process algebra, the standard means to enforce causal relationships between actions are sequential composition and communication. The causal state operator is sufficiently powerful to replace to some extent this standard causality mechanism, in particular, the standard communication mechanism. As a result, the communication function can be used to develop a framework with a step semantics, as explained in the previous section, thus combining the strength of communication and concurrency. An interesting application of the equational theory developed in this section is that it can be used to define an algebraic semantics for labeled P/T nets that conforms to the step semantics for labeled P/T nets defined in Section 4.

6.1 The equational theory

The causal state operator is a specialization of the so-called *state operator*. The state operator is an operator which has a memory to explicitly describe the state of a process. For a detailed treatment of the state operator, the reader is referred to [3, 5, 6]. A variant of the *causal* state operator first appeared in [4].

Table 6.1 presents the Algebra of Communicating Processes with iteration and causal state operator, renaming, the Recursive Specification Principle for the binary Kleene star, and standard concurrency, abbreviated ACP$^*_\lambda$+RN+RSP^*+SC. It is parameterized by a set of actions $AC(C, A)$, which is itself parameterized, and a communication function $\gamma : AC(C, A) \times AC(C, A) \to AC(C, A)$. A detailed explanation is given below.

(ACP$^*_\lambda$ + RN + RSP^* + SC)$(AC(C, A), \gamma)$	
(ACP* + RN + RSP^* + SC)$(AC(C, A), \gamma)$	
$\lambda^I_m : \boldsymbol{P} \to \boldsymbol{P}$;	
$x, y : \boldsymbol{P}$;	
$\lambda^I_m(\delta) = \delta$	*CSO*1
$\boldsymbol{ca} \restriction I \leq m \ \Rightarrow\ \lambda^I_m(a) = a$	*CSO*2
$\boldsymbol{ca} \restriction I \not\leq m \ \Rightarrow\ \lambda^I_m(a) = \delta$	*CSO*3
$\lambda^I_m(a \cdot x) = \lambda^I_m(a) \cdot \lambda^I_{m - \boldsymbol{ca}\restriction I \uplus \boldsymbol{pa}\restriction I}(x)$	*CSO*4
$\lambda^I_m(x + y) = \lambda^I_m(x) + \lambda^I_m(y)$	*CSO*5

Table 6.1: The equational theory (ACP$^*_\lambda$ + RN + RSP^* + SC)$(AC(C, A), \gamma)$.

In order to reason about places, tokens, token consumption, and token production in an equational way, it is necessary to introduce these notions in the equational theory. In a P/T net, the flow relation between places and transitions defines the token consumption and token production of a particular transition. The most straightforward way to introduce the notions of consumption and production in a process-algebraic setting is to include the information in the actions. The two parameters of the set of actions $AC(C, A)$ correspond to a set of so-called *causes* and a set of atomic actions, respectively. Causes are the algebraic equivalent of places and tokens in the P/T-net framework. In the remainder of this section, an action is assumed to be a triple in $\mathcal{B}(C) \times \mathcal{B}(A) \times \mathcal{B}(C)$. That is, $AC(C, A) = \mathcal{B}(C) \times \mathcal{B}(A) \times \mathcal{B}(C)$. The second element of such a triple denotes the actual *step* of the action; the first element represents the *consumption* of causes of the step and the third element its *production*, similar to the notions of token consumption and token production in the P/T-net framework. Three auxiliary functions on actions in $AC(C, A)$, $\boldsymbol{c}, \boldsymbol{p} : AC(C, A) \rightarrow \mathcal{B}(C)$ and $\boldsymbol{s} : AC(C, A) \rightarrow \mathcal{B}(A)$, are defined. For any $a = (c, \alpha, p)$ in $AC(C, A)$, $\boldsymbol{c}a = c$, $\boldsymbol{p}a = p$, and $\boldsymbol{s}a = \alpha$. Note that the exact structure of actions is not really important as long as these auxiliary functions can be defined.

The causal state operator is, in fact, not a single operator. Theory $(\mathrm{ACP}^*_\lambda{+}\mathrm{RN}{+}RSP^*{+}\mathrm{SC})(AC(C, A), \gamma)$ contains an entire class of causal state operators, namely one operator λ^I_m for each $I \subseteq C$ and $m \in \mathcal{B}(I)$. Set I can be seen as a set specifying *internal* causes. Causes not in I are referred to as *external* causes. Note that m is defined to be a bag of *internal* causes. In the term $\lambda^I_m(x)$, where x is a process term of sort $\boldsymbol{P}$, bag m represents the current state of process x, similar to a marking in P/T-net theory. For this reason, bag m is also referred to as the marking of x. The separation between internal and external causes provides the flexibility to distinguish between communication within a process and communication between the process and its environment. In standard P/T-net theory as introduced in Section 4, places and tokens are not divided into internal and external places and tokens. However, when a notion of modularity is introduced in P/T-net theory, as in [8, Chapter 3], the separation between internal and external places and tokens becomes meaningful in the P/T-net framework. Section 8 treats this topic in more detail.

The axioms for causal state operators in Table 6.1 are inspired by the notion of transition firing introduced in Section 4. Constant a ranges over $AC(C, A)$. Axioms $CSO2$, $CSO3$, and $CSO4$ are the most interesting ones. The other two should be clear without further explanation. Axiom $CSO2$ states that an action may occur provided that its consumption of *internal* causes is available in the marking. Axiom $CSO3$ says that if not enough causes are available, an action cannot be executed and results in a deadlock. Axiom $CSO4$ states that the result of executing an action is that the consumption is removed from the marking, whereas the (internal) production is added to the marking. If the marking does not contain enough internal causes to execute the action, then the combination of Axioms $CSO3$ and $A7$ guarantees that the result is a deadlock. The axiomatization assumes that the environment of a process is responsible for consuming and producing external causes.

The intuition behind verifications by means of the causal state operator is the following. Given a number of components, as before, the parallel composition of these components is the starting point. The communication function is defined in such a way that it allows arbitrary concurrency. Thus, the parallel composition defines the largest behavior that is possible when combining the basic components. A causal state operator is used to restrict the composition to those behaviors that are allowed in the initial state of the composition, as defined by the subscript of the state operator. Thus, the causal state operator enforces causal orderings between actions.

As in the previous section, the expansion theorem of Theorem 5.7 can be used to simplify calculations with parallel compositions. It carries over to the theory $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma)$. The proof is identical. The next theorem is a specific instance of the expansion theorem which is often useful in derivations following the above approach.

Theorem 6.2. (Expansion of iterations) *Let $a_0, \ldots, a_n$ be actions in $AC(C, A)$, where n is a positive natural number. Let I be the set $\{0, \ldots, n\}$.*

$$(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma) \vdash$$
$$(\|i : i \in I : a_i * \delta) = (+J : \emptyset \subset J \subseteq I : (|j : j \in J : a_j) \cdot (\|i : i \in I : a_i * \delta)).$$

Proof. It is a straightforward consequence of Theorem 5.7 and the axioms of $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma)$. □

Example 6.3. Let C be a set of causes including the causes $1, 2, 3, 4, 5$, and 6; let A be a set of atomic actions containing r, p_1, p_2, and s. The following are examples of actions in $AC(C, A)$: $([1], [r], [2, 3])$, $([2], [p_1], [4])$, $([3], [p_2], [5])$, and $([4, 5], [s], [6])$. To better understand this example, it might be helpful to consider the P/T net in Figure 4.2. To improve readability, in the remainder of this example, square brackets of singleton bags are omitted.

The communication function is used to specify concurrency. In principle, any action may occur in parallel with any other action. To formalize this communication function, the bagsum operator is overloaded to actions in $AC(C, A)$ as follows: For any $a, b \in AC(C, A)$, $a \uplus b = (\boldsymbol{ca} \uplus \boldsymbol{cb}, \boldsymbol{sa} \uplus \boldsymbol{sb}, \boldsymbol{pa} \uplus \boldsymbol{pb})$. Thus, the communication function γ on pairs of actions in $AC(C, A)$ is defined as $\uplus$.

Consider the parallel composition of the non-terminating iterations of the four actions introduced above: $X = (1, r, [2, 3]) * \delta \parallel (2, p_1, 4) * \delta \parallel (3, p_2, 5) * \delta \parallel ([4, 5], s, 6) * \delta$. Parallel composition X specifies the process that may execute any (non-empty) subset of the four actions simultaneously an arbitrary number of times.

As explained, the causal state operator can be used to restrict a general parallel composition such as X to all the behaviors allowed when starting from some given initial state. Assume that all relevant causes are internal. That is, $I = \{1, 2, 3, 4, 5, 6\}$. Furthermore, assume that the initial state of the process contains a single cause 1. In the equational theory $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \uplus)$, the following results can be derived. Theorem 6.2 (Expansion of iterations) is used to simplify the calculations. Note that, according to this theorem, the parallel composition of four iterations can perform *fifteen* different initial actions. However, in the initial state of process X, only one of these fifteen actions can actually occur.

$$
\begin{aligned}
& \lambda^I_1(X) \\
= {} & (1, r, [2, 3]) \cdot \lambda^I_{[2,3]}(X) \\
= {} & (1, r, [2, 3]) \cdot ((2, p_1, 4) \cdot \lambda^I_{[3,4]}(X) + (3, p_2, 5) \cdot \lambda^I_{[2,5]}(X) + ([2, 3], [p_1, p_2], [4, 5]) \cdot \lambda^I_{[4,5]}(X)) \\
= {} & (1, r, [2, 3]) \cdot ((2, p_1, 4) \cdot (3, p_2, 5) + (3, p_2, 5) \cdot (2, p_1, 4) + ([2, 3], [p_1, p_2], [4, 5])) \cdot \lambda^I_{[4,5]}(X) \\
= {} & (1, r, [2, 3]) \cdot ((2, p_1, 4) \parallel (3, p_2, 5)) \cdot ([4, 5], s, 6) \cdot \lambda^I_6(X) \\
= {} & (1, r, [2, 3]) \cdot ((2, p_1, 4) \parallel (3, p_2, 5)) \cdot ([4, 5], s, 6) \cdot \delta
\end{aligned}
$$

Note that the process $\lambda^I_1(X)$ executes each of the four actions introduced above only once, while the parallel composition X may execute each of the actions an arbitrary number of times. Another interesting observation is that $\lambda^I_1(X)$ ends in a deadlock. It does not terminate successfully. However, the attentive reader might notice the resemblance between this last result and the behavior of the P/T net of Figure 4.2 in the step semantics of Definition 4.15 (see Example 4.16). As explained in Section 4, the usual semantics of P/T nets does not distinguish deadlock and successful termination. In Section 6.3, an algebraic semantics for labeled P/T nets is defined in which the term $\lambda^I_1(X)$ forms the core of the semantics of the P/T net of Figure 4.2.

In the final result of the derivation in the above example, consumptions and productions of causes are visible whereas the consumption and production of tokens is not visible in the semantics of labeled P/T nets. However, in the algebraic framework, causes are only needed to enforce a causal ordering between actions. In general, it is not necessary that causes are visible in the final result of derivations as in Example 6.3. Therefore, a new class of algebraic operators, the so-called *cause-abstraction operators*, is introduced. A cause-abstraction operator can be used to hide consumptions and productions of causes. It is a renaming operator, as introduced in Table 5.3. In order to gain flexibility, a set of (internal) cause identifiers specifies which consumptions and productions must be hidden. For any set of cause identifiers $I \subseteq C$, the corresponding renaming function $\boldsymbol{ca}(I) : (AC(C, A) \cup \{\delta\}) \to (AC(C, A) \cup \{\delta\})$ restricts the bags representing the consumption and the production of an action to those causes not in I. As required, $\boldsymbol{ca}(\delta) = \delta$. Furthermore, for any $(c, \alpha, p) \in AC(C, A)$, $\boldsymbol{ca}(I)(c, \alpha, p) = (c \upharpoonright (C \setminus I), \alpha, p \upharpoonright (C \setminus I))$.

Example 6.4. Let C, A, γ, X, and I be defined as in Example 6.3. For the sake of readability, a triple $(\mathbf{0}, \alpha, \mathbf{0}) \in AC(C, A)$ is identified with its second element α. It is not difficult to verify that

$$\rho_{\boldsymbol{ca}(I)}(\lambda^I_{[1]}(X)) = [r] \cdot ([p_1] \parallel [p_2]) \cdot [s] \cdot \delta.$$

It is clear that the right-hand term is an appropriate algebraic expression for the behavior of the labeled P/T net of Figure 4.2.

As mentioned in the introduction to this section, the causal state operator can be used to replace to some extent the standard communication mechanism in ACP-style process algebra. A good example illustrating the use of communication is the specification of a two-place buffer in terms of two one-place buffers (see Examples 5.29, 5.33, and 5.36). Thus, it is an interesting question how to specify a two-place buffer by means of a causal state operator.

Example 6.5. Consider Example 5.36. It is straightforward to adapt the specification of the two-place buffer given in that example to the current setting where actions include input and output causes. Recall that, in Example 5.36, explicit renaming operators are used to resolve conflicts and enforce communications, which has the advantage that the communication function can be used to create a partial-order framework. Hence, assume that parameter γ of the equational theory $(\text{ACP}^*_\lambda + \text{RN} + RSP^* + \text{SC})(AC(C, A), \gamma)$ of this subsection equals bag summation $\uplus$ on actions in $AC(C, A)$. Let A be some set of atomic actions that includes the actions r_1, r_2, s_2, c_2, and s_3; let C be a set of causes including the causes $1, 2, 3, 4$, and 5.

Assume that a bag $\alpha \in \mathcal{B}(A)$ denotes the action $(\mathbf{0}, \alpha, \mathbf{0})$ in $AC(C, A)$. Under this assumption, the two terms $B_{1,2} = ([r_1] \cdot [s_2]) * \delta$ and $B_{2,3} = ([r_2] \cdot [s_3]) * \delta$ specify two one-place buffers. In order to specify a two-place buffer, it is necessary to resolve conflicts and enforce communication. Since conflicts play no role in the behavior of the two-place buffer, it suffices to adapt the renaming function $\boldsymbol{cm}$ of Example 5.36 to the current setting where actions contain information about causes. This renaming function can be lifted to actions in $AC(C, A)$ as follows. For any $(c, \alpha, p) \in AC(C, A)$, $\boldsymbol{cm}(c, \alpha, p) = (c, \boldsymbol{cm}(\alpha), p)$. To enforce communication, actions containing isolated atomic actions that should communicate must be encapsulated. Therefore, let $H \subseteq AC(C, A)$ be the set $\{(c, \alpha, p) \in AC(C, A) \mid r_2 \in \alpha \vee s_2 \in \alpha\}$. The two-place buffer can now be specified as follows.

$$\partial_H(\rho_{\boldsymbol{cm}}(B_{1,2} \parallel B_{2,3})) \tag{1}$$

It is not difficult to verify that it can be derived from the axioms of $(\text{ACP}^*_\lambda + \text{RN} + RSP^* + \text{SC})(AC(C, A), \uplus)$ that this term is equivalent to the following term.

$$[r_1] \cdot (([c_2] \cdot ([r_1] \parallel [s_3])) * \delta) \tag{2}$$

It is also possible to specify a two-place buffer by means of a causal state operator. The specification follows the same approach as the one illustrated in Examples 6.3 and 6.4. Let I be the set of causes $\{1, 2, 3, 4, 5\}$. Consider the following specification.

$$\rho_{\boldsymbol{ca}(I)}(\lambda^I_{[1,2]}(([1, 2], [r_1], [3]) * \delta \parallel ([3], [c_2], [4, 5]) * \delta \parallel ([4], [r_1], [1]) * \delta \parallel ([5], [s_3], [2]) * \delta)) \tag{3}$$

Calculations similar to the calculations in Example 6.3 show that also this specification is derivably equal to term (2). Hence, the two specifications of a two-place buffer using (explicit) communication and a causal state operator, respectively, are equivalent. However, it is interesting to compare these two specifications. Considering term (2), which is clearly the most concrete representation of the behavior of a two-place buffer, specification (1) is in some sense more abstract than specification (3). Specification (1) is a modular specification that specifies the two-place buffer in terms of two one-place buffers. It contains actions that cannot occur in isolation, but must participate in a communication. Specification (3) emphasizes the causal ordering of actions. It contains only actions that also occur in term (2).

The explanations concerning the behavior of the causal state operator given in this subsection, are based on an intuitive understanding of its operational semantics. The next subsection formalizes the semantics of the causal state operator.

6.2 A semantics

As in Section 5.2, the basis of a semantics for the theory $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma)$ is a process space. Let $\mathcal{C}(C, A)$ be the set of closed $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma)$ terms. The set of processes of the process space is the set $\mathcal{C}(C, A) \cup \{\surd\}$, where $\surd$ is the special process that can perform no actions, but can only terminate successfully. The set $AC(C, A)$ is the set of actions of the process space. The transition relation $\_ \xrightarrow{\_} \_ \subseteq (\mathcal{C}(C, A) \cup \{\surd\}) \times AC(C, A) \times (\mathcal{C}(C, A) \cup \{\surd\})$ is the smallest relation satisfying the derivation rules in Tables 5.17 and 6.6. The termination predicate is the singleton $\{\surd\}$.

$a : AC(C, A);\ I : \mathcal{P}(C);\ m : \mathcal{B}(I);\ p, p' : \mathcal{C}(C, A);$

$$\frac{p \xrightarrow{a} p'}{\lambda^I_{m \uplus \boldsymbol{ca}\lceil I}(p) \xrightarrow{a} \lambda^I_{m \uplus \boldsymbol{pa}\lceil I}(p')} \qquad \frac{p \xrightarrow{a} \surd}{\lambda^I_{m \uplus \boldsymbol{ca}\lceil I}(p) \xrightarrow{a} \surd}$$

Table 6.6: The transition relation for the causal state operator.

The process space $(\mathcal{C}(C, A) \cup \{\surd\}, AC(C, A), \longrightarrow, \{\surd\})$ can be turned into a model $\mathcal{M}(C, A, \gamma)$ of the equational theory $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma)$ following the same approach as in Section 5.2. The following property is needed in the construction.

Property 6.7. (Congruence) *Bisimilarity,* $\sim$ *, is a congruence for the operators of* $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma)$.

Proof. The property follows from the format of the derivation rules in Tables 5.17 and 6.6. For details, the reader is referred to [1, 5]. □

The following theorem formulates the important result of this subsection. The equational theory $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma)$ is a sound axiomatization of the model of closed terms modulo bisimilarity.

Theorem 6.8. (Soundness) *For any closed terms* $p, q \in \mathcal{C}(C, A)$,
$(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma) \vdash p = q \ \Rightarrow\ \mathcal{M}(C, A, \gamma) \models p = q.$

Proof. Given Theorem 5.19 and Property 6.7, it suffices to show the validity of the Axioms *CSO*1 through *CSO*5. It is not difficult to construct a bisimulation for each case. □

Example 6.9. Consider again Examples 6.3 and 6.4. Recall that the communication function γ is defined as bag summation on $AC(C, A)$. Omitting consumptions and productions of causes and representing closed terms by dots, the semantics of the term $\rho_{\boldsymbol{ca}(I)}(\lambda^I_{[1]}(X))$ in the model $\mathcal{M}(C, A, \uplus)$ is visualized by the process in Figure 3.6 (b) when removing the option to terminate successfully. The model $\mathcal{M}(C, A, \uplus)$ defines a (branching-time, interleaving) step semantics for the equational theory $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \uplus)$.

The model $\mathcal{M}(C, A, \gamma)$ defined in this subsection can also be turned into a total-order semantics. Assume that $\perp$ is the communication function that is undefined for every pair of actions. Consider again the term $\rho_{\boldsymbol{ca}(I)}(\lambda^I_{[1]}(X))$. The semantics of this term in model $\mathcal{M}(C, A, \perp)$ is visualized by the process in Figure 3.6 (a) when, again, removing the option to terminate successfully.

6.3 An algebraic semantics for labeled P/T nets

The causal state operator is inspired by the notions of consumptions and productions of tokens in Petri-net theory. Therefore, it is interesting to study the relationship between the algebraic theory of the previous two subsections and labeled P/T nets as introduced in Section 4. The goal of this subsection is to define an algebraic semantics for labeled P/T nets that is consistent with the step semantics of labeled P/T nets given in Definition 4.15.

Recall from Section 4 that U is some universe of identifiers, including place and token identifiers, and that L is some set of transition labels, which has been used to define steps in the P/T-net framework. The set of actions is defined as $AC(U, L)$. That is, an action is a triple in $\mathcal{B}(U) \times \mathcal{B}(L) \times \mathcal{B}(U)$.

To obtain an algebraic semantics for labeled P/T nets, each labeled P/T net is translated into a closed $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(U, L), \gamma)$ term, where γ is some communication function on $AC(U, L)$. The communication function is a parameter of the algebraic semantics in order to gain flexibility. The standard choice for the communication function γ is bag summation $\uplus$ as it is defined on actions in $AC(U, L)$. This means that arbitrary concurrency between actions is allowed.

The algebraic semantics of a labeled P/T net has a second parameter, namely a set of internal causes. As explained before, the distinction between internal and external causes is meaningful when distinguishing between communication within a process and communication between the process and its environment. The standard choice for this second parameter is the entire universe of identifiers U. This means that all causes are considered to be internal.

Assuming the standard choices for the two parameters of the algebraic semantics for P/T nets, it can be shown that the closed term corresponding to a labeled P/T net has the same step semantics as the P/T net (Theorem 6.12 below). Section 8 shows an application of the framework developed in this subsection with different choices for the parameters.

The basis of the algebraic semantics of a labeled P/T net is a closed term that corresponds to the unrestricted behavior of the P/T net, which is the behavior when every transition is always enabled. A causal state operator instantiated with the marking of the P/T net is applied to this term to restrict the behavior to all possible sequences of steps. A cause-abstraction operator, as introduced in Section 6.1, is used to hide consumptions and productions of causes. The unrestricted behavior of a single transition is the non-terminating iteration of its corresponding action in $AC(U, L)$. The unrestricted behavior of an entire P/T net is the parallel composition of all its transitions. Recall that $\boldsymbol{i}$ and $\boldsymbol{o}$ are functions that assign to each node in a labeled P/T net its bag of input and output nodes, respectively. For a transition, the bag of its input places corresponds to its consumption upon firing and the bag of its output places corresponds to its production upon firing. The set of closed $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(U, L), \gamma)$ terms is abbreviated $\mathcal{C}(U, L)$.

Definition 6.10. (Algebraic semantics for labeled P/T nets) Let (N, m), with $N = (P, T, F, W, \ell)$, be a marked, L-labeled P/T net in $\mathcal{N}$, as defined in Definition 4.5. Let γ be some communication function. Assume that $I \subseteq U$ is a subset of identifiers denoting internal causes. The algebraic semantics of (N, m), denoted $[\![N, m]\!]^I_\gamma$, is a closed term in $\mathcal{C}(U, L)$ defined as follows:
$[\![N, m]\!]^I_\gamma = \rho_{\boldsymbol{ca}(I)}(\lambda^I_m(\|\, t : t \in T : (\boldsymbol{i}t, \ell(t), \boldsymbol{o}t)^* \delta))$.

Example 6.11. Let $(N, [1])$ be the labeled P/T net of Figure 4.2. Consider again Examples 6.3 and 6.4. Assume that the set of causes C, the set of atomic actions A, the set of internal causes I, the closed term X, and the communication function γ are defined as in Example 6.3. Assume that the set of labels L equals the set of atomic actions A. The algebraic semantics of the P/T net $(N, [1])$, $[\![N, [1]]\!]^I_\uplus$, is the term $\rho_{\boldsymbol{ca}(I)}(\lambda^I_{[1]}(X))$ of Example 6.4. The semantics of this term in model $\mathcal{M}(C, A, \uplus)$ of the previous subsection, as discussed in Example 6.9, corresponds to the step semantics of the P/T net $(N, [1])$ as defined in Definition 4.15 and explained in Example 4.16.

The correspondence between the step semantics of a labeled P/T net and the step semantics of its algebraic representation, as observed in the above example, is formalized by the following theorem. It uses the standard choices for the two parameters of the algebraic semantics for labeled P/T nets. The theorem states that any step of a labeled P/T net can be simulated by its algebraic semantics and vice versa. Furthermore, since a labeled P/T net cannot terminate successfully, its algebraic semantics cannot terminate successfully either. This means that the algebraic semantics of a labeled P/T net cannot evolve into the special process $\surd$. The symbol $\equiv$ denotes syntactical equivalence of terms.

Theorem 6.12. *For any labeled P/T nets $(N, m), (N, m') \in \mathcal{N}$, step $\alpha \in \mathcal{B}(L)$, closed term $p \in \mathcal{C}(U, L)$, and action $a \in AC(U, L)$,*

i) $(N, m)\ [\alpha\rangle^s\ (N, m') \ \Rightarrow\ [\![N, m]\!]^U_{\uplus} \xrightarrow{(\mathbf{0},\alpha,\mathbf{0})} [\![N, m']\!]^U_{\uplus}$,

ii) $[\![N, m]\!]^U_{\uplus} \xrightarrow{a} p \ \Rightarrow$
$(\exists\, m', \alpha : m' \in \mathcal{B}(U) \wedge \alpha \in \mathcal{B}(L) : p \equiv [\![N, m']\!]^U_{\uplus} \wedge a = (\mathbf{0}, \alpha, \mathbf{0}) \wedge (N, m)\ [\alpha\rangle^s\ (N, m'))$, *and*

iii) $[\![N, m]\!]^U_{\uplus} \stackrel{a}{\not\longrightarrow} \surd$.

Proof. The proof consists of a straightforward but tedious analysis of the step firing rule of Definition 4.14 and the transition relation of Tables 5.17 and 6.6. The proof is almost identical to the proof of Theorem 3.4.8 in [8], where a similar result is proven in a total-order setting. Therefore, for more details, the reader is referred to [8]. □

Assuming that a triple $(\mathbf{0}, \alpha, \mathbf{0}) \in AC(U, L)$ is identified with its second element α, the correspondence between the step semantics of a labeled P/T net and its algebraic semantics can also be formulated as follows. In the process space that is defined as the (component-wise) union of the process spaces $(\mathcal{N}, \mathcal{B}(L), [\ \rangle^s, \emptyset)$ of Definition 4.15 (Step semantics for labeled P/T nets) and $(\mathcal{C}(U, L) \cup \{\surd\}, AC(U, L), \longrightarrow, \{\surd\})$ underlying model $\mathcal{M}(U, L, \uplus)$ of Section 6.2, a labeled P/T net in $\mathcal{N}$ and its algebraic semantics in $\mathcal{C}(U, L)$ are bisimilar.

A consequence of Definition 6.10 is that the theory $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(U, L), \uplus)$ can be used to reason, in a purely equational way, about the equivalence of labeled P/T nets in a step semantics.

Example 6.13. Consider again the marked, labeled P/T nets of Figures 4.2 and 4.11. Let $(N, [1])$ be the net of Figure 4.2 and let $(K, [1, 7])$ and $(M, [1])$ be the two P/T nets of Figure 4.11. A triple $(\mathbf{0}, \alpha, \mathbf{0}) \in AC(U, L)$ is identified with its second element α. It is derivable from the axioms of $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(U, L), \uplus)$ that $[\![N, [1]]\!]^U_{\uplus} = [r] \cdot ([p_1] \parallel [p_2]) \cdot [s] \cdot \delta$. and that $[\![K, [1, 7]]\!]^U_{\uplus} = [\![M, [1]]\!]^U_{\uplus} = [r] \cdot ([p_1] \cdot [p_2] + [p_2] \cdot [p_1]) \cdot [s] \cdot \delta$. It is not possible to derive that $[\![N, [1]]\!]^U_{\uplus}$ equals $[\![K, [1, 7]]\!]^U_{\uplus}$ or $[\![M, [1]]\!]^U_{\uplus}$.

Given the definitions so far, it is straightforward to obtain an algebraic semantics for labeled P/T nets that corresponds to the total-order semantics of Definition 4.10. As in Example 6.9, let $\perp$ be the communication function that is undefined for every pair of actions. Analogously to Theorem 6.12, it is possible to prove that any labeled P/T net $(N, m) \in \mathcal{N}$ and its algebraic semantics $[\![N, m]\!]^U_{\perp}$ represent corresponding processes in the process spaces $(\mathcal{N}, L, [\ \rangle^t, \emptyset)$ of Definition 4.10 (Total-order semantics for labeled P/T nets) and $(\mathcal{C}(U, L) \cup \{\surd\}, AC(U, L), \longrightarrow, \{\surd\})$ underlying model $\mathcal{M}(U, L, \perp)$ of Section 6.2, respectively.

6.4 Concluding remarks

Summarizing this section, it has been shown how to formalize the notion of transition firing known from Petri-net theory in ACP-style process algebra by means of the so-called causal state operator. The causal state operator can be used to some extent as a substitute for the standard communication mechanism of ACP. In this way, the standard communication mechanism can be used to obtain a partial-order framework. An interesting application of the theory of this section is that it can be used to reason in a purely equational (interleaving) way about labeled P/T nets with a step semantics.

An interesting question is to what extent the causal state operator can replace the standard causality mechanism of ACP-style process algebra consisting of sequential composition and communication. In Sections 9 and 10, this topic is investigated in more detail.

7 Intermezzo on Renaming and Communication Functions

In standard process-algebraic theory as introduced in Section 5.1, actions do not have internal structure. Thus, communication functions and renaming functions are defined on actions without making assumptions about the structure that actions might have. However, in Section 5.4 and Section 6, we have seen several examples of actions that are structured and of communication and renaming functions using that structure of actions. This section summarizes several kinds of communication and renaming functions that use the structure of actions and that are of particular interest in the context of this chapter. It is assumed that the set of actions equals $AC(C, A)$ as defined in Section 6.1, with C some set of causes and A some set of atomic actions. Recall from Section 5.1 that a renaming function $\boldsymbol{f} : AC(C, A) \cup \{\delta\} \to AC(C, A) \cup \{\delta\}$ is any function with the restriction that $\boldsymbol{f}(\delta) = \delta$.

Renamings of atomic actions and causes The first class of renaming functions that is of particular interest are those renaming functions that can be derived from renaming functions on atomic actions. Assume that $f : A \to A$ is a (partial) renaming of atomic actions. It can be turned into a renaming function $\boldsymbol{f} : AC(C, A) \cup \{\delta\} \to AC(C, A) \cup \{\delta\}$ in a standard way. Let the auxiliary function $\bar{f} : A \to A$ be a *total* function defined as follows. For any $a \in A$, $\bar{f}(a)$ equals $f(a)$ if $f(a)$ is defined and a otherwise. Using this auxiliary function, the renaming function $\boldsymbol{f}$ is defined as follows. As required, $\boldsymbol{f}(\delta) = \delta$. Furthermore, for any $(c, \alpha, p) \in AC(C, A)$, $\boldsymbol{f}(c, \alpha, p) = (c, \bar{f}(\alpha), p)$ where $\bar{f}(\alpha) = (\uplus a : a \in \alpha : [(\bar{f}(a))^{\alpha(a)}])$.

Another class of renaming functions are those functions that can be derived from renamings of causes. Let $f : C \to C$ be such a (partial) renaming of causes. The renaming function $\boldsymbol{f} : AC(C, A) \cup \{\delta\} \to AC(C, A) \cup \{\delta\}$ is defined as follows: $\boldsymbol{f}(\delta) = \delta$ and, for any $(c, \alpha, p) \in AC(C, A)$, $\boldsymbol{f}(c, \alpha, p) = (\bar{f}(c), \alpha, \bar{f}(p))$, where the auxiliary function $\bar{f}$ on bags of causes is defined in the same way as the corresponding function on bags of atomic actions above.

Encapsulation operators As explained in Section 5.1, encapsulation operators are an interesting kind of renaming operators. The basis of an encapsulation operator is a set of actions. Elements in this set are renamed to the inaction constant δ, whereas actions not in this set remain unchanged. It has also been explained how such a set can be transformed into a renaming function, thus showing that encapsulation operators belong to the class of renaming operators.

Encapsulation operators can also be derived from sets of *atomic* actions. Let $H \subseteq A$ be a set of atomic actions. The idea is that the corresponding encapsulation operator blocks actions that contain at least one atomic action of this set. Thus, assume that $\boldsymbol{H} = \{(c, \alpha, p) \in AC(C, A) \mid \alpha \upharpoonright H \neq \mathbf{0}\}$. The encapsulation operator $\partial_{\boldsymbol{H}}$ has the desired effect. The set $\boldsymbol{H}$ can be turned into a renaming function $\boldsymbol{e}(\boldsymbol{H}) : AC(C, A) \cup \{\delta\} \to AC(C, A) \cup \{\delta\}$ in the way explained in Section 5.1.

Another set of encapsulation operators can be derived from sets of causes. For any set of causes $H \subseteq C$, the corresponding set $\boldsymbol{H} \subseteq AC(C, A)$ is defined as $\{(c, \alpha, p) \in AC(C, A) \mid (c \uplus p) \upharpoonright H \neq \mathbf{0}\}$. The encapsulation operator $\partial_{\boldsymbol{H}}$ blocks any action that consumes or produces a cause in H. The corresponding renaming function $\boldsymbol{e}(\boldsymbol{H}) : AC(C, A) \cup \{\delta\} \to AC(C, A) \cup \{\delta\}$ is defined in the obvious way. Note that it is also straightforward to define encapsulation operators that encapsulate only actions that consume causes in H or only actions that produce causes in H.

Communication functions Yet another kind of interesting renaming functions are those renaming functions that can be interpreted as explicit communication functions (see Example 5.36). Any associative and commutative (partial) *communication* function on *atomic* actions $cm : A \times A \to A$ can be turned into a renaming function $\boldsymbol{cm} : AC(C, A) \cup \{\delta\} \to AC(C, A) \cup \{\delta\}$ in a standard way. Let, for any $a, b \in A$, the auxiliary function $\bar{cm}(a, b) : \mathcal{B}(A) \to \mathcal{B}(A)$ be defined as follows: For any $\alpha \in \mathcal{B}(A)$, $\bar{cm}(a, b)(\alpha) = \alpha - [a^n, b^n] \uplus [cm(a, b)^n]$ if $cm(a, b)$ is defined, $[a, b] \leq \alpha$, and $n = \alpha(a) \min \alpha(b)$; $\bar{cm}(a, b)(\alpha) = \alpha$ otherwise. Thus, in case the bag α contains at least one pair of atomic actions a and b, function $\bar{cm}(a, b)$ replaces any occurrence of that pair in α by the atomic action $cm(a, b)$. Another auxiliary function $\bar{cm} : \mathcal{B}(A) \to \mathcal{B}(A)$ is defined as the function composition of all functions $\bar{cm}(a, b)$ with $a, b \in A$. Since cm is associative and commutative, the order of the composition is not important. Function $\bar{cm}$ is a function that renames all pairs of communicating atomic actions. Finally, the desired communication function $\boldsymbol{cm}$ can now be defined as follows: $\boldsymbol{cm}(\delta) = \delta$ and, for any $(c, \alpha, p) \in AC(C, A)$, $\boldsymbol{cm}(c, \alpha, p) = (c, \bar{cm}(\alpha), p)$.

Of course, communication is not necessarily achieved by means of explicit renaming. On the contrary, as explained in Section 5, the standard approach to communication in ACP-style process algebra is to define communications by means of a communication function $\gamma : AC(C, A) \times AC(C, A) \to AC(C, A)$. However, any associative and commutative (partial) communication function on atomic actions $cm : A \times A \to A$ can simply be turned into a communication function γ on actions by means of the renaming function $\boldsymbol{cm}$ defined above. For any $a, b \in AC(C, A)$, $\gamma(a, b) = \boldsymbol{cm}(a \uplus b)$.

Cause abstraction The last class of renaming functions mentioned in this section is the class of renaming functions that forms the basis for the cause-abstraction operators introduced in Section 6.1. Clearly, also these renaming functions use the structure of actions in $AC(C, A)$.

Finally, observe that the definitions of the renaming functions that do not affect causes as well as the definition of the communication functions given in this section can easily be adapted to the simpler framework of Section 5.4, where actions are defined as bags of atomic actions.

8 Modular P/T Nets

The framework developed in Section 6 provides the basis for reasoning about the equivalence of labeled P/T nets in an equational way. However, the various examples show that it will be difficult to reason about the behavior of non-trivial P/T-net models of complex concurrent systems. A formalism for modeling and analyzing real-world concurrent systems must support *compositional* reasoning. A formalism is compositional if it allows the system designer to structure a formal model of a system into component models of subsystems in such a way that properties of the system as a whole can be derived from the properties of the components. The goal of this section is to illustrate by means of a few examples in what way compositionality can be achieved in the context developed so far. It is beyond the scope of this chapter to go into much detail. The interested reader is referred to [8, Chapter 3], which describes in detail a formalism supporting the compositional design of concurrent systems. The approach is similar to the one followed in Section 8.1. However, it is important to note that the formalism in [8, Chapter 3] is based on a total-order semantics.

The basis of any compositional formalism based on P/T nets is always some notion of a *modular* P/T net with a mechanism to construct modular P/T nets from other modular P/T nets. The idea is that a modular P/T net models a system component that may interact with its environment via a well defined *interface*.

There are several ways to modularize P/T nets based on how the interface of a modular P/T net is defined. In the remainder of this section, two approaches are illustrated. In one approach, the interface of a modular P/T net consists of *places*, whereas in the other approach, it consists of *transitions*. Figure 8.1 shows two modular-P/T-net models of a one-place buffer that are explained in more detail below.

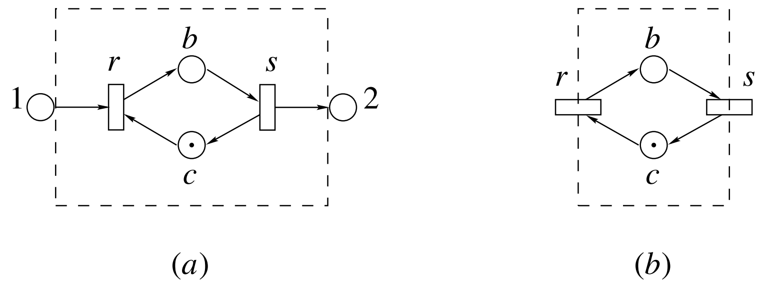


Figure 8.1: Modular-P/T-net models of a buffer.

8.1 Place fusion

The most commonly used approach to modularizing Petri nets is the one where interfaces between modular nets are based on places. It appears in the Petri-net frameworks of, for example, [15, 16, 34, 38] and [8, Chapter 3]. It is also supported by tools as Design/CPN [39], ExSpect [7], and PEP [18].

Figure 8.1(a) shows a labeled P/T net of which the set of places is partitioned into two sets by means of a dashed box. Places inside the box are *internal places*, whereas places outside the box are so-called *pins*. Pins are connectors between the system and its environment. Interaction between the system and its environment is established by exchanging tokens via these pins. A P/T net as shown in Figure 8.1(a) is a very simple example of a modular P/T net. The set of pins defines the interface of the modular net. The mechanism to construct new modular P/T nets from other modular nets is explained in more detail below.

The behavior of a modular P/T net as shown in Figure 8.1(a) and an ordinary P/T net differs in one important aspect. When considering the behavior of a modular P/T net with an interface consisting of pins, the environment is responsible for consuming tokens from and producing tokens on the pins. If the behavior of the modular P/T net is described without the context of a specific environment, then it is assumed that the environment has always tokens available when they are needed by the modular P/T net; it is also assumed that the environment is always willing to receive any tokens that the modular P/T net might produce. If a modular P/T net is put into a specific context, typically defined by another modular P/T net, the approach followed in this subsection guarantees that the behavior of the modular P/T net is restricted to the behavior allowed by its context.

Example 8.2. Consider the modular P/T net of Figure 8.1(a). As mentioned, it models a one-place buffer. Pins 1 and 2 correspond to ports. The buffer receives messages of its environment over port 1; it sends messages to its environment over port 2. Place b models buffer storage. The number of tokens in place c determines the *capacity* of the buffer. In the example, the capacity is one.

As explained, since no specific environment of the buffer has been given, it is assumed that the environment has always tokens available when they are needed by the buffer. Therefore, in the initial marking depicted in Figure 8.1(a), transition r is enabled. It has tokens on all its *internal* input places. Transition s is not enabled, because place b is not marked. Firing transition r leads to the marking with a single token in place b. In that marking, only transition s is enabled. Firing transition s, returns the modular P/T net to the state shown in Figure 8.1(a). The environment has received the token over port 2. Clearly, the behavior of the modular net corresponds to a one-place buffer.

By adding tokens to place c, it is possible to increase the capacity of the buffer. It is not difficult to see that the modular P/T net of Figure 8.1(a) behaves as an n-place buffer when place c contains n tokens in the initial marking. Note that the modular P/T net is a very abstract model that abstracts from message contents and message ordering. The *number of* tokens in place b corresponds to the *number of* messages in the buffer; the tokens do not correspond to the messages themselves.

It is possible to formalize the behavior of modular P/T nets as illustrated in the above example by means of a (step) firing rule. However, it is also possible to formally define the behavior in the equational theory $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(U, L), \uplus)$, where U, L, and $AC(U, L)$ are defined as in Section 6.3.

Example 8.3. Let I be the set of internal places $\{b, c\}$. Recall the algebraic semantics for labeled P/T nets given in Definition 6.10. The algebraic semantics of the modular P/T net of Figure 8.1(a) is the following closed term in $\mathcal{C}(U, L)$. Square brackets of singleton bags are omitted.

$$\rho_{\boldsymbol{ca}(I)}(\lambda^I_c(([1, c], r, b) * \delta \parallel (b, s, [c, 2]) * \delta)).$$

It is straightforward to prove that this term is equivalent to

$$((1, r, \mathbf{0}) \cdot (\mathbf{0}, s, 2)) * \delta.$$

Note that only *internal* consumptions and productions are hidden. The reason for not hiding external consumptions and productions will become clear when we consider the construction of modular P/T nets from other modular P/T nets. It provides a way for restricting the behavior of a modular P/T net in a specific context.

The algebraic semantics of the modular P/T net of Figure 8.1(a) with *two* tokens in place c is the term

$$\rho_{\boldsymbol{ca}(I)}(\lambda^I_{[c^2]}(([1, c], r, b) * \delta \parallel (b, s, [c, 2]) * \delta)),$$

which can be proven equal to

$$(1, r, \mathbf{0}) \cdot (((1, r, \mathbf{0}) \parallel (\mathbf{0}, s, 2)) * \delta).$$

This last result is the expected term for a two-place buffer.

Finally, consider the variant of the modular P/T net of Figure 8.1(a) with *three* tokens in c. Its algebraic semantics is the term

$$\rho_{\boldsymbol{ca}(I)}(\lambda^I_{[c^3]}(([1, c], r, b) * \delta \parallel (b, s, [c, 2]) * \delta)).$$

Let X_i, where $0 \le i \le 3$, denote the closed term $\rho_{\boldsymbol{ca}(I)}(\lambda^I_{[b^i, c^{3-i}]}(([1, c], r, b) * \delta \parallel (b, s, [c, 2]) * \delta))$. Thus, $X_0 = \rho_{\boldsymbol{ca}(I)}(\lambda^I_{[c^3]}(([1, c], r, b) * \delta \parallel (b, s, [c, 2]) * \delta))$. The following results can be obtained.

$$\begin{aligned}
X_0 &= (1, r, \mathbf{0}) \cdot X_1,\\
X_1 &= (1, r, \mathbf{0}) \cdot X_2 + (\mathbf{0}, s, 2) \cdot X_0 + (1, [r, s], 2) \cdot X_1,\\
X_2 &= (1, r, \mathbf{0}) \cdot X_3 + (\mathbf{0}, s, 2) \cdot X_1 + (1, [r, s], 2) \cdot X_2, \text{ and}\\
X_3 &= (\mathbf{0}, s, 2) \cdot X_2.
\end{aligned}$$

The semantics of term X_0 is illustrated in Figure 8.4. Note that the final result for the *two*-place buffer above is a single closed term in $\mathcal{C}(U, L)$ that contains only the sequential-composition, merge, and binary-Kleene-star operators. In particular, it does not contain a causal state operator. There is no such term in $\mathcal{C}(U, L)$ that specifies a three-place buffer. That is, there is no closed term in $\mathcal{C}(U, L)$ that is equivalent to term X_0 and that contains only the sequential-composition, merge, and binary-Kleene-star operators. The reason is that the binary Kleene star is not sufficiently expressive. More information about the expressiveness of the binary Kleene star can be found in [11, 12].

The above example shows how to obtain a modular-P/T-net model of a buffer with arbitrary capacity by varying the marking of the modular net. The examples in the algebraic framework of Section 5 suggest an-

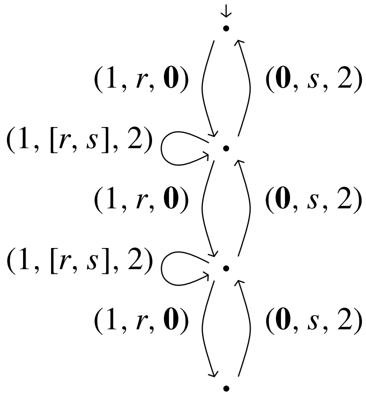
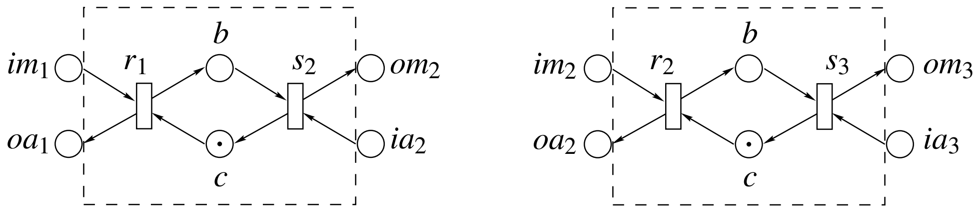
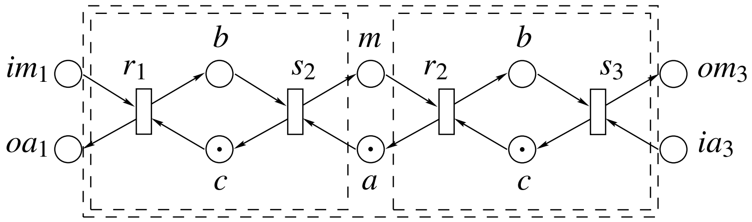


Figure 8.4: A three-place buffer in a step semantics.

other way to construct buffers with arbitrary capacity, namely by composing a number of one-place buffers. This approach is also possible in the framework of modular P/T nets developed so far.



(a)



(b)

Figure 8.5: The modular construction of a three-place buffer.

Example 8.6. Figure 8.5(a) shows two variants of the one-place buffer of Figure 8.1(a). The left P/T net models a buffer that receives messages of its environment over port 1 and sends messages over port 2. The right P/T net models a buffer communicating via ports 2 and 3. There is one important difference between the models of Figure 8.5(a) and the model of Figure 8.1(a). Consider the left P/T net of Figure 8.5(a). It models a buffer that receives messages via pin im_1 while at the same time acknowledging the receipt to its environment via pin oa_1. The buffer sends messages to the environment via pin om_2; however, it only sends a message when the environment acknowledges via pin ia_2 that it is ready to receive the message. The two acknowledgment pins oa_1 and ia_2 are not present in the modular P/T net of Figure 8.1(a). They can be used to construct asynchronous communication channels with a finite message capacity between buffers.

To illustrate the composition of modular P/T nets with interfaces consisting of pins, consider the modular P/T net of Figure 8.5(b). It consists of four components, namely the two one-place buffers of Figure 8.5(a), which are called *subnets* of the modular P/T net, and two new (internal) places m and a. Place a is marked with a single token. The modular P/T net of Figure 8.5(b) specifies a specific context for its subnets. It is constructed from its four components by means of a so-called *place-fusion* function. A place-fusion function is a (partial) mapping from the pins of the subnets of a modular P/T net onto the places of the modular P/T net. In the example of Figure 8.5(b), the place-fusion function f maps pins om_2 and im_2 onto place m and pins ia_2 and oa_2 onto place a.

Informally, the modular P/T net of Figure 8.5(b) models the two one-place buffers of Figure 8.5(a) that are connected via port 2 which, in this example, is an asynchronous communication channel that can hold a single message. The channel is modeled by places m and a. Place m models the storage for messages and the number of tokens in place a models the storage capacity of the channel. Intuitively, the modular P/T net of Figure 8.5(b) should behave as a *three*-place buffer. Messages can be stored in the two one-place buffers (the two places marked b) *and* in the channel (place m). The modular P/T net of Figure 8.5(b) is more concrete than the modular P/T net of Figure 8.1(a) with three tokens in place c that also models a three-place buffer. Recall that the modular net of Figure 8.1(a) abstracts from message contents and message ordering. Tokens in the three buffer places of the modular P/T net of Figure 8.5(b) can be seen as messages, which means that this modular net abstracts from message contents, but not from message ordering.

At this point, the reason for the exchange of acknowledgments between the one-place buffers of Figure 8.5(a) and the environment should be clear. In the modular P/T net of Figure 8.5(b) without place a, there would be no upper bound to the number of tokens in place m. Such a modular P/T net conforms to a system where the communication channel between the two one-place buffers has infinite capacity.

In a general framework of modular P/T nets, a modular P/T net may consist of subnets, transitions, places, and arcs between transitions and places. Places may be divided into internal places and pins. Internal places may contain an arbitrary number of tokens. Note that the P/T net of Figure 8.1(a) as well as the three P/T nets of Figure 8.5 all belong to this general class of modular P/T nets. In such a general framework of modular P/T nets, it is possible to construct arbitrarily complex P/T-net models using only the simple mechanism of place fusion. Note that pins of the subnets of a modular P/T net may be fused with both its internal places and pins.

The semantics of modular P/T nets as described above can be formalized by defining a process space based on a (step) firing rule. However, it is also possible to define the semantics indirectly via an algebraic semantics.

Example 8.7. Recall the algebraic semantics of the modular P/T net of Figure 8.1(a) given in Example 8.3. The algebraic semantics of the leftmost modular P/T net of Figure 8.5(a) is the following closed term in $\mathcal{C}(U, L)$. As before, square brackets of singleton bags are omitted. Let I be the set of causes $\{b, c\}$.

$$\rho_{\boldsymbol{ca}(I)}(\lambda_c^I(([im_1, c], r_1, [oa_1, b])^*\delta \parallel ([b, ia_2], s_2, [c, om_2])^*\delta)).$$

It is straightforward to prove that this term is equivalent to

$$((im_1, r_1, oa_1) \cdot (ia_2, s_2, om_2))^*\delta.$$

Let $C_{1,2}$ be an abbreviation for this term. The algebraic semantics of the rightmost modular P/T net of Figure 8.5(a) is the term

$$\rho_{\boldsymbol{ca}(I)}(\lambda_c^I(([im_2, c], r_2, [oa_2, b])^*\delta \parallel ([b, ia_3], s_3, [c, om_3])^*\delta)),$$

which is equivalent to

$$((im_2, r_2, oa_2) \cdot (ia_3, s_3, om_3))^*\delta.$$

Let $C_{2,3}$ be an abbreviation for this term. Recall that the exchange of tokens between the modular P/T nets of Figure 8.5(a) and the environment is represented in the terms $C_{1,2}$ and $C_{2,3}$ explicitly via causes.

It remains to define the algebraic semantics of the modular P/T net of Figure 8.5(b). Since the goal is to obtain a compositional formalism, it must be based on the terms $C_{1,2}$ and $C_{2,3}$ derived for the two one-place buffers of which it is composed. As explained in Example 8.6, the basis of the construction of this modular P/T net from its components is the place-fusion function f, which maps (some of the) pins of its subnets onto its places. To obtain an algebraic expression for the behavior of subnets in the context of a modular P/T net, it suffices to simply rename consumptions and productions of causes corresponding to pins of subnets according to the place-fusion function. Thus, the place-fusion function in the construction of a modular P/T net corresponds to a renaming operator in its algebraic semantics. Note that the place-fusion function f is simply a (partial) renaming of causes in U. Consequently, the place-fusion function f can be turned into a renaming function $\boldsymbol{f} : (AC(U, L) \times \{\delta\}) \to (AC(U, L) \times \{\delta\})$ as explained in Section 7.

Using the renaming function $\boldsymbol{f}$, the behavior of the modular P/T nets of Figure 8.5(a) in the context of the modular net of Figure 8.5(b) is defined by the terms $\rho_{\boldsymbol{f}}(C_{1,2})$ and $\rho_{\boldsymbol{f}}(C_{2,3})$, respectively. It follows from the axioms for renaming that

$$\rho_{\boldsymbol{f}}(C_{1,2}) = ((im_1, r_1, oa_1) \cdot (a, s_2, m))^* \delta$$

and that

$$\rho_{\boldsymbol{f}}(C_{2,3}) = ((m, r_2, a) \cdot (ia_3, s_3, om_3))^* \delta.$$

The basis of the algebraic semantics of the modular P/T net in Figure 8.5(b) is the parallel composition of the behavior of its subnets in the context of the modular net. A causal state operator instantiated with the marking of the modular P/T net is applied to this term to restrict the behavior to all possible sequences of steps that are allowed in the specific context. The cause-abstraction operator is used to hide internal consumptions and productions of tokens. Thus, the modular P/T net of Figure 8.5(b) has the following algebraic semantics. Let J be the set of places $\{a, m\}$.

$$\rho_{\boldsymbol{ca}(J)}(\lambda_a^J(\rho_{\boldsymbol{f}}(C_{1,2}) \parallel \rho_{\boldsymbol{f}}(C_{2,3}))).$$

Let X_{000} be an abbreviation for this term. The following equations can be derived from the axioms of $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(U, L),\ \uplus)$.

$$\begin{aligned}
X_{000} &= (im_1, r_1, oa_1) \cdot X_{100},\\
X_{100} &= (\mathbf{0}, s_2, \mathbf{0}) \cdot X_{010},\\
X_{010} &= (im_1, r_1, oa_1) \cdot X_{110} + (\mathbf{0}, r_2, \mathbf{0}) \cdot X_{001} + (im_1, [r_1, r_2], oa_1) \cdot X_{101},\\
X_{110} &= (\mathbf{0}, r_2, \mathbf{0}) \cdot X_{101},\\
X_{001} &= (im_1, r_1, oa_1) \cdot X_{101} + (ia_3, s_3, om_3) \cdot X_{000} + ([im_1, ia_3], [r_1, s_3], [oa_1, om_3]) \cdot X_{100},\\
X_{101} &= (\mathbf{0}, s_2, \mathbf{0}) \cdot X_{011} + (ia_3, s_3, om_3) \cdot X_{100} + (ia_3, [s_2, s_3], om_3) \cdot X_{010},\\
X_{011} &= (im_1, r_1, oa_1) \cdot X_{111} + (ia_3, s_3, om_3) \cdot X_{010} + ([im_1, ia_3], [r_1, s_3], [oa_1, om_3]) \cdot X_{110}, \text{ and}\\
X_{111} &= (ia_3, s_3, om_3) \cdot X_{110},
\end{aligned}$$

where

$$\begin{aligned}
X_{100} &= \rho_{\boldsymbol{ca}(J)}(\lambda_a^J((a, s_2, m) \cdot \rho_{\boldsymbol{f}}(C_{1,2}) \parallel \rho_{\boldsymbol{f}}(C_{2,3}))),\\
X_{010} &= \rho_{\boldsymbol{ca}(J)}(\lambda_m^J(\rho_{\boldsymbol{f}}(C_{1,2}) \parallel \rho_{\boldsymbol{f}}(C_{2,3}))),\\
X_{110} &= \rho_{\boldsymbol{ca}(J)}(\lambda_m^J((a, s_2, m) \cdot \rho_{\boldsymbol{f}}(C_{1,2}) \parallel \rho_{\boldsymbol{f}}(C_{2,3}))),\\
X_{001} &= \rho_{\boldsymbol{ca}(J)}(\lambda_a^J(\rho_{\boldsymbol{f}}(C_{1,2}) \parallel (ia_3, s_3, om_3) \cdot \rho_{\boldsymbol{f}}(C_{2,3}))),\\
X_{101} &= \rho_{\boldsymbol{ca}(J)}(\lambda_a^J((a, s_2, m) \cdot \rho_{\boldsymbol{f}}(C_{1,2}) \parallel (ia_3, s_3, om_3) \cdot \rho_{\boldsymbol{f}}(C_{2,3}))),\\
X_{011} &= \rho_{\boldsymbol{ca}(J)}(\lambda_m^J(\rho_{\boldsymbol{f}}(C_{1,2}) \parallel (ia_3, s_3, om_3) \cdot \rho_{\boldsymbol{f}}(C_{2,3}))), \text{ and}\\
X_{111} &= \rho_{\boldsymbol{ca}(J)}(\lambda_m^J((a, s_2, m) \cdot \rho_{\boldsymbol{f}}(C_{1,2}) \parallel (ia_3, s_3, om_3) \cdot \rho_{\boldsymbol{f}}(C_{2,3}))).
\end{aligned}$$

As mentioned in Example 8.6, the modular P/T net of Figure 8.5(b) should behave as a three-place buffer. This means that the above set of equations for the terms X_{000} through X_{111} must also represent the behavior of a three-place buffer. Figure 8.8 illustrates the semantics of the term X_{000}. For the sake of clarity, consumptions and productions are omitted. Recall that Figure 8.4 shows a process corresponding to a three-place buffer in a step semantics. Thus, it may be expected that the process in Figure 8.8 is similar to this process. The first impression might be that this is not the case. However, observe that the modular P/T net of Figure 8.5(b) contains two transitions that do not have any external effects, namely transitions s_2 and r_2. Hiding these transitions in the process of Figure 8.8 means that the process states X_{100}, X_{010}, and X_{001} collapse into a single state. The same happens to the process states X_{110}, X_{101}, and X_{011}. A consequence of these abstractions is that the process becomes identical – up to renaming of actions – to the process in Figure 8.4. Thus, the process of Figure 8.8 represents the behavior of a three-place buffer indeed. As mentioned earlier, it goes beyond the scope of this chapter to formalize the notion of abstraction.

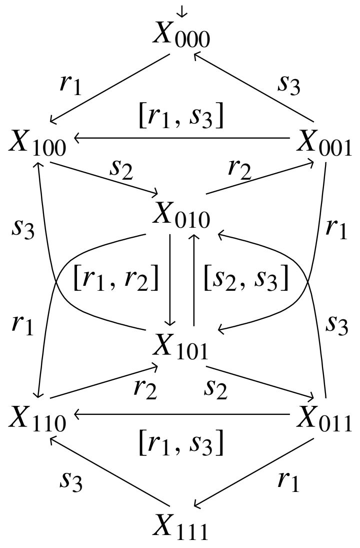


Figure 8.8: The semantics of the modular P/T net of Figure 8.5(b).

The examples given so far in this subsection illustrate the essentials of a compositional formalism based on modular P/T nets with interfaces consisting of places in a partial-order framework. The formalism is compositional in the sense that the behavior of a modular P/T net is defined as a closed term in the equational theory $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(U, L), \uplus)$ that is itself defined in terms of the (parallel) composition of the algebraic semantics of the subnets of the modular P/T net. In [8, Chapter 3], it is explained how such a formalism can be used as the basis for a design method for concurrent systems. The basic idea of this design method is to specify crucial behavioral properties of a concurrent system in process-algebraic terms, model the system by means of a modular P/T net, and verify its behavioral properties via the algebraic semantics of the modular-P/T-net model. The formalism in [8, Chapter 3] includes a formal definition of modular P/T nets. It also formalizes the operational semantics of modular P/T nets in terms of a firing rule. Several algebraic semantics of modular P/T nets are given and it is shown that these semantics correspond to the operational semantics of modular P/T nets as defined by the firing rule (cf. Theorem 6.12). In addition, the notion of abstraction is formalized both in the P/T-net framework and in the algebraic framework. Although the underlying semantics is a total-order semantics, the framework in [8, Chapter 3] can be adapted to the partial-order semantics of this subsection in a straightforward way.

8.2 Transition fusion

As explained in the introduction to this section, there are two ways to modularize P/T nets. In the previous subsection, we have concentrated on modular P/T nets with interfaces consisting of places. Another approach is to define interfaces in terms of transitions. This approach is not very common in the Petri-net literature. However, it is closely related to the notion of synchronous communication known from process algebra, as the examples in the remainder of this subsection show.

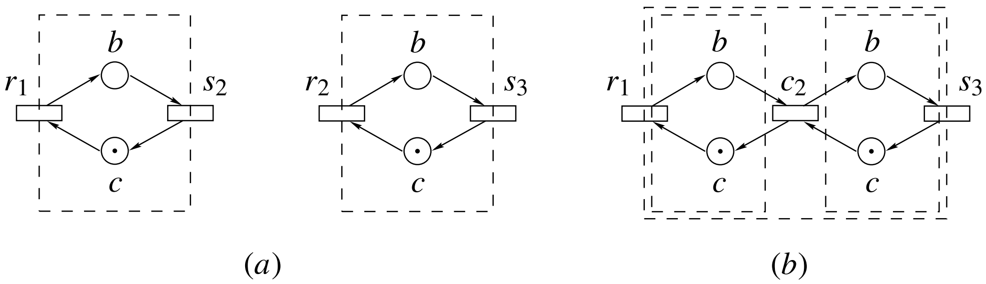


Figure 8.9: The modular construction of a two-place buffer.

Example 8.10. Figure 8.9(a) shows two modular P/T nets that are simple variants of the modular P/T net of Figure 8.1(b). Consider for example the leftmost modular P/T net of Figure 8.9(a). It will not come as a surprise that it models a one-place buffer that receives messages of its environment over port 1 and sends messages to its environment over port 2. The number of tokens in place b models the number of messages in storage, whereas the number of tokens in place c models the capacity of the buffer. The rightmost modular P/T net of Figure 8.9(a) models a buffer that communicates with its environment via ports 2 and 3.

In the P/T-net models of Figure 8.9(a), a dashed box divides the set of transitions of a modular net into *internal transitions* and *transition pins*. The modular P/T nets of Figure 8.9(a) do not have any internal transitions. Transition pins are connectors between a system and its environment. A modular P/T net with transition pins *in isolation* behaves in exactly the same way as a normal labeled P/T net. However, an environment can interact with a system via *synchronous communication*. The modular P/T net of Figure 8.9(b) illustrates one possible mechanism to obtain synchronization. Recall that the basic P/T-net framework in this chapter is the class of *labeled* P/T nets. However, for the sake of simplicity, it is assumed that in the modular P/T nets of Figure 8.9 transition labels are identical to transition identifiers.

The modular net of Figure 8.9(b) consists of three components, namely the two one-place buffers of Figure 8.9(a), called its subnets, and one new (internal) transition c_2. Synchronization between the two one-place buffers is achieved by means of a *transition-fusion* function. A transition-fusion function is a (partial) mapping from the transition pins of the subnets of a modular P/T net to the transitions of the modular net. In the example, the transition-fusion function *cm* maps transition pins s_2 and r_2 onto transition c_2. Intuitively, the modular P/T net of Figure 8.9(b) models a two-place buffer.

The semantics of modular P/T nets with transition pins can be formalized via an algebraic semantics.

Example 8.11. As mentioned in Example 8.10, the behavior of the modular P/T nets of Figure 8.9(a) in isolation is identical to the behavior of normal labeled P/T nets. This means that the algebraic semantics for labeled P/T nets of Definition 6.10 can be used to define the semantics of the modular nets of Figure 8.9(a). Note that all the places in a modular P/T net with transition pins are internal. Furthermore, the goal is to obtain a step semantics. Thus, we instantiate the parameters I and γ in Definition 6.10 with the universe of identifiers U and bag summation $\uplus$, respectively. The algebraic semantics of the two modular P/T nets of Figure 8.9(a) are the closed terms

$$\rho_{\mathbf{ca}(U)}(\lambda_c^U((c, r_1, b)^* \delta \parallel (b, s_2, c)^* \delta))$$

and

$$\rho_{\mathbf{ca}(U)}(\lambda_c^U((c, r_2, b)^* \delta \parallel (b, s_3, c)^* \delta))$$

in $\mathcal{C}(U, L)$, respectively. As before, square brackets of singleton bags are omitted. Furthermore, a triple $(\mathbf{0}, \alpha, \mathbf{0})$ in $AC(U, L)$ is identified with its second element. It is straightforward to prove that the above two terms are equivalent to

$$(r_1 \cdot s_2)^* \delta$$

and

$$(r_2 \cdot s_3)^* \delta.$$

It remains to define an algebraic semantics for the modular P/T net of Figure 8.9(b). The basis for the composition of the modular net of Figure 8.9(b) from its components is the transition-fusion function cm that maps transition pins s_2 and r_2 of the one-place buffers of Figure 8.9(a) onto transition c_2. Note that this transition-fusion function can be seen as a (partial) communication function on atomic actions in L. Thus, the transition-fusion function can be turned into an algebraic renaming function $\boldsymbol{cm} : (AC(U, L) \times \{\delta\}) \rightarrow (AC(U, L) \times \{\delta\})$ following the approach of Section 7. That renaming function can be used to define the algebraic semantics of the modular net of Figure 8.9(b) in terms of the algebraic expressions derived for the one-place buffers of Figure 8.9(a). Transition fusion in the framework of modular P/T nets with transition pins corresponds to the synchronous communication mechanism in ACP-style process algebra. Given this observation, the algebraic semantics of the modular P/T net of Figure 8.9(b) is straightforward. The idea is to take the parallel composition of the algebraic semantics of its subnets, rename actions according to the renaming function constructed from the transition-fusion function, and encapsulate isolated actions corresponding to firings of transition pins. The encapsulation operator that is needed to achieve the latter can be derived from the set of atomic actions $H = \{s_2, r_2\}$. Let $\boldsymbol{H} \subseteq AC(U, L)$ be the set of actions that is constructed from H following the approach of Section 7. The semantics of the modular P/T net of Figure 8.9(b) is the following closed term in $\mathcal{C}(U, L)$:

$$\partial_{\boldsymbol{H}}(\rho_{\boldsymbol{cm}}((r_1 \cdot s_2)^* \delta \parallel (r_2 \cdot s_3)^* \delta)).$$

It is not difficult to prove that this term is equivalent to the term

$$r_1 \cdot ((c_2 \cdot (r_1 \parallel s_3))^* \delta),$$

which is the expected expression for a two-place buffer with an internal communication action c_2.

Note that the definition of the algebraic semantics of the modular P/T net of Figure 8.9(b) does not use the causal state operator. However, in a general framework of modular P/T nets with transition pins, a modular P/T net may consist of subnets, transitions, and (internal) places. Transitions may be divided into internal transitions and transition pins and places may contain an arbitrary number of tokens. In such a framework, the basis of the algebraic semantics of the modular P/T net is still the parallel composition of the expressions for its subnets, but this composition is restricted to all possible sequences of steps by means of the causal state operator. Finally, note that the modular P/T nets in Example 8.10 are very simple. In a general framework of modular P/T nets with transition pins, the construction of communication functions from the transition-fusion function might not always be straightforward.

8.3 Concluding remarks

In this section, we have seen examples of two approaches to modularize P/T nets. One approach is based on fusion of places, whereas the other one is based on fusion of transitions. Both techniques translate to renaming operators when defining the (step) semantics of such modular P/T nets in an algebraic way, where place fusion corresponds to the renaming of causes and where transition fusion corresponds to the notion of synchronous communication. Of course, it is possible to combine the two approaches into one general framework of modular P/T nets with interfaces consisting of (place) pins and transition pins. A formalism of modular P/T nets and its algebraic semantics can be used as the basis for a compositional formalism for modeling and analyzing complex concurrent systems. In [8, Chapter 3], such a compositional formalism based on modular P/T nets with interfaces consisting of places is worked out in detail (in a total-order setting).

9 Cause Addition

The previous section illustrates two techniques to modularize P/T nets, namely place fusion and transition fusion. The two techniques are based on the addition of fresh places and transitions, respectively, in order to compose new (modular) P/T nets from other P/T nets. We have seen that the addition of fresh transitions by means of transition fusion corresponds to the notion of synchronous communication in ACP-style process algebra. The new action that results from a synchronous communication in an algebraic expression corresponds to the new transition. It is also possible to define a class of algebraic operators corresponding to the addition of fresh places in P/T nets. Such a class of operators is particularly interesting because places in P/T nets are the standard means to enforce a sequential ordering between actions. Thus, the new operators must be related to the standard sequential-composition operator in ACP-style process algebra. In the remainder of this section, we introduce the class of so-called cause-addition operators and investigate the relation between cause addition and sequential composition. The introduction of cause addition is another step in incorporating the causality mechanism of Petri nets in ACP-style process algebra. Cause addition was first studied in [4].

9.1 The equational theory

Table 9.1 presents the Algebra of Communicating Processes with iteration, cause addition, the causal state operator, renaming, the Recursive Specification Principle for the binary Kleene star, and standard concurrency, abbreviated $\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC}$. It builds upon the equational theory introduced in Section 6. It is parameterized by a set of actions $AC(C, A)$, with C a set of causes and A a set of atomic actions, and a communication function $\gamma : AC(C, A) \times AC(C, A) \to AC(C, A)$. Recall from Section 6.1 the definitions of $AC(C, A)$ and the three auxiliary functions $\boldsymbol{c}, \boldsymbol{p} : AC(C, A) \to \mathcal{B}(C)$ and $\boldsymbol{s} : AC(C, A) \to \mathcal{B}(A)$: $AC(C, A) = \mathcal{B}(C) \times \mathcal{B}(A) \times \mathcal{B}(C)$ and, for any $a = (c, \alpha, p)$ in $AC(C, A)$, $\boldsymbol{c}a = c$, $\boldsymbol{p}a = p$, and $\boldsymbol{s}a = \alpha$.

$(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC})(AC(C, A), \gamma)$			
$(\mathrm{ACP}^{*}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC})(AC(C, A), \gamma)$			
${}^{c}\_,\ \_^{c} : \boldsymbol{P} \to \boldsymbol{P}$;			
$x, y : \boldsymbol{P}$;			
${}^{c}\delta = \delta$	$IC1$	$\delta^{c} = \delta$	$OC1$
${}^{c}a = (\boldsymbol{c}a \uplus [c], \boldsymbol{s}a, \boldsymbol{p}a)$	$IC2$	$a^{c} = (\boldsymbol{c}a, \boldsymbol{s}a, \boldsymbol{p}a \uplus [c])$	$OC2$
${}^{c}(x + y) = {}^{c}x + {}^{c}y$	$IC3$	$(x + y)^{c} = x^{c} + y^{c}$	$OC3$
${}^{c}(x \cdot y) = {}^{c}x \cdot y$	$IC4$	$(x \cdot y)^{c} = x \cdot y^{c}$	$OC4$

Table 9.1: The equational theory $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC})(AC, \gamma)$.

For any cause $c \in C$, the equational theory $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC})(AC(C, A), \gamma)$ contains two new operators when compared to $(\mathrm{ACP}^{*}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC})(AC(C, A), \gamma)$, namely input-cause addition ${}^{c}\_$ and output-cause addition $\_^{c}$. The idea is that, for any process term x, ${}^{c}x$ is the process that behaves as x with the one difference that its initial action consumes an extra cause c; x^{c} is the process that behaves as

x but produces an extra cause c with its last action. Given this informal interpretation, the axiomatization of cause addition is straightforward. In Table 9.1, constant a ranges over $AC(C, A)$. Axioms $IC1$ and $OC1$ state that the inaction constant δ does not consume or produce causes. Axiom $IC2$ says that the addition of an input cause to an action means that it is added to the consumption of the action; Axiom $OC2$ states that the addition of an output cause to an action means that it is added to its production. Axioms $IC3$ and $OC3$ say that both cause-addition operators distribute over choice. Finally, Axioms $IC4$ and $OC4$ state that the addition of an input cause to a sequential composition means that the cause is added to the left operand of the sequential composition, whereas the addition of an output cause to a sequential composition means that the cause is added to the right operand.

9.2 A semantics

As in Sections 5.2 and 6.2, the basis of a semantics for theory $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma)$ is a process space. Let $\mathcal{C}(C, A)$ be the set of closed $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma)$ terms. The set of processes of the process space is defined as the set $\mathcal{C}(C, A) \cup \{\surd\}$. Process $\surd$ is the special process that can perform no actions, but can only terminate successfully. The set $AC(C, A)$ is the set of actions of the process space. The transition relation $\_ \xrightarrow{\_} \_ \subseteq (\mathcal{C}(C, A) \cup \{\surd\}) \times AC(C, A) \times (\mathcal{C}(C, A,) \cup \{\surd\})$ is the smallest relation satisfying the derivation rules in Tables 5.17, 6.6, and 9.2. The termination predicate is the singleton $\{\surd\}$.

$a : AC(C, A); c : C; p, p' : \mathcal{C}(C, A);$

$$\frac{p \xrightarrow{a} p'}{{}^{c}p \xrightarrow{(\boldsymbol{c}a \uplus [c], \boldsymbol{s}a, \boldsymbol{p}a)} p'} \qquad \frac{p \xrightarrow{a} \surd}{{}^{c}p \xrightarrow{(\boldsymbol{c}a \uplus [c], \boldsymbol{s}a, \boldsymbol{p}a)} \surd} \qquad \frac{p \xrightarrow{a} p'}{p^{c} \xrightarrow{a} p'^{c}} \qquad \frac{p \xrightarrow{a} \surd}{p^{c} \xrightarrow{(\boldsymbol{c}a, \boldsymbol{s}a, \boldsymbol{p}a \uplus [c])} \surd}$$

Table 9.2: The transition relation for cause addition.

The process space $(\mathcal{C}(C, A) \cup \{\surd\}, AC(C, A), \longrightarrow, \{\surd\})$ can be turned into a model $\mathcal{M}(C, A, \gamma)$ of the theory $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma)$ following the approach of Section 5.2. The following congruence property is needed in the construction.

Property 9.3. (Congruence) *Bisimilarity,* $\sim$ *, is a congruence for the operators of* $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma)$.

Proof. The desired property follows from the format of the derivation rules in Tables 5.17, 6.6, and 9.2. (See [1, 5], for details.) □

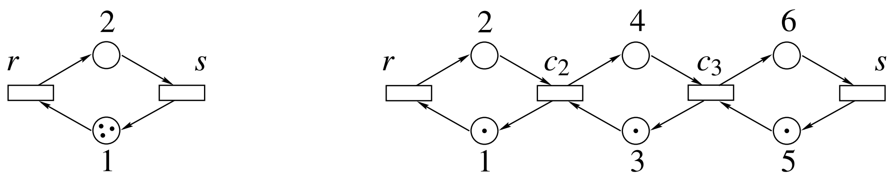
The following theorem states that the equational theory $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma)$ is a sound axiomatization of the model of closed terms modulo bisimilarity.

Theorem 9.4. (Soundness) *For any closed terms* $p, q \in \mathcal{C}(C, A)$,
$(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \gamma) \vdash p = q \;\Rightarrow\; \mathcal{M}(C, A, \gamma) \models p = q.$

Proof. Given Theorem 6.8 and Property 9.3, it suffices to show the validity of the Axioms $IC1$ through $IC4$ and $OC1$ through $OC4$. It is straightforward to construct a bisimulation for each case. □

9.3 Buffers

The use of cause addition can be best illustrated by means of a few examples. Since the cause-addition operators are inspired by Petri-net theory, (variants of) P/T-net models given earlier are a good source of examples.



(a) (b)

Figure 9.5: Two P/T-net models of three-place buffers.

Example 9.6. Consider the P/T-net model of Figure 9.5(a). It is a variant of the (modular) P/T net of Figure 8.1(b) that models a three-place buffer. Definition 6.10 defines the algebraic semantics of such a P/T net as a closed term in the theory $(\mathrm{ACP}^*_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(U, L), \gamma)$ where U and L are as defined in Section 4 and where $\gamma : AC(U, L) \times AC(U, L) \to AC(U, L)$ is a parameter that can be used to define a step semantics or a total-order semantics. The standard choice $\uplus$ for this parameter yields a step semantics. In the algebraic semantics of Definition 6.10, the information about the consumption and production of transitions is already included in the actions. However, in the current setting, cause-addition operators can be used to make this information explicit. The step semantics of the P/T net of Figure 9.5(a) can also be formalized by the following closed $(\mathrm{ACP}^{*c}_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(U, L), \uplus)$ term. As before, a triple $(\mathbf{0}, \alpha, \mathbf{0})$ in $AC(U, L)$ is identified with its second element and square brackets of singleton bags are omitted.

$$\rho_{\boldsymbol{ca}(U)}(\lambda^U_{[1^3]}({}^1r^2 * \delta \parallel {}^2s^1 * \delta)).$$

Let X_i, where $0 \le i \le 3$, denote the closed term $\rho_{\boldsymbol{ca}(U)}(\lambda^U_{[1^{3-i}, 2^i]}({}^1r^2 * \delta \parallel {}^2s^1 * \delta))$. Thus, X_0 equals the algebraic semantics of the P/T net of Figure 9.5(a). The following results can be obtained.

$$\begin{aligned}
X_0 &= r \cdot X_1, \\
X_1 &= r \cdot X_2 + s \cdot X_0 + [r, s] \cdot X_1, \\
X_2 &= r \cdot X_3 + s \cdot X_1 + [r, s] \cdot X_2, \text{ and} \\
X_3 &= s \cdot X_2.
\end{aligned}$$

Figure 8.4 visualizes the semantics of X_0 when ignoring consumptions and productions.

Example 9.7. Figure 9.5(b) shows another P/T-net model of a three-place buffer. It can be obtained by combining three one-place buffers by means of transition fusion. As a result, the internal communication actions c_2 and c_3 are explicit in the model. For the P/T net of Figure 9.5(b), cause-addition operators can be used to define the following alternative algebraic semantics:

$$\rho_{\boldsymbol{ca}(U)}(\lambda^U_{[1,3,5]}({}^1r^2 * \delta \parallel {}^2({}^3c_2{}^4)^1 * \delta \parallel {}^5({}^4c_3{}^3)^6 * \delta \parallel {}^6s^5 * \delta)).$$

Let X_{ijk}, where $0 \le i, j, k \le 1$, denote the closed term

$$\rho_{\boldsymbol{ca}(U)}(\lambda^U_{[1^{1-i}, 2^i, 3^{1-j}, 4^j, 5^{1-k}, 6^k]}({}^1r^2 * \delta \parallel {}^2({}^3c_2{}^4)^1 * \delta \parallel {}^5({}^4c_3{}^3)^6 * \delta \parallel {}^6s^5 * \delta)).$$

The following results can be obtained from the axioms of $(\mathrm{ACP}^{*c}_\lambda + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(U, L), \uplus)$.

$$\begin{aligned}
X_{000} &= r \cdot X_{100}, \\
X_{100} &= c_2 \cdot X_{010}, \\
X_{010} &= r \cdot X_{110} + c_3 \cdot X_{001} + [r, c_3] \cdot X_{101}, \\
X_{110} &= c_3 \cdot X_{101}, \\
X_{001} &= r \cdot X_{101} + s \cdot X_{000} + [r, s] \cdot X_{100}, \\
X_{101} &= c_2 \cdot X_{011} + s \cdot X_{100} + [c_2, s] \cdot X_{010}, \\
X_{011} &= r \cdot X_{111} + s \cdot X_{010} + [r, s] \cdot X_{110}, \text{ and} \\
X_{111} &= s \cdot X_{110}.
\end{aligned}$$

Figure 8.8 visualizes the semantics of term X_{000} when assuming that r_1, s_2, r_2, and s_3 are replaced by r, c_2, c_3, and s, respectively.

As explained in the introduction to this section, cause addition is related to sequential composition.

Example 9.8. Consider again the P/T-net model of Figure 9.5(b). The ordering between transitions r and c_2 is enforced by places 1 and 2. In the algebraic semantics of this P/T net given in the previous example, these two places translate to cause-addition operators. It is possible to replace these cause-addition operators with a sequential composition. Similarly, it is possible to replace the cause-addition operators corresponding to places 5 and 6. The result of these replacements is an alternative closed term in $\mathcal{C}(U, L)$ defining the behavior of a three-place buffer, namely the term

$$\rho_{\boldsymbol{ca}(U)}(\lambda_3^U(X \parallel Y)),$$

where X and Y are abbreviations for the closed terms

$$(r \cdot {}^3c_2{}^4) * \delta$$

and

$$({}^4c_3{}^3 \cdot s) * \delta.$$

In the P/T-net model of Figure 9.5(b), the above replacements correspond to *internalizing* the places 1, 2, 5, and 6. The result is a *modular* P/T net with a structure that is very similar to the structure of the modular P/T net of Figure 8.5(b).

It remains to substantiate our claim that term $\rho_{\boldsymbol{ca}(U)}(\lambda_3^U(X \parallel Y))$ is an alternative expression for the three-place buffer of the previous example. Assuming that X_{000} is an abbreviation for the term $\rho_{\boldsymbol{ca}(U)}(\lambda_3^U(X \parallel Y))$ and assuming that X_{100} through X_{111} are defined as below, exactly the same equations can be derived from the axioms of $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + \mathit{RSP}^* + \mathrm{SC})(AC(U, L), \uplus)$ as the equations for the corresponding terms in Example 9.7. As a consequence, term X_{000} of this example has the same semantics as the term X_{000} of Example 9.7.

$$\begin{aligned}
X_{100} &= \rho_{\boldsymbol{ca}(U)}(\lambda_3^U(c_2 \cdot X \parallel Y)),\\
X_{010} &= \rho_{\boldsymbol{ca}(U)}(\lambda_4^U(X \parallel Y)),\\
X_{110} &= \rho_{\boldsymbol{ca}(U)}(\lambda_4^U(c_2 \cdot X \parallel Y)),\\
X_{001} &= \rho_{\boldsymbol{ca}(U)}(\lambda_3^U(X \parallel s \cdot Y)),\\
X_{101} &= \rho_{\boldsymbol{ca}(U)}(\lambda_3^U(c_2 \cdot X \parallel s \cdot Y)),\\
X_{011} &= \rho_{\boldsymbol{ca}(U)}(\lambda_4^U(X \parallel s \cdot Y)), \text{ and}\\
X_{111} &= \rho_{\boldsymbol{ca}(U)}(\lambda_4^U(c_2 \cdot X \parallel s \cdot Y)).
\end{aligned}$$

Note the structural similarity between the terms X_{000} through X_{111} as defined in this example and those defined in Example 8.6. Another interesting observation is that the term X_{000} of Example 9.7 has three causes in the marking of the state operator and six different cause-addition operators. Term X_{000} of this example has two causes and four cause-addition operators fewer, but it has two sequential-composition operators instead (in the auxiliary terms X and Y). A final observation is that it appears to be impossible to derive the equality of the terms X_{000} of this example and X_{000} of Example 9.7 from the axioms of the equational theory $(\mathrm{ACP}^{*c}_{\lambda}+\mathrm{RN}+\mathit{RSP}^*+\mathrm{SC})(AC(U, L), \uplus)$. Since these two terms have the same semantics in the model of the previous subsection, this observation means that the equational theory is not complete for this model. It appears that the generalization of RSP^* to the general recursive specification principle RSP, as defined in for example [5, 6], is necessary to obtain the desired equality.

Examples 9.7 and 9.8 show that cause-addition operators and sequential-composition operators can replace each other. The relation between cause addition and sequential composition is studied in more detail in the next subsection.

9.4 Cause addition and sequential composition

The most straightforward way to specify a causal ordering between two processes in a process-algebraic setting is the sequential-composition operator. In an equational theory with causal state operators and cause addition, causes provide an alternative. In this subsection, it is shown that cause addition and sequential composition are exchangeable notions. The main result is a theorem stating that any sequential-composition operator in a process term can be replaced by a pair of cause-addition operators. This theorem can be seen as a formalization of the statement by Milner in [44] that sequential composition is not necessary as a basic operator in a process-algebraic theory.

The main objective of this subsection is to illustrate the *conceptual* relation between cause addition and sequential composition. Therefore, to facilitate reasoning, we consider an equational theory without iteration. This simplification means that it is only possible to specify processes with bounded behavior. Assume that C is a set of causes and A a set of atomic actions. Let $\gamma : AC(C, A) \times AC(C, A) \to AC(C, A)$ be some communication function. The equational theory considered in this subsection is the theory $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma)$. The basis of this equational theory is the theory $(\mathrm{ACP}+\mathrm{RN})(AC(C, A), \gamma)$ of Table 5.3. This basic theory is extended with the causal state operator of Section 6.1 and cause addition of Section 9.1, yielding the Algebra of Communicating Processes with cause addition, causal state operator, and renaming. Let $\mathcal{C}(C, A)$ be the set of closed $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma)$ terms.

It is possible to construct a model of the equational theory $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma)$ following the same approach as in earlier sections. However, all the results in this subsection are proven in terms of the equational theory without referring to a particular model. Therefore, the construction of a model is left to the reader.

The main advantage of restricting our attention to processes with bounded behavior is that each process can be expressed in terms of the inaction constant, actions, choice, and action prefix. The latter is a restricted form of sequential composition (also present in, for example, CCS [43, 44]) where the left operand is required to be an action. A closed process term that contains only the aforementioned constants and operators is called a *basic* term. Formally, the set of basic terms is defined as follows.

Definition 9.9. (Basic terms) The set of *basic* terms over the signature of $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma)$, denoted $\mathcal{B}(C, A)$, is inductively defined as follows. The inaction constant is an element of $\mathcal{B}(C, A)$. The set of actions $AC(C, A)$ is contained in $\mathcal{B}(C, A)$. Furthermore, for any $a \in AC(C, A)$ and basic terms $s, t \in \mathcal{B}(C, A)$, also $a \cdot t$ and $s + t$ are elements of $\mathcal{B}(C, A)$.

Note the similarity between basic terms and terms in head normal form, as defined in Definition 5.9. Clearly, any basic term is in head normal form. The converse is not true.

The following theorem states that any closed process term is derivably equal to a basic term.

Theorem 9.10. (Elimination) *For any closed term $p \in \mathcal{C}(C, A)$, there exists a basic term $t \in \mathcal{B}(C, A)$ such that $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma) \vdash p = t$.*

Proof. The proof uses the strategy based on term-rewriting techniques explained in [5]. Consider the rewriting system obtained when interpreting Axioms $A3$ through $A7$, $C3$, $CF1$ and $CF2$, and $CM1$ through $CM9$ of Table 5.1, $RN1$ through $RN3$ of Table 5.3, $CSO1$ through $CSO5$ of Table 6.1, and $IC1$ through $IC4$ and $OC1$ through $OC4$ of Table 9.1 as rewrite rules from left to right. Assuming that this rewriting system is strongly normalizing, it is not difficult to see that each closed term in $\mathcal{C}(C, A)$ has a normal form that is a derivably-equivalent basic term.

To prove that the rewriting system is strongly normalizing, the so-called method of the recursive path ordering can be applied. The basis of the proof is a well-founded ordering on the constants and operators of $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma)$. The required ordering is inspired by very similar orderings that are given in [5]. To define the ordering, it is necessary to *rank* the causal state operators, the merge, the left merge, and the communication merge as explained in [5]. For any $I \subseteq C$, $m \in \mathcal{B}(I)$, and $n \in \mathbb{N}$ with $n > 0$, $\lambda^I_{m,n}$ denotes the causal state operator λ^I_m with rank n; for any $n \in \mathbb{N}$ with $n > 1$, $\|_n$, $⌊⌊_n$, and $|_n$ denote the merge, left merge, and communication merge with rank n, respectively. The required ordering can now be defined as the transitive closure of the relation $<$ specified as follows: $+ < \cdot$; for any $n \in \mathbb{N}$ with $n > 1$, $\cdot < ⌊⌊_n < \|_n < ⌊⌊_{n+1}$ and $\cdot < |_n < \|_n < |_{n+1}$; for all $a \in AC(C, A)$, $\delta < a$ and $a < |_2$; for all $\boldsymbol{f} \in RF$, $\cdot < \rho_{\boldsymbol{f}}$ and, for all $a \in AC(C, A)$, $a < \rho_{\boldsymbol{f}}$; for all $c \in C$, $\cdot < {}^c\_$, $\cdot < \_^c$, and, for all $a \in AC(C, A)$, $a < {}^c\_$ and $a < \_^c$; for all $I \subseteq C$ and $m \in \mathcal{B}(I)$, $\cdot < \lambda^I_{m,1}$ and, for all $a \in AC(C, A)$, $a < \lambda^I_{m,1}$; finally, for all $I \subseteq C$, $m \in \mathcal{B}(I)$, and $n \in \mathbb{N}$ with $n > 0$, $\lambda^I_{m,n} < \lambda^I_{m,n+1}$. Given this ordering, the details of the proof are straightforward and, hence, omitted. □

Note that Theorem 9.10 implies that any cause-addition operator in a closed term can be eliminated. Thus, cause addition does not add expressive power to an equational theory with choice and sequential composition. However, it is well known that in an interleaving framework choice and sequential composition are sufficient to specify processes with bounded behavior. A more interesting result is the fact that an equational theory with parallel composition, cause addition, causal state operators, and cause abstraction is sufficiently expressive to eliminate sequential composition.

Example 9.11. Let r and s be two actions in $AC(C, A)$. Consider the simple sequential process $r \cdot s$. This example shows how to eliminate the occurrence of the sequential-composition operator in this simple process. The idea is to replace the sequential-composition operator with a parallel-composition operator while using two additional causes and a causal state operator to restrict the parallel composition in such a way that it behaves as a sequential composition. Cause abstraction is used to hide the new causes and, thus, to obtain the original sequential process. It is assumed that the communication function equals $\uplus$. Let c be a cause in C that does not occur in the consumption or production of r and s. Consider the term $\rho_{\boldsymbol{ca}(\{c\})}(\lambda^{\{c\}}_{\boldsymbol{0}}(r^c \parallel {}^c s))$. It specifies that action r produces an extra output cause c, whereas s requires an extra input cause c. As a consequence, the state operator $\lambda^{\{c\}}_{\boldsymbol{0}}$ enforces a sequential ordering between r and s:

$$
\begin{aligned}
\rho_{\boldsymbol{ca}(\{c\})}(\lambda^{\{c\}}_{\boldsymbol{0}}(r^c \parallel {}^c s)) &= \rho_{\boldsymbol{ca}(\{c\})}(\lambda^{\{c\}}_{\boldsymbol{0}}\ (\ (\boldsymbol{cr}, \boldsymbol{sr}, \boldsymbol{pr} \uplus [c]) \cdot (\boldsymbol{cs} \uplus [c], \boldsymbol{ss}, \boldsymbol{ps}) \\
&\qquad + (\boldsymbol{cs} \uplus [c], \boldsymbol{ss}, \boldsymbol{ps}) \cdot (\boldsymbol{cr}, \boldsymbol{sr}, \boldsymbol{pr} \uplus [c]) \\
&\qquad + (\boldsymbol{cr} \uplus \boldsymbol{cs} \uplus [c], \boldsymbol{sr} \uplus \boldsymbol{ss}, \boldsymbol{pr} \uplus \boldsymbol{ps} \uplus [c]) \\
&\qquad) \\
&= r \cdot \rho_{\boldsymbol{ca}(\{c\})}(\lambda^{\{c\}}_{[c]}((\boldsymbol{cs} \uplus [c], \boldsymbol{ss}, \boldsymbol{ps}))) + \delta + \delta \\
&= r \cdot s.
\end{aligned}
$$

Note that two assumptions are essential in the above derivation. First, cause c must be a new cause not already occurring in the consumption or production of r and s. If this assumption is not satisfied, the causal state operator $\lambda^{\{c\}}_{\boldsymbol{0}}$ might disturb the desired behavior. Second, the communication function may not specify any actual communications that affect the new cause c, because such communications can also disturb the desired result. The assumption that the communication function equals $\uplus$, which means that a partial-order framework is obtained, satisfies this restriction. Also any communication function that can be derived from a communication function on atomic actions, as explained in Section 7, is an allowed choice. Yet another possibility is to choose the communication function that is undefined for any pair of actions, which implies a total-order setting.

The remainder of this subsection is devoted to proving that the approach to eliminating a sequential-composition operator illustrated in Example 9.11 can be generalized to arbitrary closed terms in $\mathcal{C}(C, A)$.

The reason for introducing the notion of a basic term and proving the elimination result of Theorem 9.10 is not only to illustrate that cause addition can be eliminated from closed terms. An important consequence of Theorem 9.10 is that properties formulated in terms of closed terms can be rephrased in terms of basic terms without losing generality. Properties on basic terms can often be proved by means of *structural* *induction*, which is the technique also used in Section 5. Structural induction is also used to prove the main result of this subsection.

In reasoning about cause addition, it is convenient to know the set of causes that occur in a closed term. For this purpose, the *causes function* is introduced. The causes function yields for each closed term in $\mathcal{C}(C, A)$ the set of causes that the corresponding process in the model obtained by following the approach of Section 5.2 consumes or produces during any of its execution paths. The causes function is defined inductively using the structure of basic terms, under the assumption that two terms that are derivably equal have the same set of causes. In combination with the elimination result of Theorem 9.10, this assumption means that it is possible to calculate the causes of arbitrary closed terms in $\mathcal{C}(C, A)$.

Definition 9.12. (Causes function) The causes function $causes : \mathcal{C}(C, A) \to \mathcal{P}(C)$ is a function such that, for any closed terms p and q in $\mathcal{C}(C, A)$, $((\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma) \vdash p = q) \Rightarrow causes(p) = causes(q)$. For any $a \in AC(C, A)$ and $p, q \in \mathcal{C}(C, A)$, $causes(\delta) = \emptyset$, $causes(a) = \{c \mid c \in \boldsymbol{ca} \uplus \boldsymbol{pa}\}$, $causes(a \cdot p) = causes(a) \cup causes(p)$, $causes(p + q) = causes(p) \cup causes(q)$.

Note that it should be verified that the definition of the causes function is consistent with the axioms of set theory. Inconsistencies arise when the combination of the requirement that derivably equal terms have the same set of causes and the inductive definition allows the derivation of an equality between sets that is not derivable from set theory. It is beyond the scope of this chapter to prove that Definition 9.12 is consistent with set theory.

The definitions given so far are sufficient to formalize the main result of this subsection, namely the elimination of sequential composition by means of cause addition. The proof uses a number of auxiliary properties that are given below. Note that, for the sake of simplicity, the communication function is assumed to be $\uplus$. However, any communication function that does not affect causes is allowed.

Theorem 9.13. (Elimination of sequential composition) *Let p, q be closed terms in $\mathcal{C}(C, A)$ and c a cause in C; let $I = \{c\}$.*

$c \notin causes(p) \cup causes(q) \Rightarrow$
$(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \uplus) \vdash p \cdot q = \rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}(p^c \mathbin{⌊\!⌊} {}^c q)) = \rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}(p^c \parallel {}^c q))$.

Proof. The proof is by induction on the structure of basic terms. Recall that $\equiv$ denotes structural equivalence of terms. It follows from Theorem 9.10 that there exists a basic term $t \in \mathcal{B}(C, A)$ such that $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \uplus) \vdash p = t$. Thus, it suffices to show that $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \uplus) \vdash t \cdot q = \rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}(t^c \mathbin{⌊\!⌊} {}^c q)) = \rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}(t^c \parallel {}^c q))$. It follows from Definition 9.12 (Causes function) that $causes(t) = causes(p)$ and, thus, that $c \notin causes(t)$.

i) Assume that $t \equiv \delta$. It is straightforward to prove from the axioms of $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \uplus)$ that

$$\rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}(\delta^c \mathbin{⌊\!⌊} {}^c q)) \overset{OC1}{=} \rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}(\delta \mathbin{⌊\!⌊} {}^c q)) \overset{CM2,A7}{=} \rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}(\delta)) \overset{CSO1,RN1}{=} \delta \overset{A7,t\equiv\delta}{=} t \cdot q,$$

which completes the first part of the proof for the case that $t \equiv \delta$. In addition, using the above result, the following derivation can be obtained.

$$\rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}(\delta^c \parallel {}^c q)) \overset{CM1,OC1}{=} \rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}(\delta^c \mathbin{⌊\!⌊} {}^c q + {}^c q \mathbin{⌊\!⌊} \delta + \delta \mid {}^c q)) \overset{CSO5,RN2}{=}$$
$$\rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}(\delta^c \mathbin{⌊\!⌊} {}^c q)) + \rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}({}^c q \mathbin{⌊\!⌊} \delta)) + \rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}(\delta \mid {}^c q)) \overset{9.14,9.15,RN1}{=}$$
$$\rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}(\delta^c \mathbin{⌊\!⌊} {}^c q)) + \delta + \delta \overset{A6}{=} t \cdot q,$$

which completes the proof for the case $t \equiv \delta$.

ii) Assume that $t \equiv a$, for some action $a \in AC(C, A)$. The second step in the following derivation uses the fact that $c \notin causes(t)$; the third step uses that $c \notin causes(q)$.

$$\rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}(a^c \mathbin{⌊\!⌊} {}^c q)) \overset{OC2,CM2}{=} \rho_{\boldsymbol{ca}(I)}(\lambda^I_{\boldsymbol{0}}((\boldsymbol{ca}, sa, \boldsymbol{pa} \uplus [c]) \cdot {}^c q)) \overset{CSO4,CSO2,RN3,RN1}{=}$$
$$a \cdot \rho_{\boldsymbol{ca}(I)}(\lambda^I_{[c]}({}^c q)) \overset{9.17,t\equiv a}{=} t \cdot q.$$

Using this result, the following derivation can be obtained.

$\rho_{ca(I)}(\lambda^I_{\mathbf{0}}(a^c \parallel {}^c q)) \overset{CM1}{=} \rho_{ca(I)}(\lambda^I_{\mathbf{0}}(a^c \mathbin{⌊\!⌊} {}^c q + {}^c q \mathbin{⌊\!⌊} a^c + a^c \mid {}^c q)) \overset{CSO5,RN2}{=}$
$\rho_{ca(I)}(\lambda^I_{\mathbf{0}}(a^c \mathbin{⌊\!⌊} {}^c q)) + \rho_{ca(I)}(\lambda^I_{\mathbf{0}}({}^c q \mathbin{⌊\!⌊} a^c)) + \rho_{ca(I)}(\lambda^I_{\mathbf{0}}(a^c \mid {}^c q)) \overset{9.14,9.15,RN1,A6}{=} t \cdot q,$

which completes the proof for this case.

iii) Assume that $t \equiv a \cdot s$, for some action $a \in AC(C, A)$ and basic term $s \in \mathcal{B}(C, A)$. The second step in the following derivation uses the fact that $c \notin causes(t)$;

$\rho_{ca(I)}(\lambda^I_{\mathbf{0}}((a \cdot s)^c \mathbin{⌊\!⌊} {}^c q)) \overset{OC4,CM3}{=} \rho_{ca(I)}(\lambda^I_{\mathbf{0}}(a \cdot (s^c \parallel {}^c q))) \overset{CSO4,CSO2,RN3,RN1}{=}$
$a \cdot \rho_{ca(I)}(\lambda^I_{\mathbf{0}}(s^c \parallel {}^c q)) \overset{\text{Induction}}{=} a \cdot s \cdot q \overset{t \equiv a \cdot s}{=} t \cdot q.$

Using this result, the following derivation can be made.

$\rho_{ca(I)}(\lambda^I_{\mathbf{0}}((a \cdot s)^c \parallel {}^c q)) \overset{CM1,CSO5,RN2}{=} \rho_{ca(I)}(\lambda^I_{\mathbf{0}}((a \cdot s)^c \mathbin{⌊\!⌊} {}^c q)) + \rho_{ca(I)}(\lambda^I_{\mathbf{0}}({}^c q \mathbin{⌊\!⌊} (a \cdot s)^c))$
$+ \rho_{ca(I)}(\lambda^I_{\mathbf{0}}((a \cdot s)^c \mid {}^c q)) \overset{9.14,9.15,RN1,A6}{=} t \cdot q.$

iv) Assume that $t \equiv u + v$, for basic terms $u, v \in \mathcal{B}(C, A)$.

$\rho_{ca(I)}(\lambda^I_{\mathbf{0}}((u + v)^c \mathbin{⌊\!⌊} {}^c q)) \overset{OC3,CM4}{=} \rho_{ca(I)}(\lambda^I_{\mathbf{0}}(u^c \mathbin{⌊\!⌊} {}^c q + v^c \mathbin{⌊\!⌊} {}^c q)) \overset{CSO5,RN2}{=}$
$\rho_{ca(I)}(\lambda^I_{\mathbf{0}}(u^c \mathbin{⌊\!⌊} {}^c q)) + \rho_{ca(I)}(\lambda^I_{\mathbf{0}}(u^c \mathbin{⌊\!⌊} {}^c q)) \overset{\text{Induction}(2\times)}{=} u \cdot q + v \cdot q \overset{A4, t \equiv u+v}{=} t \cdot q.$

In addition,

$\rho_{ca(I)}(\lambda^I_{\mathbf{0}}((u + v)^c \parallel {}^c q)) \overset{CM1,CSO5,RN2}{=} \rho_{ca(I)}(\lambda^I_{\mathbf{0}}((u + v)^c \mathbin{⌊\!⌊} {}^c q)) + \rho_{ca(I)}(\lambda^I_{\mathbf{0}}({}^c q \mathbin{⌊\!⌊} (u + v)^c))$
$+ \rho_{ca(I)}(\lambda^I_{\mathbf{0}}((u + v)^c \mid {}^c q)) \overset{9.14,9.15,RN1,A6}{=} t \cdot q,$

which completes the proof. □

The above proof uses a number of auxiliary properties that are given below. Example 9.11 can be used to better understand these properties. All the properties are proven by means of structural induction. However, since the proofs are much simpler than the proof of Theorem 9.13, the details are omitted. Note that three of the four properties can be proven for arbitrary communication functions. Only for Property 9.15, it is required that the communication function does not affect causes.

Property 9.14. *Let p, q be closed terms in $\mathcal{C}(C, A)$, $I \subseteq C$ a set of causes, and c a cause in I.*
$(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma) \vdash \lambda^I_{\mathbf{0}}({}^c p \mathbin{⌊\!⌊} q) = \delta.$

Proof. It follows from Theorem 9.10 that there exists a basic term $t \in \mathcal{B}(C, A)$ such that $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma) \vdash p = t$. It suffices to show that $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma) \vdash \lambda^I_{\mathbf{0}}({}^c t \mathbin{⌊\!⌊} q) = \delta$. The proof is by induction on the structure of basic term t. □

Property 9.15. *Let p, q be closed terms in $\mathcal{C}(C, A)$, $I \subseteq C$ a set of causes, and c a cause in I.*
$(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \uplus) \vdash \lambda^I_{\mathbf{0}}(p \mid {}^c q) = \delta.$

Proof. It follows from Theorem 9.10 that there exist basic terms $s, t \in \mathcal{B}(C, A)$ such that $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \uplus) \vdash p = s \wedge q = t$. It suffices to show that $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \uplus) \vdash \lambda^I_{\mathbf{0}}(s \mid {}^c t) = \delta$. The proof is by induction on the structure of basic term s. The first three cases are proven by induction on the structure of basic term t. The proofs of the second and third case use the fact that the communication function equals $\uplus$. □

Property 9.16. *Let p be a closed term in $\mathcal{C}(C, A)$, $I \subseteq C$ a set of causes, and m a bag of internal causes in $\mathcal{B}(I)$.*
$causes(p) \cap I = \emptyset \;\Rightarrow\; (\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma) \vdash \rho_{ca(I)}(\lambda^I_m(p)) = p.$

Proof. It follows from Theorem 9.10 that there exists a basic term $t \in \mathcal{B}(C, A)$ such that $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma) \vdash p = t$. It suffices to show that $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma) \vdash \rho_{\boldsymbol{ca}(I)}(\lambda^I_m(t)) = t$. The proof is by induction on the structure of basic term t. □

Property 9.17. *Let p be a closed term in $\mathcal{C}(C, A)$, $I \subseteq C$ a set of causes, and c a cause in I.*
$causes(p) \cap I = \emptyset \Rightarrow (\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma) \vdash \rho_{\boldsymbol{ca}(I)}(\lambda^I_{[c]}({}^c p)) = p.$

Proof. It follows from Theorem 9.10 that there exists a basic term $t \in \mathcal{B}(C, A)$ such that $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma) \vdash p = t$. It suffices to show that $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma) \vdash \rho_{\boldsymbol{ca}(I)}(\lambda^I_{[c]}({}^c t)) = t$. The proof is by induction on the structure of basic term t. The proof of the third case uses Property 9.16. □

9.5 Concluding remarks

In this section, it has been shown how cause-addition operators can be used to specify causal orderings between actions in an algebraic expression. The cause-addition operators are inspired by the causality mechanism known from Petri nets. In labeled P/T nets as defined in Section 4, causal relationships are enforced via places. An occurrence of a specific (input or output) cause-addition operator in an algebraic expression corresponds to a place and an accompanying (input or output) arc in a labeled P/T net. In the algebraic framework of Section 6, information about input and output causes of steps is implicit in actions. Cause addition can be used to make causality information explicit. It can be used to adapt the algebraic semantics of labeled P/T nets, as defined in Definition 6.10, accordingly.

Theorem 9.13 shows that the combination of the causal state operator and cause addition can replace sequential composition. It substantiates the statement by Milner in [44] that sequential composition is not necessarily needed as a primitive operator in a process-algebraic theory. Note that Theorem 9.13 is formulated for processes with bounded behavior only. However, we claim that it can be generalized to a setting with iteration and even to a framework allowing general recursion.

10 A Non-Interleaving Process Algebra

As mentioned before, Petri-net theory is one of the most well-known examples of a partial-order theory for specifying and analyzing concurrent systems. In Section 5, it has been shown how standard ACP-style process algebra can be turned into a partial-order framework. In Sections 6 and 9, two extensions of the standard process-algebraic framework have been introduced that are inspired by concepts from Petri-net theory, namely the causal state operator and cause addition. Thus, it is possible to give algebraic specifications in the style of Petri-net theory. The basis of such a specification is a parallel composition of component specifications. Causal relationships are specified by means of cause-addition operators. A causal state operator is used to restrict the parallel composition with respect to a specific initial marking of causes and, thus, to enforce an actual ordering of actions.

However, several aspects of the algebraic framework developed so far have no equivalent in Petri-net theory. First, as already mentioned in Section 4, standard Petri-net theory does not distinguish between successful termination and deadlock, whereas such a distinction is made in ACP-style process algebra. Second, the communication-merge operator has no straightforward interpretation in the Petri-net framework of Section 4. Consider, for example, the specification of process C in Example 5.35. It is not possible to specify this process by means of a labeled P/T net as defined in Definition 4.1. Finally, all the equational theories and their models that have been considered so far are interleaving theories and algebras, which means that each parallel composition can be written as a so-called head normal form. The notion of a head normal form is another notion that does not have a straightforward counterpart in Petri-net theory.

In Section 10.1, we develop an algebra of so-called non-terminating serializable processes. The domain of this algebra contains only processes that cannot terminate successfully. Moreover, the signature of this algebra does not contain a communication merge. It turns out that this algebra of non-terminating serializable processes is a non-interleaving process algebra. Thus, the algebra has several important characteristics of a Petri-net framework. Section 10.2 contains a brief discussion on causality in process algebra, in particular, non-interleaving process algebra. In Section 10.3, it is shown that all non-terminating serializable processes with *bounded* behavior can be specified without using choice, sequential composition, or communication. Thus, in the context of non-terminating serializable processes with bounded behavior, the causality mechanism borrowed from Petri-net theory in the form of the causal state operator and cause addition is sufficiently powerful to replace the standard algebraic operators and, thus, the standard causality mechanism in ACP-style process algebra.

10.1 The algebra of non-terminating serializable processes

The basis of the algebra of non-terminating serializable processes is the process algebra $\mathcal{M}(C, A, \gamma)$ of Section 9.2, where C is a set of causes, A a set of atomic actions, and γ a communication function on the set of actions $AC(C, A) = \mathcal{B}(C) \times \mathcal{B}(A) \times \mathcal{B}(C)$. In order to obtain a partial-order theory, it is assumed that the communication function equals $\uplus$. The goal is to restrict the domain of $\mathcal{M}(C, A, \uplus)$ in such a way that the resulting set of processes contains only processes that have an intuitive interpretation in terms of labeled P/T nets.

The notion of a non-terminating serializable process is defined in the context of a process space as defined in Definition 3.1. Therefore, let $(\mathcal{P}, \mathcal{A}, \longrightarrow, \downarrow)$ be some process space. Let $\overset{*}{\Longrightarrow} \subseteq \mathcal{P} \times \mathcal{P}$ be the reachability relation in this process space as defined in Definition 3.2. A process is said to be non-terminating if and only if it cannot terminate successfully. That is, a process is non-terminating if and only if it terminates in a deadlock or it does not terminate because its behavior is unbounded.

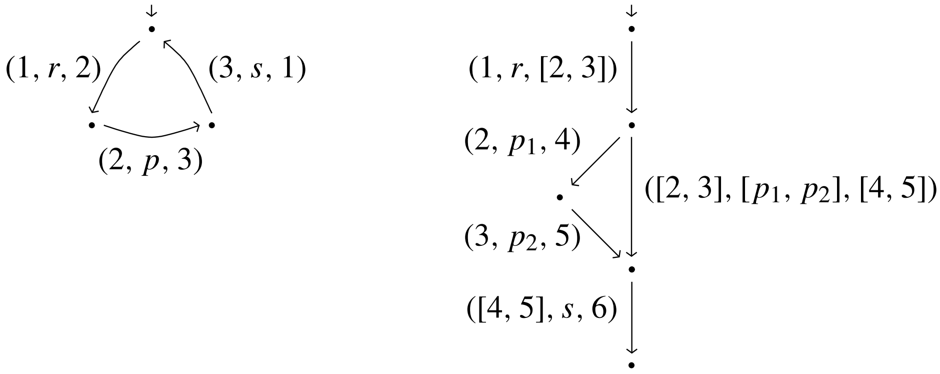
Definition 10.1. (Non-terminating process) Let p be a process in $\mathcal{P}$. Process p has the option to terminate, denoted $\Downarrow p$, if and only if there is a process p' in $\mathcal{P}$ such that $p \overset{*}{\Longrightarrow} p'$ and $\downarrow p'$. Process p is *non-terminating* if and only if $\neg \Downarrow p$.

The notion of *serializability* is only meaningful if processes can perform some kind of *steps*. Therefore, assume that the set of actions in the process space $\mathcal{A}$ equals $AC(C, A)$. The basic idea of serializability is as follows. Assume that a process can perform a step that consists of several atomic actions. The process is serializable only if it can perform all the atomic actions in the step in isolation and in any order. The bags of input and output causes of the step may be distributed over the atomic actions in some arbitrary way.

Definition 10.2. (Serializable process) Let p be a process in $\mathcal{P}$. Process p is *serializable* if and only if, for any $p', p'' \in \mathcal{P}$, $a \in AC(C, A)$, and $\alpha_1, \alpha_2 \in \mathcal{B}(A) \setminus \{\mathbf{0}\}$ such that $p \overset{*}{\Longrightarrow} p'$ and $sa = \alpha_1 \uplus \alpha_2$,

$$
\begin{aligned}
p' \overset{a}{\longrightarrow} p'' \ \Rightarrow\ (\exists\, p''', a_1, a_2 : p''' \in \mathcal{P} \wedge a_1, a_2 \in AC(C, A) \wedge sa_1 = \alpha_1 \wedge sa_2 = \alpha_2 \wedge a = a_1 \uplus a_2 : \\
p' \overset{a_1}{\longrightarrow} p''' \overset{a_2}{\longrightarrow} p'' \\
).
\end{aligned}
$$

Example 10.4. Let p, p_1, p_2, r, and s be atomic actions in A; let $1, 2, 3, 4, 5,$ and 6 be causes in C. Figure 10.3 shows two simple processes (that are variants of processes depicted in Figures 3.4 and 3.6). Square brackets of singleton bags are omitted. Clearly, both processes are non-terminating. The process shown in Figure 10.3(a) corresponds to an unbounded iteration, whereas the process shown in Figure 10.3(b) terminates in a deadlock. Process (a) is serializable for the simple reason that it cannot perform a step consisting of more than one atomic action. Process (b) is not serializable. After its initial action, it can



(a) (b)

Figure 10.3: Examples of a non-terminating serializable process (a) and a non-terminating process that is not serializable (b).

perform a step $[p_1, p_2]$ or it can perform the sequence of the two steps $[p_1]$ and $[p_2]$. However, it cannot perform the sequence $[p_2]$ followed by $[p_1]$.

In the introduction to this section, it is implicitly claimed that labeled P/T nets as defined in Section 4 correspond to non-terminating serializable processes. Note that Definition 10.2 (Serializability) can be easily adapted to a setting where actions do not contain causes. Consider the step semantics of marked, labeled P/T nets defined in Definition 4.15. Clearly, marked, labeled P/T nets cannot terminate successfully. In addition, it follows from Definition 4.14 (Step firing rule) that all marked, labeled P/T nets define serializable processes. If several transitions in a labeled P/T net are enabled simultaneously, then each of these transitions is also enabled separately and firing one of these transitions does not disable any of the other transitions.

Example 10.5. Consider the marked, labeled P/T net of Figure 4.2 and its step semantics discussed in Example 4.16. It is not difficult to see that the P/T net of Figure 4.2 defines a serializable process. It is interesting to compare this process to the process depicted in Figure 10.3(b).

Let us return to the process algebra $\mathcal{M}(C, A, \uplus)$. This algebra is a model of the equational theory $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC})(AC(C, A), \uplus)$ of Section 9.1. Let $\mathcal{C}(C, A)$ be the set of closed $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC})(AC(C, A), \uplus)$ terms. Let $(\mathcal{C}(C, A) \cup \{\surd\}, AC(C, A), \longrightarrow, \{\surd\})$ be the process space defined in Section 9.2, underlying the algebra $\mathcal{M}(C, A, \uplus)$, with reachability relation $\stackrel{*}{\Longrightarrow} \subseteq (\mathcal{C}(C, A) \cup \{\surd\}) \times (\mathcal{C}(C, A) \cup \{\surd\})$. The following result is a direct corollary of Definition 10.1.

Corollary 10.6. *For any closed term p in $\mathcal{C}(C, A)$, $\Downarrow p \Leftrightarrow p \stackrel{*}{\Longrightarrow} \surd$.*

As required by Definition 10.2, the set of actions in the process space $(\mathcal{C}(C, A) \cup \{\surd\}, AC(C, A), \longrightarrow, \{\surd\})$ corresponds to $AC(C, A)$. The set of all non-terminating serializable processes in this process space is denoted $\mathcal{C}_{\mathrm{ns}}(C, A)$.

Example 10.7. Consider again Example 10.4. The process depicted in Figure 10.3(a) is specified by the term $({}^1r^2 \cdot {}^2p^3 \cdot {}^3s^1)^{*}\delta$, where a triple $(\mathbf{0}, \alpha, \mathbf{0}) \in AC(C, A)$ is identified with its second element and square brackets of singleton bags are omitted. Another term representing the same process is $\lambda_1^{\{1,2,3\}}({}^1r^2 {*} \delta \parallel {}^2p^3 {*} \delta \parallel {}^3s^1 {*} \delta)$. The process of Figure 10.3(b) is specified by, for example, term $({}^1r^2)^3 \cdot ({}^2p_1{}^4 \cdot {}^3p_2{}^5 + {}^2p_1{}^4 \mid {}^3p_2{}^5) \cdot {}^4({}^5s^6) \cdot \delta$.

Another example of a non-terminating serializable process is the term $({}^1r^2)^3 \cdot ({}^2p_1{}^4 \parallel {}^3p_2{}^5) \cdot {}^4({}^5s^6) \cdot \delta$. Equivalent terms, thus representing the same process, can be found in Example 6.3.

Two other examples of (non-terminating) processes that are not serializable are ${}^{2}({}^{3}[p_1, p_2]^{4})^{5} \cdot \delta$ and $\rho_f({}^{2}p^{3} \cdot \delta)$ where $\boldsymbol{f}$ is the renaming function that renames action $(2, p, 3)$ into action $([2, 3], [p_1, p_2], [4, 5])$. Finally, term C as defined in Example 5.35 corresponds to a (terminating) process that is not serializable (provided the definition of serializability is adapted to the setting without causes).

The next step is to turn the set of non-terminating serializable processes into an algebra. For this purpose, it is necessary to define a signature for the algebra containing a number of meaningful functions on non-terminating serializable processes. The signature of $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC})(AC(C, A), \uplus)$ might appear to be a suitable starting point. However, not all constants and operators in this signature are useful.

The domain of an algebra must be closed under function application. In other words, a constant in the signature of the algebra of non-terminating serializable processes must, of course, be a non-terminating serializable process; an operator applied to non-terminating serializable processes must yield a non-terminating serializable process again. The consequences of these observations become clear when considering the following. First, the actions in $AC(C, A)$ correspond to processes that can terminate successfully. Thus, these actions are not suitable as constants in our algebra. Second, as illustrated in Example 10.7, processes that can perform isolated actions of which the step contains more than one atomic action are not serializable. Third, the communication merge often yields processes that are not serializable. Finally, some renaming functions, such as the one given in Example 10.7, may yield processes that are not serializable even if they are applied to a serializable process.

The above observations identify a number of constants and operators that cause technical problems. However, in addition, several of the operators in the signature of $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC})(AC(C, A), \uplus)$ are not very meaningful in the context of non-terminating processes. The standard ACP-style sequential-composition operator, for example, is not very useful when its left operand is a non-terminating process. The same is true for the binary Kleene star. In addition, also output-cause addition is not very meaningful for non-terminating processes.

Many of the problems can be solved by replacing the sequential-composition operator by a set of action-prefix operators in the style of CCS [43, 44] and the binary Kleene star by a class of prefix-iteration operators [12, 24]. It is important to define a suitable class of actions that can form the basis for these prefix operators. The following definition is inspired by the notion of a transition in a P/T net. A Petri element can be seen as the algebraic equivalent of a transition combined with its bags of input and output places.

Definition 10.8. (Petri elements) The set of *Petri elements* over the sets of causes C and atomic actions A, denoted $PE(C, A)$, is inductively defined as follows. For each atomic action a in A, $(\mathbf{0}, [a], \mathbf{0})$ is an element of $PE(C, A)$. For each Petri element e in $PE(C, A)$ and cause c in C, ${}^{c}e$ and e^{c} are elements of $PE(C, A)$.

Note that each Petri element in $PE(C, A)$ is derivably equal to an action in $AC(C, A)$ with a step consisting of only a single atomic action. The converse is also true.

For each Petri element e in $PE(C, A)$, the operator $e \cdot \_$ is an action-prefix operator and $e * \_$ is a prefix-iteration operator. The semantics of action-prefix and prefix-iteration operators follows immediately from the semantics of actions, input-cause addition, output-cause addition, sequential composition, and the binary Kleene star.

Most of the problems identified above are solved if the action constants, the sequential-composition operator, the binary Kleene star, and the output-cause-addition operators are replaced by the above prefix operators. Two problems remain, namely the communication merge and the renaming operators. The solution to the first problem is simply to remove the communication merge from the signature of our algebra. To solve the second problem, recall the subset of renaming functions defined in Section 7. All these renaming functions are derived either from a renaming function or a communication function on atomic actions or from a renaming function on causes. That is, they are all defined in terms of the structure of actions. All these renaming functions using the structure of actions can be used as the basis for a renaming operator in the signature of our algebra. Observe that the renaming function in Example 10.7 is not defined in terms of the structure of actions. Recall from Section 5.1 that *RF* denotes the set of all renaming functions. Let $SRF \subseteq RF$ be the subset of renaming functions that consists of all the renaming functions introduced in Section 7.

At this point, we have obtained a signature of meaningful operators. It contains the inaction constant δ, the standard ACP operators choice, merge, left merge, and a whole class of renaming operators, including encapsulation operators, explicit communication operators, and cause-abstraction operators, as well as a notion of sequential composition, a notion of iteration, the causal state operator, and input-cause addition. Let Σ denote the signature containing constant δ, the operators $+$, $\|$, and $⌊⌊$, for each $f \in SRF$, the operator ρ_f, for each $e \in PE(C, A)$, the operators $e\,\cdot$ and $e\,^*$, for each $I \subseteq C$ and $m \in \mathcal{B}(I)$, the operator λ^I_m, and, for each $c \in C$, the operator ${}^c\_$.

Example 10.9. Consider again Example 10.7. The iteration in the term $({}^1r^2 \cdot {}^2p^3 \cdot {}^3s^1)\,^*\,\delta$ is not a prefix iteration. However, the equivalent term $\lambda_1^{\{1,2,3\}}({}^1r^2\,^*\,\delta \parallel {}^2p^3\,^*\,\delta \parallel {}^3s^1\,^*\,\delta)$ only contains prefix iterations. The term $({}^1r^2)^3 \cdot ({}^2p_1{}^4 \parallel {}^3p_2{}^5) \cdot {}^4({}^5s^6) \cdot \delta$ contains two Petri elements outside the context of an action prefix and one general sequential composition. Thus, it is not in the restricted signature Σ defined above. It is an interesting exercise to construct an equivalent term that only uses operators of the restricted signature (see also Example 6.3). Finally, it is not difficult to verify that none of the terms in Example 10.7 corresponding to a process that is not serializable is in the restricted signature Σ.

Theorem 10.10. *The set of non-terminating serializable processes $\mathcal{C}_{\mathrm{ns}}(C, A)$ is closed under the operators in signature Σ.*

Proof. The proof consists of a tedious, but rather straightforward, analysis of the transition relation defined by the derivation rules in Tables 5.17, 6.6, and 9.2. Let p and q be two closed terms in $\mathcal{C}_{\mathrm{ns}}(C, A)$.

i) It must be shown that $p + q$ is an element of $\mathcal{C}_{\mathrm{ns}}(C, A)$. First, it must be proven that $\neg \Downarrow p + q$. This follows immediately from the observation that $p + q$ can only perform an action if either p or q can perform that action and the fact that $\neg \Downarrow p$ and $\neg \Downarrow q$. Second, it must be shown that $p + q$ is serializable. The desired result follows from again the observation that $p + q$ can only perform an action if either p or q can perform that action and the fact that both p and q are serializable.

ii) It must be shown that $p \| q$ is an element of $\mathcal{C}_{\mathrm{ns}}(C, A)$. First, it follows immediately from the derivation rules in Table 5.17 that $\Downarrow p \parallel q$ if and only if both $\Downarrow p$ and $\Downarrow q$. Hence, $\neg \Downarrow p \parallel q$. Second, assume that r is a closed term in $\mathcal{C}(C, A)$ such that $p \parallel q \stackrel{*}{\Longrightarrow} r$. It follows from the fact that both $\neg \Downarrow p$ and $\neg \Downarrow q$ that r must be of the form $p' \parallel q'$ with $p', q' \in \mathcal{C}(C, A)$ such that $p \stackrel{*}{\Longrightarrow} p'$ and $q \stackrel{*}{\Longrightarrow} q'$. Let a be an action in $AC(C, A)$ and let α_1 and α_2 be two non-empty bags in $\mathcal{B}(A) \setminus \{\mathbf{0}\}$ such that $\mathbf{s}a = \alpha_1 \uplus \alpha_2$ and r can perform the action a. Three cases can be distinguished. First, there exists a closed term $p'' \in \mathcal{C}(C, A)$ such that $p' \xrightarrow{a} p''$, which means that $r \equiv p' \parallel q' \xrightarrow{a} p'' \parallel q'$. Since p is serializable, there exist a closed term $p''' \in \mathcal{C}(C, A)$ and actions $a_1, a_2 \in AC(C, A)$ such that $\mathbf{s}a_1 = \alpha_1$, $\mathbf{s}a_2 = \alpha_2$, $a = a_1 \uplus a_2$, and $p' \xrightarrow{a_1} p''' \xrightarrow{a_2} p''$. Hence, $p' \parallel q' \xrightarrow{a_1} p''' \parallel q' \xrightarrow{a_2} p'' \parallel q'$, which means that $p \parallel q$ satisfies the serializability requirement in this case. Second, there exists a closed term $q'' \in \mathcal{C}(C, A)$ such that $q' \xrightarrow{a} q''$, which means that $r \equiv p' \| q' \xrightarrow{a} p' \| q''$. The argument proving that $p \| q$ satisfies the serializability requirement in this case is identical to the previous case. Third, there exist closed terms $p'', q'' \in \mathcal{C}(C, A)$ and actions $b, c \in AC(C, A)$ such that $a = \gamma(b, c) = b \uplus c$, $p' \xrightarrow{b} p''$, and $q' \xrightarrow{c} q''$, which means that $r \equiv p' \parallel q' \xrightarrow{a} p'' \parallel q''$. Since $\mathbf{s}a = \alpha_1 \uplus \alpha_2$, it follows that there are bags $\alpha_1^b, \alpha_1^c, \alpha_2^b, \alpha_2^c \in \mathcal{B}(A)$ such that $\alpha_1 = \alpha_1^b \uplus \alpha_1^c$, $\alpha_2 = \alpha_2^b \uplus \alpha_2^c$, $\mathbf{s}b = \alpha_1^b \uplus \alpha_2^b$, and $\mathbf{s}c = \alpha_1^c \uplus \alpha_2^c$. We only consider the case that $\alpha_1^b, \alpha_2^b, \alpha_1^c$, and α_2^c are all non-empty. The serializability of p and q yields that there exist $p''', q''' \in \mathcal{C}(C, A)$ and actions $b_1, b_2, c_1, c_2 \in AC(C, A)$ such that $\mathbf{s}b_1 = \alpha_1^b$,

$sb_2 = \alpha_2^b$, $b = b_1 \uplus b_2$, $p' \xrightarrow{b_1} p''' \xrightarrow{b_2} p''$, $sc_1 = \alpha_1^c$, $sc_2 = \alpha_2^c$, $c = c_1 \uplus c_2$, and $q' \xrightarrow{c_1} q''' \xrightarrow{c_2} q''$. It easily follows that $r \equiv p' \parallel q' \xrightarrow{b_1 \uplus c_1} p''' \parallel q''' \xrightarrow{b_2 \uplus c_2} p'' \parallel q''$ with $s(b_1 \uplus c_1) = \alpha_1$, $s(b_2 \uplus c_2) = \alpha_2$, and $a = (b_1 \uplus c_1) \uplus (b_2 \uplus c_2)$, which means that also in this case $p \parallel q$ satisfies the serializability requirement.

iii) It must be shown that $p \;\|\!\!\lfloor\; q$ is an element of $\mathcal{C}_{\mathrm{ns}}(C, A)$. The proof is almost identical to the proof for the merge operator.

iv) It must be shown that $\rho_f(p)$ with $f \in SRF$ is an element of $\mathcal{C}_{\mathrm{ns}}(C, A)$. Clearly, $\neg \Downarrow \rho_f(p)$, because $\neg \Downarrow p$. To proof that $\rho_f(p)$ is serializable, assume that r is a closed term in $\mathcal{C}(C, A)$ such that $\rho_f(p) \stackrel{*}{\Longrightarrow} r$. It follows from the fact that $\neg \Downarrow p$ that r must be of the form $\rho_f(p')$ with $p' \in \mathcal{C}(C, A)$ such that $p \stackrel{*}{\Longrightarrow} p'$. Furthermore, assume that fa is an action in $AC(C, A)$ and that α_1 and α_2 are two non-empty bags in $\mathcal{B}(A) \setminus \{\mathbf{0}\}$ such that $sfa = \alpha_1 \uplus \alpha_2$ and r can perform the action fa. For all the renaming functions in SRF, it follows from the fact that p is serializable that there are actions $a, a_1, a_2 \in AC(C, A)$ and closed terms $p'', p''' \in \mathcal{C}(C, A)$ such that (1) $f(a) = fa$, $p' \xrightarrow{a} p''$, and $r \xrightarrow{fa} \rho_f(p'')$, (2) $a = a_1 \uplus a_2$ and $p' \xrightarrow{a_1} p''' \xrightarrow{a_2} p''$, and (3) $s(f(a_1)) = \alpha_1$, $s(f(a_2)) = \alpha_2$, $fa = f(a_1) \uplus f(a_2)$, and $r \xrightarrow{f(a_1)} \rho_f(p''') \xrightarrow{f(a_2)} \rho_f(p'')$. As a result, $\rho_f(p)$ is serializable.

v) It must be shown that $e \cdot p$ with $e \in PE(C, A)$ is an element of $\mathcal{C}_{\mathrm{ns}}(C, A)$. It follows from Definition 10.8 (Petri elements) that there exist unique bags of causes $c, d \in \mathcal{B}(C)$ and a unique atomic action $a \in A$ such that $e \cdot p \xrightarrow{(c,[a],d)} p$. Process $e \cdot p$ cannot perform any other action. Thus, it follows easily from the fact that $\neg \Downarrow p$ that also $\neg \Downarrow e \cdot p$. In addition, since the first action of $e \cdot p$ consists of a single atomic action and p is serializable, $e \cdot p$ satisfies the serializability requirement.

vi) It must be shown that $e^* p$ with $e \in PE(C, A)$ is an element of $\mathcal{C}_{\mathrm{ns}}(C, A)$. First, since $\neg \Downarrow p$, also $\neg \Downarrow e^* p$. Second, assume that r is a closed term in $\mathcal{C}(C, A)$ such that $e^* p \stackrel{*}{\Longrightarrow} r$. It follows from the fact that e is a Petri element and that $\neg \Downarrow p$ that either r must be equal to $e^* p$ and all actions executed to reach r from $e^* p$ correspond to e or r must be of the form p' with $p' \in \mathcal{C}(C, A)$ such that $p \stackrel{*}{\Longrightarrow} p'$. Thus, serializability of $e^* p$ follows easily from the fact that e is a Petri element and that p is serializable.

vii) It must be shown that $\lambda_m^I(p)$ with $I \subseteq C$ and $m \in \mathcal{B}(I)$ is an element of $\mathcal{C}_{\mathrm{ns}}(C, A)$. Clearly, $\neg \Downarrow \lambda_m^I(p)$, because $\neg \Downarrow p$. The fact that $\lambda_m^I(p)$ is serializable follows from the serializability of p and the observation that, for any two actions $a_1, a_2 \in AC(C, A)$, $c(a_1 \uplus a_2) = ca_1 \uplus ca_2$.

viii) Finally, it must be shown that ${}^c p$ with $c \in C$ is an element of $\mathcal{C}_{\mathrm{ns}}(C, A)$. Again, $\neg \Downarrow {}^c p$, because $\neg \Downarrow p$. The fact that ${}^c p$ is serializable follows from the serializability of p and the observation that the input-cause addition operator ${}^c\_$ only affects the consumption of the first action performed by p. □

The final step in the construction of the algebra of non-terminating serializable processes is to restrict the set of closed terms in $\mathcal{C}_{\mathrm{ns}}(C, A)$ to closed terms over the restricted signature Σ. The set of all non-terminating serializable processes over signature Σ is denoted $\mathcal{C}_{\mathrm{ns}}^-(C, A)$. The following result is a direct consequence of Theorem 10.10.

Corollary 10.11. *The set $\mathcal{C}_{\mathrm{ns}}^-(C, A)$ of all non-terminating serializable processes over signature Σ is closed under the operators in Σ.*

Based on this result, the set of non-terminating serializable processes over signature Σ can be turned into a process algebra following the approach of Section 5.2.

Definition 10.12. (The algebra of non-terminating serializable processes) The process algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$ of non-terminating serializable processes is defined as follows. The domain of $\mathcal{A}_{\mathrm{ns}}(C, A)$ consists of equivalence classes of closed terms in $\mathcal{C}^{-}_{\mathrm{ns}}(C, A)$ modulo bisimilarity. The signature of $\mathcal{A}_{\mathrm{ns}}(C, A)$ contains, for each function $\oplus$ in Σ, a function $\bar{\oplus}$ which is function $\oplus$ lifted to equivalence classes of closed terms.

The process algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$ is a subalgebra of the algebra $\mathcal{M}(C, A, \uplus)$ given in Section 9.2. The algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$ incorporates several important features of the Petri-net formalism. An interesting aspect of $\mathcal{A}_{\mathrm{ns}}(C, A)$ is that it is a (branching-time) *non-interleaving, partial-order* process algebra. To prove this claim, it is necessary to slightly adapt the framework developed in Section 5.2. Definition 5.22 (Head normal form) is based on the assumption that the signature of an algebra contains a set of action constants and a general sequential-composition operator, which is not the case for $\mathcal{A}_{\mathrm{ns}}(C, A)$. However, it is straightforward to adapt the definition of a head normal form to the current setting as follows.

Definition 10.13. (Head normal form) The set of processes over the signature of the algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$ in *head normal form*, denoted $\mathcal{H}_{\mathrm{ns}}(C, A)$, is inductively defined as follows. First, $\bar{\delta}$ is an element of $\mathcal{H}_{\mathrm{ns}}(C, A)$. Second, for any Petri element e in $PE(C, A)$ and process d in the domain of $\mathcal{A}_{\mathrm{ns}}(C, A)$, the process $\bar{e} \bar{\cdot} d$ is an element of $\mathcal{H}_{\mathrm{ns}}(C, A)$. Finally, for any processes u and v in $\mathcal{H}_{\mathrm{ns}}(C, A)$, process $u \mathbin{\bar{+}} v$ is an element of $\mathcal{H}_{\mathrm{ns}}(C, A)$.

Theorem 10.14. *The process algebra* $\mathcal{A}_{\mathrm{ns}}(C, A)$ *is a* non-interleaving, partial-order *process algebra.*

Proof. To prove the theorem, consider the closed term $e \cdot \delta \parallel f \cdot \delta$, where $e \equiv (\mathbf{0}, [a], \mathbf{0})$ and $f \equiv (\mathbf{0}, [b], \mathbf{0})$, for $a, b \in A$, are two Petri elements in $PE(C, A)$. Ignoring consumptions and productions, Figure 10.15 visualizes this process. Clearly, this process is non-terminating and serializable, which means that it is an element of $\mathcal{C}^{-}_{\mathrm{ns}}(C, A)$. The corresponding process in the algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$ is $\bar{e} \bar{\cdot} \bar{\delta} \mathbin{\bar{\parallel}} \bar{f} \bar{\cdot} \bar{\delta}$. Since the process can perform an action $(\mathbf{0}, [a, b], \mathbf{0})$, $\mathcal{A}_{\mathrm{ns}}(C, A)$ is a partial-order algebra. In addition, the process is not an element of $\mathcal{H}_{\mathrm{ns}}(C, A)$, as defined in Definition 10.13, which means that it is not in head normal form. The process cannot be written in head normal form, because action $(\mathbf{0}, [a, b], \mathbf{0})$ is not a Petri element and because $|$ is not an element of signature Σ. As a consequence, according to Definition 5.23 (Interleaving process algebra), $\mathcal{A}_{\mathrm{ns}}(C, A)$ is a non-interleaving process algebra. $\square$

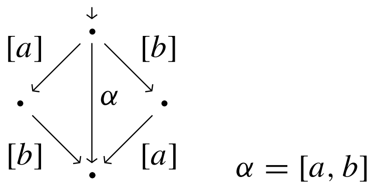


Figure 10.15: The visualization of a process in $\mathcal{A}_{\mathrm{ns}}(C, A)$ that can perform two atomic actions simultaneously and is not in head normal form.

The process algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$ is not obtained directly as a model of an equational theory, but indirectly as a subalgebra of a model of an equational theory, namely the theory $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC})(AC(C, A), \uplus)$. An interesting consequence is that this equational theory can be used to reason about the equivalence of processes in the algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$. The following theorem states that an equation of two closed terms in $\mathcal{C}^{-}_{\mathrm{ns}}(C, A)$ that can be derived from the equational theory $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC})(AC(C, A), \uplus)$ is valid in the algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$. That is, the two closed terms specify the same process in the domain of the algebra. (Note that it is straightforward to adapt the notion of validity to the current setting.)

Theorem 10.16. (Soundness) *For any closed terms* $p, q \in \mathcal{C}^{-}_{\mathrm{ns}}(C, A)$,
$(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC})(AC(C, A), \uplus) \vdash p = q \Rightarrow \mathcal{A}_{\mathrm{ns}}(C, A) \models p = q.$

Proof. It is a direct consequence of Theorem 9.4 that states that $\mathcal{M}(C, A, \uplus)$ is a model for $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^{*} + \mathrm{SC})(AC(C, A), \uplus)$ and Definition 10.12 that defines $\mathcal{A}_{\mathrm{ns}}(C, A)$. $\square$

Example 10.17. Consider the closed term $e \cdot \delta \parallel \delta$, where $e \equiv (\mathbf{0}, [a], \mathbf{0})$, for $a \in A$, is a Petri element in $PE(C, A)$. Clearly, this term is an element of $\mathcal{C}^-_{\mathrm{ns}}(C, A)$. The following derivation shows that it is equivalent to term $e \cdot \delta$ which is also an element of $\mathcal{C}^-_{\mathrm{ns}}(C, A)$. The interesting aspect of the derivation is that the intermediate term is not an element of $\mathcal{C}^-_{\mathrm{ns}}(C, A)$.

$$
\begin{array}{ll}
 & e \cdot \delta \parallel \delta \\
= & \quad \{ \, CM1, CM3, CM5 \, \} \\
 & e \cdot (\delta \cdot \delta + \delta \cdot \delta + \delta \mid \delta) + \delta \cdot e \cdot \delta + (e \mid \delta) \cdot \delta \\
= & \quad \{ \, A6, A7, C1, C3 \, \} \\
 & e \cdot \delta
\end{array}
$$

It is not difficult to see that the first and last terms of the above derivation correspond to the same process in the algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$.

It is interesting to study a slightly larger example.

Example 10.18. Let us return to Example 6.5. This example describes two specifications of a two-place buffer, one specification using the standard communication mechanism of ACP-style process algebra and one specification using the state operator. It is straightforward to adapt the latter specification in such a way that the result is a term in $\mathcal{C}^-_{\mathrm{ns}}(C, A)$. Let A be some set of atomic actions that includes the actions r_1, r_2, s_2, c_2, and s_3; let C be a set of causes including the causes 1, 2, 3, 4, and 5. Let I be the set of causes $\{1, 2, 3, 4, 5\}$.

$$\rho_{\boldsymbol{ca}(I)}(\lambda^I_{[1,2]}({}^1({}^2r_1{}^3)^* \delta \parallel ({}^3c_2{}^4)^{5\,*} \delta \parallel {}^4r_1{}^{1\,*} \delta \parallel {}^5s_3{}^{2\,*} \delta)) \tag{1}$$

The other specification in Example 6.5 is a modular specification. Two terms $B_{1,2}$ and $B_{2,3}$ specify two one-place buffers. Clearly, these terms are not terms over the restricted signature Σ. Therefore, in this example, $B_{1,2}$ and $B_{2,3}$ are defined as follows.

$B_{1,2} = \rho_{\boldsymbol{ca}(\{1,2\})}(\lambda^{\{1,2\}}_1({}^1r_1{}^{2\,*} \delta \parallel {}^2s_2{}^{1\,*} \delta))$ and
$B_{2,3} = \rho_{\boldsymbol{ca}(\{2,3\})}(\lambda^{\{2,3\}}_2({}^2r_2{}^{3\,*} \delta \parallel {}^3s_3{}^{2\,*} \delta))$.

It is not difficult to verify that the encapsulation operator ∂_H and the renaming operator $\rho_{\boldsymbol{cm}}$ defined in Example 6.5 satisfy the requirements explained in Section 7, which means that they are elements of signature Σ. As a consequence, the two-place buffer can be specified by the following term in $\mathcal{C}^-_{\mathrm{ns}}(C, A)$.

$$\partial_H(\rho_{\boldsymbol{cm}}(B_{1,2} \parallel B_{2,3})) \tag{2}$$

By now, it should no longer be a problem to verify by means of the equational theory $(\mathrm{ACP}^{*c}_{\lambda} + \mathrm{RN} + RSP^* + \mathrm{SC})(AC(C, A), \uplus)$ that both specifications (1) and (2) are equivalent to the following term (see also Example 6.5).

$$r_1 \cdot ((c_2 \cdot (r_1 \parallel s_3))^* \delta) \tag{3}$$

Clearly, this term is not an element of $\mathcal{C}^-_{\mathrm{ns}}(C, A)$ (although it is an element of $\mathcal{C}_{\mathrm{ns}}(C, A)$).

Summarizing, closed terms (1) and (2) have direct interpretations in the algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$. The equivalence of these two terms can be proven via an intermediate term (3), which has no direct interpretation in $\mathcal{A}_{\mathrm{ns}}(C, A)$.

So far, it has been explained how to construct an algebra of non-terminating serializable processes. It has been shown that this algebra is a non-interleaving partial-order process algebra. In that sense, it has the same characteristics as the P/T-net framework of Section 4. It allows algebraic specifications in the style of labeled P/T nets (using parallel compositions, cause addition, and causal state operators as the main operators). In addition, it has been shown that the (interleaving) equational theory of Section 9.1 can be used to reason about processes in the non-interleaving algebra. Example 10.18 illustrates another interesting aspect of the algebra of non-terminating serializable processes. It incorporates both the standard causality mechanism known from process algebra (consisting of sequential composition and communication) and the causality mechanism adopted from Petri nets (consisting of the causal state operator and cause addition). It is interesting to study these two mechanisms in more detail.

10.2 A brief discussion on causality

In an interleaving process algebra, each parallel composition can be written equivalently in head normal form. In standard ACP-style process algebra, this means that sequential composition is sufficient to express any causal relationships. That is, communication is not necessary as a primitive causality mechanism. However, in a non-interleaving process algebra, it is no longer possible to express each parallel composition by means of a head normal form. In the algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$ developed in the previous subsection, it is not possible to specify the two-place buffer of Example 10.18 without either communication or a causal state operator. However, the reason is the restricted expressivity of the prefix-iteration operators. In [4], a non-interleaving, partial-order algebra of (possibly terminating) serializable processes is defined in a way very similar to the definition of the non-interleaving algebra of the previous subsection. The algebra in [4] contains the standard algebraic operators, including general sequential composition, plus the causal state operator and cause addition. In addition, the algebra incorporates linear recursion. As a result, in that algebra, it is possible to specify a two-place buffer without communication or a causal state operator. However, it is shown that it is not possible to specify a *three*-place buffer without communication or a causal state operator.

The consequence of this last observation becomes clear when we observe that the technique to construct a non-interleaving process algebra can also be applied to standard ACP-style process algebras. Consider, for example, the standard theory $(\mathrm{ACP}^*+\mathrm{RN}+RSP^*+\mathrm{SC})(\mathcal{B}(A), \uplus)$ as introduced in Section 5, where A is some set of atomic actions. Its model $\mathcal{M}(\mathcal{B}(A), \uplus)$ of Section 5.2 can be reduced to a non-interleaving process algebra following the approach of the previous subsection. The communication merge and the action constants other than the singleton bags are removed from the signature of theory $(\mathrm{ACP}^*+\mathrm{RN}+RSP^*+\mathrm{SC})(\mathcal{B}(A), \uplus)$. This restricted signature and the set of serializable processes over this signature form the basis of a non-interleaving process algebra that is defined in the same way as in Definition 10.12. The interesting aspect is that the signature of the resulting non-interleaving process algebra does not contain causal state operators or cause-addition operators. That is, it only contains the standard process-algebraic causality mechanism consisting of sequential composition and (explicit) communication. Thus, the consequence of the above-mentioned result of [4] is that, in a non-interleaving process algebra derived from a standard ACP-style process algebra including linear recursion, it is not possible to specify even a relatively simple process as a three-place buffer without (explicit) communication. That is, in such a non-interleaving process algebra, sequential composition is no longer sufficient to express causal relationships.

The question remains to what extent the causality mechanism adopted from Petri-net theory consisting of cause addition and the causal state operator is sufficiently powerful to replace sequential composition and communication. It is difficult to answer this question for a general algebraic theory including some form of recursion. In [4], it is claimed that the causal state operator and cause addition are sufficiently powerful to replace both communication and sequential composition in a very general setting of (not necessarily serializable) processes with bounded behavior. In the next subsection, we prove a theorem formalizing this claim for non-terminating serializable processes with bounded behavior.

10.3 Elimination of choice, sequential composition, and communication

As explained, the aim of this subsection is to prove that, in a process algebra of non-terminating serializable processes with bounded behavior, the ACP-style causality mechanism consisting of communication and sequential composition can be replaced by a causality mechanism consisting of the causal state operator and cause addition. In fact, it is possible to prove even a slightly stronger result: In the proposed setting, the standard ACP operators choice, sequential composition, and (explicit) communication can be eliminated from any process specification. That is, also the choice operator is no longer needed as a primitive operator.

Let C be some set of causes and A some set of atomic actions. Consider the equational theory $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \gamma)$, the Algebra of Communicating Processes with cause addition, causal state operator, renaming, and standard concurrency, where $AC(C, A)$ is the standard set of actions defined before and where γ is some communication function. Theory $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \gamma)$ extends the equational theory of Section 9.4, $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \gamma)$, with the standard-concurrency axioms of Table 5.6.

As before, it is assumed that the communication function equals $\uplus$. Let $\mathcal{C}(C, A)$ be the set of closed $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus)$ terms. Following the approach of Section 5.2, it is possible to construct a model $\mathcal{M}(C, A, \uplus)$ of the equational theory with the underlying process space $(\mathcal{C}(C, A) \cup \{\surd\}, AC(C, A), \longrightarrow, \{\surd\})$. Note that this model is an interleaving, partial-order process algebra with a domain that contains only processes with bounded behavior. The set of non-terminating serializable processes in $\mathcal{C}(C, A)$, as defined in Definitions 10.1 (Non-terminating process) and 10.2 (Serializability), is denoted $\mathcal{C}_{\mathrm{bns}}(C, A)$. This set of non-terminating serializable processes can be turned into a non-interleaving, partial-order process algebra $\mathcal{A}_{\mathrm{bns}}(C, A)$ following the approach explained in Section 10.1. Recall that $SRF \subseteq RF$ is the subset of renaming functions that consists of all the renaming functions introduced in Section 7 and that $PE(C, A)$ is the set of Petri elements over C and A as defined in Definition 10.8. The signature of $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus)$ is restricted to the signature Σ_{b} containing inaction constant δ, the operators $+$, $\|$, and $\lfloor\!\lfloor$, for each $f \in SRF$, the operator ρ_f, for each $e \in PE(C, A)$, the action-prefix operator $e\,\cdot$, for each $I \subseteq C$ and $m \in \mathcal{B}(I)$, the operator λ^I_m, and, for each $c \in C$, the operator ${}^c\_$. Note that the signature Σ_{b} differs from the signature Σ introduced in Section 10.1 only with respect to prefix-iteration operators. It follows from (the proof of) Theorem 10.10 that the set of non-terminating serializable processes $\mathcal{C}_{\mathrm{bns}}(C, A)$ (as well as the set of non-terminating serializable processes over the restricted signature Σ_{b}) is closed under the operators in Σ_{b}. Thus, the algebra $\mathcal{A}_{\mathrm{bns}}(C, A)$ can be defined as explained in Definition 10.12 (Process algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$). In addition, it is possible to prove that equational theory $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \gamma)$ can be used to reason about the equivalence of processes in $\mathcal{A}_{\mathrm{bns}}(C, A)$, similar to Theorem 10.16 for the algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$. Observe that $\mathcal{A}_{\mathrm{bns}}(C, A)$ is a subalgebra of $\mathcal{A}_{\mathrm{ns}}(C, A)$; its signature is contained in the signature of $\mathcal{A}_{\mathrm{ns}}(C, A)$ and it contains all processes in the domain of $\mathcal{A}_{\mathrm{ns}}(C, A)$ with a bounded behavior and no other processes.

Example 10.19. Let p_1, p_2, r, and s be atomic actions in A; let 1, 2, 3, 4, 5, and 6 be causes in C. As before, assume that an action in $AC(C, A)$ with empty consumption and empty production is identified with its second element and that square brackets of singleton bags are omitted. Consider the term $r \cdot (p_1 \| p_2) \cdot s \cdot \delta$. It is not difficult to verify that it specifies a non-terminating serializable process with bounded behavior. That is, it is an element of $\mathcal{C}_{\mathrm{bns}}(C, A)$ (although it is not a term over the restricted signature Σ_{b}). Let $I = \{1, 2, 3, 4, 5, 6\}$. It is possible to derive from the axioms of $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus)$ that the above term is equivalent to term $\rho_{\boldsymbol{ca}(I)}(\lambda^I_1(({}^1r^2)^3 \cdot \delta \,\|\, {}^2{p_1}^4 \cdot \delta \,\|\, {}^3{p_2}^5 \cdot \delta \,\|\, {}^4({}^5s^6) \cdot \delta))$. The latter is a term over the restricted signature Σ_{b}. Furthermore, observe that it does not contain any choice operators or (explicit) communication operators. In addition, it only contains very specific occurrences of action-prefix operators; each occurrence of an action-prefix operator has operand δ.

The main theorem of this subsection (Theorem 10.27 given below) generalizes Example 10.19. That is, it shows that any non-terminating serializable process in $\mathcal{C}_{\mathrm{bns}}(C, A)$ is derivably equal to a term over the restricted signature Σ_{b} that does not contain any occurrences of the choice operator, any explicit communication operator, or any action-prefix operator applied to another term than the inaction constant δ. Note that, for any Petri element $e \in PE(C, A)$, the action prefix $e \cdot \delta$ can be seen as a *constant*, specifying a process that can perform a single action before terminating in a deadlock. In the remainder, we refer to such specific action prefixes as action-prefix constants in order to emphasize that such an action prefix should not be seen as a sequential composition. Theorem 10.27 proves that any term in $\mathcal{C}_{\mathrm{bns}}(C, A)$ is derivably equivalent to a parallel composition of action-prefix constants that is restricted by means of a causal state operator and where any newly-added causes used to enforce causal orderings are hidden by means of a cause-abstraction operator. Interpreting this result in the algebras $\mathcal{A}_{\mathrm{bns}}(C, A)$ and $\mathcal{A}_{\mathrm{ns}}(C, A)$, it means that all processes in the domain of $\mathcal{A}_{\mathrm{bns}}(C, A)$ and, thus, all processes with a bounded behavior in $\mathcal{A}_{\mathrm{ns}}(C, A)$ can be specified without choice, sequential composition, or communication. Theorem 10.27 strengthens our claim that the algebra $\mathcal{A}_{\mathrm{ns}}(C, A)$ of Section 10.1 incorporates some of the most important characteristics of Petri-net theory.

As explained in Section 9.4, an advantage of considering only processes with bounded behavior is that each process can be specified by means of a so-called basic term. Note that the signature of theory $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus)$ is identical to the signature of theory $(\mathrm{ACP}^c_\lambda + \mathrm{RN})(AC(C, A), \uplus)$ of Section 9.4. As a result, Definition 9.9 (Basic terms) carries over to the setting of this subsection. The following theorem states the elimination result for theory $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus)$.

Theorem 10.20. (Elimination) *For any closed term $p \in \mathcal{C}(C, A)$, there exists a basic term $t \in \mathcal{B}(C, A)$ such that $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash p = t$.*

Proof. The proof is identical to the proof of Theorem 9.10 (Elimination) of Section 9.4. □

The proof of Theorem 10.27 uses induction on the structure of basic terms. Five auxiliary properties are needed in the proof.

The next property is not used in the proof of Theorem 10.27, but in the proof of the first auxiliary result. It is a simple property of bisimilar processes in some arbitrary process space.

Property 10.21. *Assume that $(\mathcal{P}, \mathcal{A}, \longrightarrow, \downarrow)$ is some process space as defined in Definition 3.1. Furthermore, assume that $\stackrel{*}{\Longrightarrow} \subseteq \mathcal{P} \times \mathcal{P}$ is the reachability relation of Definition 3.2. Let p and q be two processes in $\mathcal{P}$; let $\mathcal{R}$ be a bisimulation between p and q as defined in Definition 3.8. For any $p', q' \in \mathcal{P}$,*

i) $p \stackrel{*}{\Longrightarrow} p' \Rightarrow (\exists q' : q' \in \mathcal{P} : q \stackrel{*}{\Longrightarrow} q' \wedge p'\mathcal{R}q')$ and

ii) $q \stackrel{*}{\Longrightarrow} q' \Rightarrow (\exists p' : p' \in \mathcal{P} : p \stackrel{*}{\Longrightarrow} p' \wedge p'\mathcal{R}q')$.

Proof. The two properties can be proven by means of straightforward induction on the number of actions needed to reach p' from p and q' from q, respectively. □

Property 10.21 can be used to prove the following result, which is the first auxiliary property needed in the proof of Theorem 10.27. Consider two derivably equivalent closed terms p and q in $\mathcal{C}(C, A)$. Process p is (non-)terminating if and only if process q is (non-)terminating; in addition, p is serializable if and only if q is serializable.

Property 10.22. *For any closed terms $p, q \in \mathcal{C}(C, A)$ such that $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash p = q$,*

i) $\Downarrow p \Leftrightarrow \Downarrow q$ and

ii) p serializable $\Leftrightarrow q$ serializable.

Proof. The two properties follow immediately from the fact that $\mathcal{M}(C, A, \uplus)$, with underlying process space $(\mathcal{C}(C, A) \cup \{\surd\}, AC(C, A), \longrightarrow, \{\surd\})$, is a model of $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus)$, Property 10.21, and Definitions 3.8 (Bisimilarity), 10.1 (Non-terminating process), and 10.2 (Serializability). □

The second auxiliary property states that any closed term specifying a non-terminating process is derivably equivalent to the sequential composition of a closed term and the inaction constant δ.

Property 10.23. *For any closed term $p \in \mathcal{C}(C, A)$,*
$\neg \Downarrow p \Leftrightarrow (\exists q : q \in \mathcal{C}(C, A) : (\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash p = q \cdot \delta)$.

Proof. First, assume that $\neg \Downarrow p$. It follows from Theorem 10.20 (Elimination) that there exists a basic term $t \in \mathcal{B}(C, A)$ such that $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash p = t$. Property 10.22*i*) implies that $\neg \Downarrow t$. Thus, it suffices to prove for any basic term $t \in \mathcal{B}(C, A)$ that $\neg \Downarrow t$ implies $(\exists q : q \in \mathcal{C}(C, A) : (\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash t = q \cdot \delta)$. The proof is by induction on the structure of t. The details are straightforward and left to the reader.

Second, assume that $(\exists q : q \in \mathcal{C}(C, A) : (\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash p = q \cdot \delta)$. It must be shown that $\neg \Downarrow p$. The desired result follows immediately from Property 10.22*i*) and the observation that $\neg \Downarrow q \cdot \delta$ for any $q \in \mathcal{C}(C, A)$. □

The last three auxiliary properties needed to prove Theorem 10.27 are simple properties concerning the merge, the communication merge, action-prefix constants, and the inaction constant. The proof of Property 10.26 uses Theorem 5.7 (Expansion) which carries over to the current setting.

Property 10.24. *For any closed term* $p \in \mathcal{C}(C, A)$,
$(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash p \parallel \delta = p \cdot \delta$.

Proof. It follows from Theorem 10.20 (Elimination) that it suffices to prove the property for basic terms. The proof by induction on the structure of basic terms is straightforward and left to the reader. □

Property 10.25. *For any non-empty, finite set of Petri elements* $E \subseteq PE(C, A)$,
$(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash (|\, e : e \in E : e \cdot \delta) = (|\, e : e \in E : e) \cdot \delta$.

Proof.

$$
\begin{array}{ll}
 & (|\, e : e \in E : e \cdot \delta) \\
= & \quad \{ \text{Definition 10.8 (Petri elements)}, CM7 \} \\
 & (|\, e : e \in E : e) \cdot (\parallel e : e \in E : \delta) \\
= & \quad \{ \text{Property 10.24}, A7 \} \\
 & (|\, e : e \in E : e) \cdot \delta
\end{array}
$$

□

Property 10.26. *For any non-empty, finite set of Petri elements* $E \subseteq PE(C, A)$,
$(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash (\parallel e : e \in E : e \cdot \delta) = (\parallel e : e \in E : e) \cdot \delta$.

Proof. The proof is by means of natural induction on the size of set E. If E contains only a single element, the property is trivial, because both sides of the equality reduce to the same term. Assume that E contains at least two elements. In the following derivation, applications of axioms of the equational theory $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus)$ are not explicitly mentioned.

$$
\begin{array}{ll}
 & (\parallel e : e \in E : e \cdot \delta) \\
= & \quad \{ \text{Expansion} \} \\
 & (+D : \emptyset \subset D \subset E : (|d : d \in D : d \cdot \delta) \mathbin{\lfloor\!\lfloor} (\|e : e \in E \setminus D : e \cdot \delta)) + (|e : e \in E : e \cdot \delta) \\
= & \quad \{ \text{Property 10.25, induction} \} \\
 & (+D : \emptyset \subset D \subset E : (|d : d \in D : d) \cdot \delta \mathbin{\lfloor\!\lfloor} (\|e : e \in E \setminus D : e) \cdot \delta) + (|e : e \in E : e) \cdot \delta \\
= & \quad \{ \text{Definition 10.8 (Petri elements), Property 10.24} \} \\
 & (+D : \emptyset \subset D \subset E : (|d : d \in D : d) \mathbin{\lfloor\!\lfloor} (\|e : e \in E \setminus D : e)) \cdot \delta + (|e : e \in E : e) \cdot \delta \\
= & \quad \{ \text{Expansion} \} \\
 & (\parallel e : e \in E : e) \cdot \delta
\end{array}
$$

□

At this point, it is possible to prove the main theorem of this subsection. It gives a canonical form for each process term in $\mathcal{C}_{\mathrm{bns}}(C, A)$. This canonical term consists of a parallel composition of action-prefix constants restricted by means of a causal state operator. Causes that do not occur in the original term but are added in the parallel composition to enforce the necessary orderings are hidden by means of a cause-abstraction operator. In the construction of the parallel composition of action-prefix constants, a continuous supply of fresh causes for specifying causal orderings is needed. Therefore, assume that the set of causes C is *infinite*. Note that Definition 9.12 (Causes function) carries over to the current setting.

Theorem 10.27. (Elimination of choice, sequential composition, and communication) *For any closed term $p \in \mathcal{C}_{\text{bns}}(C, A)$, there are $I \subseteq C$, $m \in \mathcal{B}(I)$, $n \in \mathbb{N}$, and $e_0, \ldots, e_n \in PE(C, A)$ such that*
$(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash p = \rho_{\boldsymbol{ca}(I)}(\lambda^I_m(\| i : 0 \leq i \leq n : e_i \cdot \delta))$.

Proof. It follows from Property 10.23 that there exists a closed term $q \in \mathcal{C}(C, A)$ such that $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash p = q \cdot \delta$. Property 10.22*ii*) implies that $q \cdot \delta$ is serializable. It follows from the derivation rules in Table 5.17 that also q is serializable. Theorem 10.20 (Elimination) and, again, Property 10.22*ii*) yield that there is a basic term $t \in \mathcal{B}(C, A)$ such that $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash q = t$ and t is serializable. Thus, to prove the theorem, it suffices to prove the following property: For any *serializable* basic term $t \in \mathcal{B}(C, A)$, there are $I \subseteq C$, $m \in \mathcal{B}(I)$, $n \in \mathbb{N}$, and $e_0, \ldots, e_n \in PE(C, A)$ such that

$$I \cap causes(t) = \emptyset \wedge (\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash t \cdot \delta = \rho_{\boldsymbol{ca}(I)}(\lambda^I_m(\| i : 0 \leq i \leq n : e_i \cdot \delta)).$$

The extra requirement concerning the set of causes I is needed in the proof. Let t be a serializable basic term in $\mathcal{B}(C, A)$. The proof is by induction on the structure of t.

i) Assume $t \equiv \delta$. Let c be an arbitrary cause in C and e an arbitrary Petri element in $PE(C, A)$. It easily follows that $\{c\} \cap causes(t) = \emptyset$ and that $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash \rho_{\boldsymbol{ca}(\{c\})}(\lambda^{\{c\}}_{\mathbf{0}}({}^c e \cdot \delta)) = \delta \cdot \delta = t \cdot \delta$, which completes the proof in this case.

ii) Assume $t \equiv a$, for some action $a \in AC(C, A)$. It follows from the fact that t is serializable, the derivation rules in Table 5.17, and Definition 10.2 (Serializability) that $\boldsymbol{sa}$ must be a singleton bag. Consequently, it follows from Definition 10.8 (Petri elements) that there is a Petri element $e \in PE(C, A)$ such that $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash a = e$. It is not difficult to prove that $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash \rho_{\boldsymbol{ca}(\emptyset)}(\lambda^{\emptyset}_{\mathbf{0}}(e \cdot \delta)) = a \cdot \delta = t \cdot \delta$, which completes the proof also in this case.

iii) Assume $t \equiv a \cdot s$, for some action $a \in AC(C, A)$ and basic term $s \in \mathcal{B}(C, A)$. Since t is serializable, it follows from Definition 10.2 (Serializability) that $\boldsymbol{sa}$ is a singleton bag and that also s is serializable. Consequently, Definition 10.8 (Petri elements) implies that there is a Petri element $e \in PE(C, A)$ such that $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash a = e$. In addition, by induction, it follows that there are $I \subseteq C$, $m \in \mathcal{B}(I)$, $n \in \mathbb{N}$, and $e_0, \ldots, e_n \in PE(C, A)$ such that

$$I \cap causes(s) = \emptyset \wedge$$
$$(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash s \cdot \delta = \rho_{\boldsymbol{ca}(I)}(\lambda^I_m(\| i : 0 \leq i \leq n : e_i \cdot \delta)) \tag{1}$$

Clearly, since C is infinite, it is possible to choose I in such a way that $I \cap causes(a) = \emptyset$ and, hence, $I \cap causes(t) = \emptyset$.

Let $c_0, \ldots, c_n \in C$ be new, distinct causes not in $I \cup causes(t)$. Assume that $N = \{i \mid 0 \leq i \leq n\}$. Let $I' = I \cup \{c_i \mid i \in N\}$; let $m' = m \uplus [c_i \mid i \in N]$; let $d \equiv (\ldots(e^{c_0})^{\cdots})^{c_n}$. Note that

$$(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash d = (\boldsymbol{ca}, \boldsymbol{sa}, \boldsymbol{pa} \uplus [c_i \mid i \in N]) \tag{2}$$

Furthermore, for all $J \subseteq N$ with $J \neq \emptyset$, there is some action $b \in AC(C, A)$ such that

$$(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash$$
$$(| j : j \in J : e_j) = b \wedge (| j : j \in J : {}^{c_j} e_j) = ([c_j \mid j \in J] \uplus \boldsymbol{cb}, \boldsymbol{sb}, \boldsymbol{pb}) \tag{3}$$

The remainder of this part of the proof is devoted to showing that $t \cdot \delta$ is equivalent to the following term.

$$\rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_m(d \cdot \delta \parallel (\| i : i \in N : {}^{c_i} e_i \cdot \delta))) \tag{4}$$

(Note the correspondence between this term and the construction of Theorem 9.13 (Elimination of sequential composition).) The next derivation shows that the only initial action of term (4) is the action a. Applications of axioms of the equational theory $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus)$ are not explicitly mentioned in the derivation unless no other results are used in a step.

$$
\begin{array}{ll}
& \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_m(d \cdot \delta \parallel (\parallel i : i \in N : {}^{c_i}e_i \cdot \delta))) \\
= & \quad \{ \mathit{CM}1 \} \\
& \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_m(d \cdot \delta \mathbin{⌊\!⌊} (\parallel i : i \in N : {}^{c_i}e_i \cdot \delta) + (\parallel i : i \in N : {}^{c_i}e_i \cdot \delta) \mathbin{⌊\!⌊} d \cdot \delta + d \cdot \delta \mid (\parallel i : i \in N : {}^{c_i}e_i \cdot \delta))) \\
= & \quad \{ \text{Expansion} \} \\
& \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_m(d \cdot \delta \mathbin{⌊\!⌊} (\parallel i : i \in N : {}^{c_i}e_i \cdot \delta))) \\
& + (+J : \emptyset \subset J \subset N : \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_m(((\mid j : j \in J : {}^{c_j}e_j \cdot \delta) \mathbin{⌊\!⌊} (\parallel i : i \in N \setminus J : {}^{c_i}e_i \cdot \delta)) \mathbin{⌊\!⌊} d \cdot \delta))) \\
& + \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_m((\mid i : i \in N : {}^{c_i}e_i \cdot \delta) \mathbin{⌊\!⌊} d \cdot \delta)) \\
& + (+J : \emptyset \subset J \subset N : \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_m(d \cdot \delta \mid ((\mid j : j \in J : {}^{c_j}e_j \cdot \delta) \mathbin{⌊\!⌊} (\parallel i : i \in N \setminus J : {}^{c_i}e_i \cdot \delta))))) \\
& + \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_m(d \cdot \delta \mid (\mid i : i \in N : {}^{c_i}e_i \cdot \delta))) \\
= & \quad \{ \text{Property 10.25, (2), (3)} \} \\
& \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_m(d \cdot (\delta \parallel (\parallel i : i \in N : {}^{c_i}e_i \cdot \delta)))) \\
& + (+J : \emptyset \subset J \subset N : \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_m((\mid j : j \in J : {}^{c_j}e_j) \cdot (\delta \parallel (\parallel i : i \in N \setminus J : {}^{c_i}e_i \cdot \delta) \parallel d \cdot \delta)))) \\
& + \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_m((\mid i : i \in N : {}^{c_i}e_i) \cdot (\delta \parallel d \cdot \delta))) \\
& + (+J : \emptyset \subset J \subset N : \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_m((d \mid (\mid j : j \in J : {}^{c_j}e_j)) \cdot (\delta \parallel (\parallel i : i \in N \setminus J : {}^{c_i}e_i \cdot \delta))))) \\
& + \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_m((d \mid (\mid i : i \in N : {}^{c_i}e_i)) \cdot (\delta \parallel \delta))) \\
= & \quad \{ (2), (3) \} \\
& a \cdot \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}(\delta \parallel (\parallel i : i \in N : {}^{c_i}e_i \cdot \delta))) \\
= & \quad \{ \text{Properties 10.24 and 10.26} \} \\
& a \cdot \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}(\parallel i : i \in N : {}^{c_i}e_i \cdot \delta))
\end{array}
$$

The next step is to show the following result.

$$
\begin{array}{l}
(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash \\
\quad \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}(\parallel i : i \in N : {}^{c_i}e_i \cdot \delta)) = \rho_{\boldsymbol{ca}(I)}(\lambda^{I}_{m}(\parallel i : i \in N : e_i \cdot \delta))
\end{array} \tag{5}
$$

That is, the extra input causes in the merge quantification in term (4) do not prevent the occurrence of any action of this quantification after the execution of the action corresponding to Petri element d has added the necessary causes to marking m resulting in marking m'. Recall that $N = \{i \mid 0 \le i \le n\}$. It is straightforward to show the desired result if n equals zero, because under this assumption the two quantifications in (5) reduce to single action-prefix constants ${}^{c_0}e_0 \cdot \delta$ and $e_0 \cdot \delta$, respectively. For n greater than zero, the property is proven by means of (strong) natural induction on n. The proof uses the following auxiliary property, which is an immediate corollary of property (3) above. For all $J \subseteq N$ with $J \neq \emptyset$,

$$
\begin{array}{l}
(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash \\
\quad \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}(\mid j : j \in J : {}^{c_j}e_j)) = \rho_{\boldsymbol{ca}(I)}(\lambda^{I}_{m}(\mid j : j \in J : e_j))
\end{array} \tag{6}
$$

Assume that $n > 0$. To prove property (5) in this case, the following auxiliary notations are needed. Based on Definition 10.8 (Petri element), it is possible to lift the consumption and production functions $\boldsymbol{c}\_$ and $\boldsymbol{p}\_$ from actions to Petri elements. For any $J \subseteq N$, $m(J) = m - (\uplus j : j \in J : \boldsymbol{c}e_j) \uplus (\uplus j : j \in J : \boldsymbol{p}e_j)$ and $m'(J) = m' - (\uplus j : j \in J : \boldsymbol{c}e_j) \uplus (\uplus j : j \in J : \boldsymbol{p}e_j) - [c_j \mid j \in J]$. Note that $m(\emptyset) = m$ and $m'(\emptyset) = m'$. Assume that, for any J with $J \subset N$, $E(J)$ denotes the following equality.

$$
\rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'(J)}(\parallel i : i \in N \setminus J : {}^{c_i}e_i \cdot \delta)) = \rho_{\boldsymbol{ca}(I)}(\lambda^{I}_{m(J)}(\parallel i : i \in N \setminus J : e_i \cdot \delta))
$$

Clearly, property (5) conforms to

$$
(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash E(\emptyset) \tag{7}
$$

The basis of the inductive proof is the following property. For all $i \in N$,

$$
(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash E(N \setminus \{i\}) \tag{8}
$$

Property (8) is easily proven, because in each case the quantifications in the desired equality reduce to single action-prefix constants.

The induction hypothesis is as follows. For all J with $\emptyset \subset J \subset N$,

$$(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash E(J) \tag{9}$$

The following derivation shows that $(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash E(\emptyset)$.

$$\begin{array}{ll}
 & \rho_{\mathbf{ca}(I')}(\lambda^{I'}_{m'}(\| i : i \in N : {}^{c_i}e_i \cdot \delta)) \\
= & \quad \{ \text{ Property 10.26, expansion, (3) } \} \\
 & (+J : \emptyset \subset J \subset N : \rho_{\mathbf{ca}(I')}(\lambda^{I'}_{m'}(| j : j \in J : {}^{c_j}e_j)) \cdot \rho_{\mathbf{ca}(I')}(\lambda^{I'}_{m'(J)}((\| i : i \in N \setminus J : {}^{c_i}e_i) \cdot \delta))) \\
 & \;+ \rho_{\mathbf{ca}(I')}(\lambda^{I'}_{m'}(| i : i \in N : {}^{c_i}e_i)) \cdot \delta \\
= & \quad \{ \text{ Property 10.26, (6), (9) } \} \\
 & (+J : \emptyset \subset J \subset N : \rho_{\mathbf{ca}(I)}(\lambda^{I}_{m}(| j : j \in J : e_j)) \cdot \rho_{\mathbf{ca}(I)}(\lambda^{I}_{m(J)}((\| i : i \in N \setminus J : e_i) \cdot \delta))) \\
 & \;+ \rho_{\mathbf{ca}(I)}(\lambda^{I}_{m}(| i : i \in N : e_i)) \cdot \delta \\
= & \quad \{ \text{ (3), expansion, Property 10.26 } \} \\
 & \rho_{\mathbf{ca}(I)}(\lambda^{I}_{m}(\| i : i \in N : e_i \cdot \delta))
\end{array}$$

This derivation proves property (7) and, thus, completes the proof of property (5).

Using property (5), the following derivation can be made.

$$\begin{array}{ll}
 & a \cdot \rho_{\mathbf{ca}(I')}(\lambda^{I'}_{m'}(\| i : i \in N : {}^{c_i}e_i \cdot \delta)) \\
= & \quad \{ \text{ (5) } \} \\
 & a \cdot \rho_{\mathbf{ca}(I)}(\lambda^{I}_{m}(\| i : i \in N : e_i \cdot \delta)) \\
= & \quad \{ \text{ (1) } \} \\
 & a \cdot (s \cdot \delta) \\
= & \quad \{ t \equiv a \cdot s \} \\
 & t \cdot \delta
\end{array}$$

Hence, combining the results obtained so far shows that

$$\begin{array}{l} I' \cap causes(t) = \emptyset \wedge \\ \quad (\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash t \cdot \delta = \rho_{\mathbf{ca}(I')}(\lambda^{I'}_{m}(d \cdot \delta \parallel (\| i : 0 \le i \le n : {}^{c_i}e_i \cdot \delta))), \end{array}$$

which completes the proof in this case.

iv) Assume $t \equiv u + v$, for some basic terms $u, v \in \mathcal{B}(C, A)$. Since t is serializable, it follows that u and v are both serializable. By induction, it follows that there are $I_u, I_v \subseteq C, m_u \in \mathcal{B}(I_u), m_v \in \mathcal{B}(I_v), n_u, n_v \in \mathbb{N}$, and $e_{u0}, \ldots, e_{un_u}, e_{v0}, \ldots, e_{vn_v} \in PE(C, A)$ such that

$$\begin{array}{l} I_u \cap causes(u) = \emptyset \wedge \\ \quad (\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash u \cdot \delta = \rho_{\mathbf{ca}(I_u)}(\lambda^{I_u}_{m_u}(\| i : 0 \le i \le n_u : e_{ui} \cdot \delta)) \end{array} \tag{1}$$

and

$$\begin{array}{l} I_v \cap causes(v) = \emptyset \wedge \\ \quad (\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash v \cdot \delta = \rho_{\mathbf{ca}(I_v)}(\lambda^{I_v}_{m_v}(\| j : 0 \le j \le n_v : e_{vj} \cdot \delta)) \end{array} \tag{2}$$

Since the set of causes C is infinite, it is possible to choose I_u and I_v in such a way that $I_u \cap I_v = \emptyset$, $I_u \cap causes(v) = \emptyset$, and $I_v \cap causes(u) = \emptyset$. As a result, $(I_u \cup I_v) \cap causes(t) = \emptyset$.

Let $N_u = \{i \mid 0 \le i \le n_u\}$ and $N_v = \{j \mid 0 \le j \le n_v\}$. Assume that, for any $i \in N_u$ and $j \in N_v$, $c_{ij} \in C$ is a new cause not in $I_u \cup I_v \cup causes(t) \cup \{c_{kl} \mid k \in N_u \wedge l \in N_v \wedge (k \ne i \vee l \ne j)\}$. Let $I' = I_u \cup I_v \cup \{c_{ij} \mid i \in N_u \wedge j \in N_v\}$ and $m' = m_u \uplus m_v \uplus [c_{ij} \mid i \in N_u \wedge j \in N_v]$. Finally, let, for any $i \in N_u$, $d_{ui} \equiv {}^{c_{i0}}(\cdots({}^{c_{in_v}}e_{ui})\ldots)$ and, for any $j \in N_v$, $d_{vj} \equiv {}^{c_{0j}}(\cdots({}^{c_{n_u j}}e_{vj})\ldots)$. The remainder of the proof shows that $t \cdot \delta$ is equivalent to the following term.

$$\rho_{\mathbf{ca}(I')}(\lambda^{I'}_{m'}((\| i : i \in N_u : d_{ui} \cdot \delta) \parallel (\| j : j \in N_v : d_{vj} \cdot \delta))) \tag{3}$$

Informally, the idea of the proof is as follows. New causes are added to each Petri element e_{ui} with $i \in N_u$, namely one cause c_{ij} for all $j \in N_v$, resulting in Petri element d_{ui}; similarly, for all $i \in N_u$ one cause c_{ij} with $j \in N_v$ is added to Petri element e_{vj}. This construction has two important consequences. First, no two Petri elements d_{ui} and d_{uk}, with $i, k \in N_u$ such that $i \neq k$, share any of the newly added causes. The same is true for any two Petri elements d_{vj} and d_{vl}, with $j, l \in N_v$ such that $j \neq l$. As a consequence, the newly added causes do not restrict the occurrence of actions *within* any of the two merge quantifications in term (3). Second, any two Petri elements d_{ui} and d_{vj} with $i \in N_u$ and $j \in N_v$ share exactly one input cause c_{ij}. Thus, assuming the initial marking m', it is not possible that actions of the two quantifications occur simultaneously. In addition, the occurrence of any action in one of the merge quantifications in term (3) effectively disables all actions in the other quantification, thus enforcing a true choice between the two quantifications.

The main line of the proof that term (3) equals $t \cdot \delta$ is as follows. The second step in the derivation uses four properties that are given below. These properties formalize the claims made above.

$$
\begin{array}{ll}
& \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}((\| i : i \in N_u : d_{ui} \cdot \delta) \parallel (\| j : j \in N_v : d_{vj} \cdot \delta))) \\
= & \quad \{ \text{ Axioms of } (\text{ACP}^c_\lambda + \text{RN} + \text{SC})(AC(C, A), \uplus) \ \} \\
& \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}((\| i : i \in N_u : d_{ui} \cdot \delta) \mathbin{\|\!\!\lfloor} (\| j : j \in N_v : d_{vj} \cdot \delta))) + \\
& \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}((\| j : j \in N_v : d_{vj} \cdot \delta) \mathbin{\|\!\!\lfloor} (\| i : i \in N_u : d_{ui} \cdot \delta))) + \\
& \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}((\| i : i \in N_u : d_{ui} \cdot \delta) \mid (| j : j \in N_v : d_{vj} \cdot \delta))) \\
= & \quad \{ \ (6), (7), \text{Property } 10.25, (4), (5) \ \} \\
& \rho_{\boldsymbol{ca}(I_u)}(\lambda^{I_u}_{m_u}(\| i : i \in N_u : e_{ui} \cdot \delta)) + \rho_{\boldsymbol{ca}(I_v)}(\lambda^{I_v}_{m_v}(\| j : j \in N_v : e_{vj} \cdot \delta)) + \delta \\
= & \quad \{ \ (1), (2) \ \} \\
& u \cdot \delta + v \cdot \delta \\
= & \quad \{ \ t \equiv u + v \ \} \\
& t \cdot \delta
\end{array}
$$

Properties (4) and (5) needed in the above derivation follow immediately from the definitions given earlier. For all $K \subseteq N_u$ with $K \neq \emptyset$, there is some action $a \in AC(C, A)$ such that

$$
\begin{array}{l}
(\text{ACP}^c_\lambda + \text{RN} + \text{SC})(AC(C, A), \uplus) \vdash \\
\quad (| k : k \in K : e_{uk}) = a \wedge (| k : k \in K : d_{uk}) = ([c_{kj} \mid k \in K \wedge j \in N_v] \uplus \boldsymbol{ca}, \boldsymbol{sa}, \boldsymbol{pa})
\end{array}
\tag{4}
$$

Furthermore, for all $L \subseteq N_v$ with $L \neq \emptyset$, there is some action $b \in AC(C, A)$ such that

$$
\begin{array}{l}
(\text{ACP}^c_\lambda + \text{RN} + \text{SC})(AC(C, A), \uplus) \vdash \\
\quad (| l : l \in L : e_{vl}) = b \wedge (| l : l \in L : d_{vl}) = ([c_{il} \mid i \in N_u \wedge l \in L] \uplus \boldsymbol{cb}, \boldsymbol{sb}, \boldsymbol{pb})
\end{array}
\tag{5}
$$

The combination of properties (4) and (5) implies that, in the initial marking m', actions from the two merge quantifications in term (3) cannot occur simultaneously, which means that the third alternative in the second expression of the above derivation can be reduced to the inaction constant δ.

It remains to prove the following two properties. They formalize the claims that the newly added causes do not restrict the occurrence of actions within one of the quantifications in term (3) and that the occurrence of an action within one quantification disables any actions from the other quantification.

$$
\begin{array}{l}
(\text{ACP}^c_\lambda + \text{RN} + \text{SC})(AC(C, A), \uplus) \vdash \\
\quad \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}((\| i : i \in N_u : d_{ui} \cdot \delta) \mathbin{\|\!\!\lfloor} (\| j : j \in N_v : d_{vj} \cdot \delta))) = \\
\qquad \rho_{\boldsymbol{ca}(I_u)}(\lambda^{I_u}_{m_u}(\| i : i \in N_u : e_{ui} \cdot \delta))
\end{array}
\tag{6}
$$

and

$$
\begin{array}{l}
(\text{ACP}^c_\lambda + \text{RN} + \text{SC})(AC(C, A), \uplus) \vdash \\
\quad \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}((\| j : j \in N_v : d_{vj} \cdot \delta) \mathbin{\|\!\!\lfloor} (\| i : i \in N_u : d_{ui} \cdot \delta))) = \\
\qquad \rho_{\boldsymbol{ca}(I_v)}(\lambda^{I_v}_{m_v}(\| j : j \in N_v : e_{vj} \cdot \delta))
\end{array}
\tag{7}
$$

For reasons of symmetry, it suffices to prove only property (6). The proof is similar to the proof of property (5) in part *iii*) of the proof of this theorem. The following auxiliary notation is needed. For any $K \subseteq N_u$, $m_u(K) = m_u - (\uplus k : k \in K : \boldsymbol{ce}_{uk}) \uplus (\uplus k : k \in K : \boldsymbol{pe}_{uk})$ and $m'(K) = m' - (\uplus k :$

$k \in K : \boldsymbol{c}d_{uk}) \uplus (\uplus k : k \in K : \boldsymbol{p}d_{uk})$. Using this notation, the following auxiliary property can be formulated.

$$(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'(N_u)}(\| \, j : j \in N_v : d_{vj} \cdot \delta)) = \delta \tag{8}$$

The proof is straightforward and left to the reader.

Recall that $N_u = \{i \mid 0 \leq i \leq n_u\}$. For $n_u = 0$, property (6) follows rather directly from property (8). Let $n_u > 0$. The basis of the proof is an inductive argument. Assume that, for any $K \subset N_u$, $E(K)$ denotes the following equality.

$$\rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'(K)}((\| \, i : i \in N_u \setminus K : d_{ui} \cdot \delta) \mathbin{\|\!\lfloor} (\| \, j : j \in N_v : d_{vj} \cdot \delta))) =$$
$$\rho_{\boldsymbol{ca}(I_u)}(\lambda^{I_u}_{m_u(K)}(\| \, i : i \in N_u \setminus K : e_{ui} \cdot \delta))$$

Clearly, property (6) reduces to the following property.

$$(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash E(\emptyset) \tag{9}$$

The basis of the inductive proof is the following property, which easily follows from property (8) above. For all $i \in N_u$,

$$(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash E(N_u \setminus \{i\}) \tag{10}$$

The induction hypothesis is as follows. For all $K \subset N_u$ with $K \neq \emptyset$,

$$(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash E(K) \tag{11}$$

The proof needs one more auxiliary property. For all $K \subseteq N_u$ with $K \neq \emptyset$,

$$(\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash$$
$$\rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}(|k : k \in K : d_{uk})) = \rho_{\boldsymbol{ca}(I_u)}(\lambda^{I_u}_{m_u}(|k : k \in K : e_{uk})) \tag{12}$$

Consider the following derivation.

$$\begin{aligned}
& \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}((\| \, i : i \in N_u : d_{ui} \cdot \delta) \mathbin{\|\!\lfloor} (\| \, j : j \in N_v : d_{vj} \cdot \delta))) \\
= & \quad \{ \text{ Property 10.26, expansion } \} \\
& \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}(\\
& \quad ((+K : \emptyset \subset K \subset N_u : (|k : k \in K : d_{uk}) \mathbin{\|\!\lfloor} (\|i : i \in N_u \setminus K : d_{ui})) + (|i : i \in N_u : d_{ui})) \\
& \qquad \cdot \delta \mathbin{\|\!\lfloor} (\| \, j : j \in N_v : d_{vj} \cdot \delta))) \\
= & \quad \{ \text{ (4), Properties 10.24 and 10.26 } \} \\
& (+K : \emptyset \subset K \subset N_u : \\
& \quad \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}(|k : k \in K : d_{uk})) \cdot \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'(K)}((\|i : i \in N_u \setminus K : d_{ui} \cdot \delta) \mathbin{\|\!\lfloor} (\| \, j : j \in N_v : d_{vj} \cdot \delta)))) \\
& \;\; + \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}(|i : i \in N_u : d_{ui})) \cdot \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'(N_u)}(\| \, j : j \in N_v : d_{vj} \cdot \delta)) \\
= & \quad \{ \text{ (12), (11), (8) } \} \\
& (+K : \emptyset \subset K \subset N_u : \rho_{\boldsymbol{ca}(I_u)}(\lambda^{I_u}_{m_u}(|k : k \in K : e_{uk})) \cdot \rho_{\boldsymbol{ca}(I_u)}(\lambda^{I_u}_{m_u(K)}(\|i : i \in N_u \setminus K : e_{ui} \cdot \delta))) \\
& \;\; + \rho_{\boldsymbol{ca}(I_u)}(\lambda^{I_u}_{m_u}(|i : i \in N_u : e_{ui})) \cdot \delta \\
= & \quad \{ \text{ Property 10.26, (4), expansion } \} \\
& \rho_{\boldsymbol{ca}(I_u)}(\lambda^{I_u}_{m_u}(\| \, i : i \in N_u : e_{ui} \cdot \delta))
\end{aligned}$$

This derivation proves property (9) and, hence, property (6) for the case that $n_u > 0$. As mentioned, property (7) can be proven by means of a symmetrical argument.

Finally, it suffices to combine the results to prove that

$$\begin{aligned}
& I' \cap causes(t) = \emptyset \wedge \\
& \quad (\mathrm{ACP}^c_\lambda + \mathrm{RN} + \mathrm{SC})(AC(C, A), \uplus) \vdash \\
& \qquad t \cdot \delta = \rho_{\boldsymbol{ca}(I')}(\lambda^{I'}_{m'}((\| \, i : i \in N_u : d_{ui} \cdot \delta) \parallel (\| \, j : j \in N_v : d_{vj} \cdot \delta))),
\end{aligned}$$

which completes the proof of the theorem.

□

10.4 Concluding remarks

In this section, it has been shown how to obtain a (branching-time) non-interleaving, partial-order process algebra that incorporates several important characteristics of the Petri-net formalism of Section 4. The algebra consists of non-terminating serializable processes and contains two causality mechanisms, one based on sequential composition and communication and one based on the causal state operator and cause addition. It has been shown that all non-terminating serializable processes with bounded behavior can be specified using only the latter causality mechanism (Theorem 10.27). That is, each such process has an algebraic specification in the style of Petri-net theory.

It is interesting to look briefly at the generalization of Theorem 10.27 to a setting with a simple form of recursion, namely iteration. It appears that it is possible to generalize Theorem 10.27 to some extent. This generalization is inspired by Definition 6.10 (Algebraic semantics for labeled P/T nets) and Theorem 6.12. Definition 6.10 gives an algebraic expression for any labeled P/T net. Theorem 6.12 proves that the (step) semantics of a labeled P/T net and its algebraic semantics are identical. It is interesting to observe the similarity between the algebraic term in Definition 6.10 and the canonical term in Theorem 10.27. Assuming that cause-addition operators are used to make causes explicit in the expression of Definition 6.10, the only difference between the two terms is that the action-prefix constants in the term in Theorem 10.27 are replaced by prefix-iterations in the term in Definition 6.10. It is our claim that any non-terminating serializable process in set $\mathcal{C}_{\mathrm{ns}}(C, A)$ of Section 10.1 is step bisimilar to a canonical term consisting of a parallel composition of prefix-iterations and restricted by means of a causal state operator where causes added to enforce orderings are hidden. However, it is still an open problem whether any closed term in $\mathcal{C}_{\mathrm{ns}}(C, A)$ is also derivably equivalent to such a term in canonical form, which is a more general result. Recall that Theorem 10.27 is proven by means of induction on the structure of basic terms. This technique does not carry over to a setting with iteration.

A few final remarks are in order. First, it appears to be possible to prove results very similar to Theorem 10.27 for theories concerning processes with bounded behavior that are not necessarily non-terminating and serializable. The interested reader is referred to [4] for more details. Second, it remains an interesting topic for future study to investigate to what extent Theorem 10.27 can be generalized to a setting with linear or even general recursion. Third, an interesting topic for future research is to study axiomatizations of non-interleaving, partial-order process algebras in the style proposed in this section.

11 Conclusions

Concluding Remarks In this chapter, we have studied two major themes. The first theme is the development of a partial-order ACP-style algebraic theory. It has been shown that the standard ACP-style algebraic framework is sufficiently flexible to develop a partial-order theory based on the semantic equivalence of step bisimilarity. The communication mechanism of ACP can be used to specify which actions are causally independent and can, thus, occur concurrently. In addition, the notions of a (non-)interleaving equational theory and a (non-)interleaving process algebra have been formalized. It has been argued that the characterizations interleaving versus non-interleaving and total-order versus partial-order for process-algebraic theories are orthogonal. In particular, it has been shown that the partial-order theory developed in Section 5 is an interleaving theory.

The second major theme is the study of concepts known from Petri-net theory, one of the most well-known approaches to modeling and analyzing concurrent systems by means of partial orders, in a process-algebraic framework. In particular, the causality mechanism of Petri-net theory has been adopted in the partial-order framework of Section 5. The resulting algebraic theory allows for specifications in the style of Petri-net theory. It has been shown that the theory can be used to reason in a purely equational way about the equivalence of labeled P/T nets. In addition, the relation between the causality mechanism adopted from Petri-net theory and the standard algebraic causality mechanism has been investigated. It has been shown that the Petri-net mechanism can replace the standard algebraic mechanism to some extent.

The main contribution of this chapter is that it improves our understanding of some of the most important concepts that play a role in describing and analyzing the behavior of concurrent systems, namely communication and causality.

Related work The literature on concurrency theory contains several approaches to combining some Petri-net formalism with some process-algebraic theory. One line of research is concerned with the translation of process-algebraic terms into Petri nets. The aim of such an approach is to provide the algebraic theory via a Petri-net semantics with a partial-order semantics. The most well-known example of this approach is the Petri Box Calculus of [15, 16]. Other examples of this line of research are [19, 21, 29, 30, 45, 49, 59]. In this chapter, the converse approach is pursued. In particular, in Section 6.3, a translation from labeled P/T nets into algebraic terms is given. Other examples of this approach are [20, 22]. The main difference between the approach taken in this chapter and other approaches is that this chapter emphasizes equational reasoning, whereas other approaches often emphasize the semantic framework.

Some other authors have taken an approach to modeling and analyzing concurrent systems based on an explicit representation of concurrent-system behavior in terms of partial orders. Examples of such frameworks can be found in [25, 31, 37, 54]. Other well-known partial-order based theories are trace theory as described in [41] and the theory of event structures presented in [47, 60].

Future research Several topics for future study have already been mentioned earlier in this chapter. At this point, we mention two other general directions for future research. First, it is interesting to study the topics addressed in this chapter in a framework based on some other partial-order semantics. There are partial-order semantics that capture causalities more accurately than the step semantics used in this chapter (see, for example, [53], for more details). Second, it would be interesting to investigate the application of the concepts studied in this chapter in the development and analysis of complex concurrent systems. Such applications require the extension of the framework developed in this chapter with, at least, an abstraction mechanism and a way to reason about data. Abstraction can be included following the approach of [8, Chapter 3]. Extensions of Petri-net theory with data can be found in, for example, [34, 38]. An extension of ACP-style process algebra with data is the theory μCRL described in [32, 33].

References

1. L. Aceto, W.J. Fokkink, and C. Verhoef. Structured Operational Semantics. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*. Elsevier Science, Amsterdam, The Netherlands. To appear.

2. J.C.M. Baeten. The Total Order Assumption. In S. Purushothaman and A. Zwarico, editors, *First North American Process Algebra Workshop, Proceedings*, Workshops in Computing, pages 231–240, Stony Brook, New York, USA, August 1992. Springer, Berlin, Germany, 1993.

3. J.C.M. Baeten and J.A. Bergstra. Global Renaming Operators in Concrete Process Algebra. *Information and Computation*, 78(3):205–245, 1988.

4. J.C.M. Baeten and J.A. Bergstra. Non Interleaving Process Algebra. In E. Best, editor, *CONCUR '93, 4th. International Conference on Concurrency Theory, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 308–323, Hildesheim, Germany, August 1993. Springer, Berlin, Germany, 1993.

5. J.C.M. Baeten and C. Verhoef. Concrete Process Algebra. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 4, *Semantic Modelling*, pages 149–268. Oxford University Press, Oxford, UK, 1995.

6. J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1990.

7. Bakkenist Management Consultants. *ExSpect 6 User Manual*, 1997. Information available at `http://www.exspect.com/`

8. T. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. PhD thesis, Eindhoven University of Technology, Department of Mathematics and Computing Science, Eindhoven, The Netherlands, December 1998.

9. T. Basten and M. Voorhoeve. An Algebraic Semantics for Hierarchical P/T Nets. Computing Science Report 95/35, Eindhoven University of Technology, Department of Mathematics and Computing Science, Eindhoven, The Netherlands, December 1995. An extended abstract appeared as [10].

10. T. Basten and M. Voorhoeve. An Algebraic Semantics for Hierarchical P/T Nets (extended abstract). In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995, 16th. International Conference, Proceedings*, volume 935 of *Lecture Notes in Computer Science*, pages 45–65, Torino, Italy, June 1995. Springer, Berlin, Germany, 1995.

11. J.A. Bergstra, I. Bethke, and A. Ponse. Process Algebra with Iteration and Nesting. *The Computer Journal*, 37(4):241–258, 1994.

12. J.A. Bergstra, W.J. Fokkink, and A. Ponse. Process Algebra with Recursive Operations. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*. Elsevier Science, Amsterdam, The Netherlands. To appear.

13. J.A. Bergstra and J.W. Klop. Process Algebra for Synchronous Communication. *Information and Control*, 60(1–3):109–137, 1984.

14. J.A. Bergstra and J.W. Klop. Algebra of Communicating Processes with Abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

15. E. Best, R. Devillers, and J.G. Hall. The Box Calculus: A New Causal Algebra with Multi-label Communication. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 21–69. Springer, Berlin, Germany, 1992.

16. E. Best, R. Devillers, and M. Koutny. A Consistent Model for Nets and Process Algebras. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*. Elsevier Science, Amsterdam, The Netherlands. To appear.

17. E. Best and C. Fernández C. *Nonsequential Processes: A Petri Net View*, volume 13 of *EATCS monographs on Theoretical Computer Science*. Springer, Berlin, Germany, 1988.

18. E. Best and B. Grahlmann. *PEP: Documentation and User Guide*. Universität Hildesheim, Institut für Informatik, 1995. Information available at `http://theoretica.informatik.uni-oldenburg.de/~pep`.

19. G. Boudol and I. Castellani. Flow Models of Distributed Computations: Three Equivalent Semantics for CCS. *Information and Computation*, 114(2):247–314, 1994.

20. G. Boudol, G. Roucairol, and R. de Simone. Petri Nets and Algebraic Calculi of Processes. In G. Rozenberg, editor, *Advances in Petri Nets 1985*, volume 222 of *Lecture Notes in Computer Science*, pages 41–58. Springer, Berlin, Germany, 1985.

21. P. Degano, R. De Nicola, and U. Montanari. A Distributed Operational Semantics for CCS Based on Condition/Event Systems. *Acta Informatica*, 26(1/2):59–91, October 1988.

22. C. Dietz and G. Schreiber. A Term Representation of P/T Systems. In R. Valette, editor, *Application and Theory of Petri Nets 1994, 15th. International Conference, Proceedings*, volume 815 of *Lecture Notes in Computer Science*, pages 239–257, Zaragoza, Spain, June 1994. Springer, Berlin, Germany, 1994.

23. E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics*. Springer, Berlin, Germany, 1990.

24. W.J. Fokkink. A Complete Equational Axiomatization for Prefix Iteration. *Information Processing Letters*, 52(6):333–337, 1994.

25. J.L. Gischer. The Equational Theory of Pomsets. *Theoretical Computer Science*, 61(2,3):199–224, November 1988.

26. R.J. van Glabbeek. The Linear Time – Branching Time Spectrum I: The Semantics of Concrete, Sequential Processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*. Elsevier Science, Amsterdam, The Netherlands. To appear.

27. R.J. van Glabbeek. The Linear Time – Branching Time Spectrum. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension, Proceedings*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297, Amsterdam, The Netherlands, August 1990. Springer, Berlin, Germany, 1990.

28. R.J. van Glabbeek. The Linear Time – Branching Time Spectrum II: The Semantics of Sequential Systems with Silent Moves (extended abstract). In E. Best, editor, *CONCUR '93, 4th. International Conference on Concurrency Theory, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81, Hildesheim, Germany, August 1993. Springer, Berlin, Germany, 1993.

29. R.J. van Glabbeek and F.W. Vaandrager. Petri Net Models for Algebraic Theories of Concurrency. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *PARLE Parallel Architectures and Languages Europe,* volume II*, Parallel Architectures*, volume 259 of *Lecture Notes in Computer Science*, pages 224–242, Eindhoven, The Netherlands, June 1987. Springer, Berlin, Germany, 1987.

30. U. Goltz. CCS and Petri Nets. In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes, LITP Spring School on Theoretical Computer Science, Proceedings*, volume 469 of *Lecture Notes in Computer Science*, pages 334–357, La Roche Posay, France, April 1990. Springer, Berlin, Germany, 1990.

31. J. Grabowski. On Partial Languages. *Fundamenta Informaticae*, 4(2):427–498, 1981.

32. J.F. Groote and A. Ponse. The Syntax and Semantics of μCRL. In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, *Algebra of Communicating Processes 1994*, Workshops in Computing, pages 26–62, Utrecht, The Netherlands, May 1994. Springer, Berlin, Germany, 1995.

33. J.F. Groote and M.A. Reniers. Algebraic Process Verification. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*. Elsevier Science, Amsterdam, The Netherlands. To appear.

34. K.M. van Hee. *Information Systems Engineering: A Formal Approach*. Cambridge University Press, Cambridge, UK, 1994.

35. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice–Hall International, London, UK, 1985.

36. T.W. Hungerford. *Algebra*. Holt, Rinehart, and Winston Inc., New York, USA, 1974.

37. W.P.M. Janssen. *Layered Design of Parallel Systems*. PhD thesis, University of Twente, Department of Computer Science, Enschede, The Netherlands, 1994.

38. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use,* volume 1*, Basic Concepts*. EATCS monographs on Theoretical Computer Science. Springer, Berlin, Germany, 1992.

39. K. Jensen, S. Christensen, P. Huber, and M. Holla. *Design/CPN: A Reference Manual*. Meta Software Corporation, 1991. Information available at `http://www.daimi.au.dk/designCPN`.

40. S.C. Kleene. Representation of Events in Nerve Nets and Finite Automata. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, number 34 in Annals of Mathematics Studies, pages 3–41. Princeton University Press, Princeton, New Jersey, USA, 1956.

41. A. Mazurkiewicz. Trace Theory. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course*, volume 255 of *Lecture Notes in Computer Science*, pages 279–324, Bad Honnef, Germany, September 1986. Springer, Berlin, Germany, 1987.

42. A.R. Meyer. Report on the 5th. International Workshop on the Semantics of Programming Languages in Bad Honnef. In *Bulletin of the EATCS,* No. 27, pages 83–84. European Association for Theoretical Computer Science, 1985.

43. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, Berlin, Germany, 1980.

44. R. Milner. *Communication and Concurrency*. Prentice–Hall International, London, UK, 1989.

45. U. Montanari and D. Yankelevich. Combining CCS and Petri Nets via Structural Axioms. *Fundamenta Informaticae*, 20(1–3):193–229, May 1994.

46. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

47. M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part I. *Theoretical Computer Science*, 13(1):85–108, 1981.

48. M. Nielsen and P.S. Thiagarajan. Degrees of Non-Determinism and Concurrency: A Petri Net View. In M. Joseph and R. Shyamasundar, editors, *Foundations of Software Technology and Theoretical Computer Science, 4th. Conference, FST&TCS4, Proceedings*, volume 181 of *Lecture Notes in Computer Science*, pages 89–117, Bangalore, India, December 1984. Springer, Berlin, Germany, 1984.

49. E.-R. Olderog. *Nets, Terms and Formulas*, volume 23 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1991.

50. D. Park. Concurrency and Automata on Infinite Sequences. In P. Deussen, editor, *Theoretical Computer Science, 5th. GI-Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, Karlsruhe, Germany, March 1981. Springer, Berlin, Germany, 1981.

51. J.L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1981.

52. C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, Germany, 1962. In German.

53. L. Pomello, G. Rozenberg, and C. Simone. A Survey of Equivalence Notions for Net Based Systems. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 410–472. Springer, Berlin, Germany, 1992.

54. V.R. Pratt. Modeling Concurrency with Partial Orders. *International Journal of Parallel Programming*, 15(1):33–71, February 1986.

55. W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS monographs on Theoretical Computer Science*. Springer, Berlin, Germany, 1985.

56. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science. Advances in Petri Nets*. Springer, Berlin, Germany, 1998.

57. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets II: Applications*, volume 1492 of *Lecture Notes in Computer Science. Advances in Petri Nets*. Springer, Berlin, Germany, 1998.

58. P. Sewell. Nonaxiomatisability of Equivalences over Finite State Processes. *Annals of Pure and Applied Logic*, 90(1–3):163–191, 1997.

59. D. Taubner. Representing CCS Programs by Finite Predicate/Transition Nets. *Acta Informatica*, 27(6):533–565, May 1990.

60. G. Winskel. An Introduction to Event Structures. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Proceedings*, volume 354 of *Lecture Notes in Computer Science*, pages 364–397, Noordwijkerhout, the Netherlands, May/June 1988. Springer, Berlin, Germany, 1989.