A Consistent Model for Nets and Process Algebras

Eike Best[1], Raymond Devillers[2], and Maciej Koutny[3]

[1] Fachb. Inf., Carl von Ossietzky Universität, D–26111 Oldenburg, Germany
[2] Départ. d'Inform., Université Libre de Bruxelles, B-1050 Bruxelles, Belgium
[3] Dept. of Comp. Sci., University of Newcastle, Newcastle upon Tyne NE1 7RU, U.K.

Abstract. This chapter addresses a range of issues that arise when process algebras and Petri nets are combined together; in particular, compositionality of structure and behaviour, refinement and equivalences. A generic algebra of nets and process expressions is defined and equipped with two, fully consistent, types of semantics: a Petri net semantics based on step sequences and causal partial orders, and a structured operational semantics based on a system of derivation rules. A concrete example of this algebraic framework is the Petri Box Calculus (PBC) which is used as a vehicle to convey the basic ideas contained in this chapter.

1 Introduction

This chapter is about combining two well studied theories of concurrency: *process algebras* and *Petri nets*. Among the advantages of process algebras, we can identify the following:

- They allow the study of connectives directly related to real programming languages.
- They are compositional by definition, i.e., they enable one to compose larger systems from smaller ones in a structured way.
- They come with a variety of concomitant and derived logics which may facilitate reasoning about important properties of systems.
- They support a variety of algebraic laws which may be used to manipulate systems, either to refine them or to prove them correct with respect to some specification.

The main advantages of Petri nets, on the other hand, are the following:

- They sharply distinguish between states and activities (the latter being defined as changes of state), through the distinction made between places (local states) and transitions (local activities).
- Global states and global activities are not basic notions, but are derived from their local counterparts, which is both intuitive and suits well the description of a distributed system as the sum of its local parts; in particular, it allows the description of concurrent behaviour by partial orders.
- While being formal, Petri nets also come with a graphical representation which is easy to comprehend and has therefore some wide appeal for practitioners.
- By their representation as bipartite graphs, Petri nets have useful links both to graph theory and to linear algebra, which can be exploited for the verification of systems.

This chapter describes an approach combining a fundamental (but restricted) class of Petri nets and a basic (but again restricted) process algebra, striving to make the links between these two different models of concurrent systems as tight as it is ever possible. The investigation of the structural and behavioural aspects of these two basic models will provide a justification for the claim that there is indeed an extremely strong equivalence between them.

The basic process algebraic setup which lies at the origin of the model presented in this chapter is the Petri Box Calculus (PBC) [10] – itself a variant CCS [49,50] – which over the past few years has been developed as a model based on basic (place/transition) nets [4,9,18,23–25,37,43,44], as well as high level nets [11,12,26,35,42,47], and a formal underpinning of a concurrent programming notation and the associated specification and verification tool [5,13,14]. Several other process algebras, some of which will be discussed later in this chapter, have been subject to investigation [1,36,39,40]. The reason why CCS was chosen as a starting point was that it seemed to be most directly useful for describing the meaning of concurrent programming languages, using which most concurrent systems are actually implemented.

The basic Petri net theoretic setup is the place/transition net model [51,54]. Again, this is but one of the many net based models that have been investigated [17,41]. However, it is a fundamental one of which the others are extensions or variations, or both. Also, place/transition nets share with CCS-like process algebras the property of being rather directly applicable to dealing with concurrent programming languages [3] and concurrent algorithms [55,56].

The idea to combine a process algebra, such as CCS, with a Petri net model, such as place/transition nets, is not new and presumably was first expressed in 1974 [45]. In the years between this early memorandum and now, many articles and theses, as well as several books, have been written with exactly this connection in mind. In particular, [58] gives high-level Petri net semantics to a process algebra which is akin to CCS, [52] uses place/transition Petri net semantics of another process algebraic variant as a stepping stone in the design of concurrent systems, and in [40] place/transition nets are used as semantics of yet another process algebraic variant. All these setups differ in a subtle way one from another, and they are also limited in their various specific ways. For instance, recursion is absent in [40], and is treated non-denotationally and limited to the guarded case in [52,58]. More important, perhaps, than these differences, is the fact that none of the cited works has demonstrated a very strong equivalence ranging from the set of operators in each framework to the generated behaviour, and holding in full generality. It was the desire to obtain such an equivalence which distinguishes the work on the box algebra from other already mentioned.

In the course of searching for the desired strong connections, the authors of this chapter have experienced several surprises (apart from the very fact that it can be found). One germ to such a surprise was the separation, already proposed in PBC, of the CCS synchronisation operator into parallel composition and (the rest of the) synchronisation. This has eventually led to a general classification of all operators into place-based ones and transition-based ones.

An advantage of this classification is that all operators in the first class can be treated similarly, and all operators in the second class can be treated similarly as well. An exception to this rule is recursion, which can be seen as belonging to the first class, but needs a more

involved treatment than the other operators of that class. Also iteration needs a very careful consideration, although ultimately all these operators are based on a single net-theoretic operation: transition refinement. (It may seem odd to the reader that place-based operators should be uniformly based on a transition-related operation, but the simple explanation is that the environment of a transition, which is a set of places, plays an important role in such a definition.) Likewise, all operators in the second class are based on a single operation: relabelling. This is not actually a Petri net indigenous operation, but rather, it refers to the transition labelling of a net. It can be defined in a sufficiently generic way so that all operators of the second class are special cases. As a matter of fact, we will integrate refinement and relabelling into only one operation, called *net refinement*.

Viewing all the operators as incarnations of such a powerful operation leads to a further generalisation: refining transitions in a net can be viewed as a function from (tuples of) nets to nets. For instance, sequential composition can be viewed as a very simple two-transition net which takes two nets as arguments and creates another one (their sequence) as a result. Nets used in this way as functions from (tuples of) nets to nets will be called *operator boxes*. In a typical process algebra, a handful of operators (each corresponding to a particular operator box) is considered. However, thanks to the truly generic setup described above, we may now formulate all the results in terms of a very large class of operator boxes, all of which could be interpreted as process-algebraic operators. It is one of the principal outcomes of this approach to state precisely which of such nets can, and which cannot, be viewed as good operators.

The main criterion for this, in fact, will be whether or not the desired equivalence holds. This is a so-called behavioural equivalence. What we desire is that the entire reachability graph[1] is exactly the same, whether one formalises it in terms of process algebra or in terms of nets. In order to achieve this equivalence, we will introduce a net-independent behavioural semantics of process algebra in the form of a structured operational semantics [53], and compare it with the net-induced semantics. The equivalence of both, for a large class of operator boxes, is the central result of this chapter, and will be stated in section 8.4.

In this chapter, all and any proofs are omitted. The reader is referred to [8] or to the literature.

1.1 Petri nets and process algebras

To explain the ideas behind our approach, we will outline a process algebra, whose name – Petri Box Calculus (PBC) – arises from its original Petri net semantics [7, 10]. PBC combines a number of features taken from existing process algebras, notably COSY [40], CCS [49,50], SCCS [50], (T)CSP [38,39] and ACP [1,2,27].[2] But there are also some differences with each of these process algebras since PBC has been designed with two specific objectives in mind:

– To support a compositional Petri net semantics, together with an equivalent structured operational semantics (SOS) [53].

[1] That is, the set of reachable states, together with the information as to which of them are reachable from which others.

[2] Although PBC has not been specifically designed as a common generalisation of the previously mentioned existing formalisms, it turns out that these can be encoded in the same framework; we will show this in section 9.

– To provide a sound and flexible basis for a compositional semantics of high level concurrent specification and programming languages with full data and control features.

Since Milner's CCS can be viewed as a process algebra directly inspiring the design of PBC, we shall use it as a reference model when explaining the specific features of PBC, here and in section 2.

Petri nets The graph of a Petri net N describes the structure of a dynamic system. The *behaviour* of such a system is defined with respect to a given starting marking (global state) of the graph, called the *initial marking*. In general, a *marking* M of N is a function from the set of places of N (local states) to the set of natural numbers $\mathbf{N} = \{0, 1, 2, \ldots\}$; if $M(s) = n$ then we say that s contains n *tokens*. A transition t (representing some activity) is *enabled by* M if each input places of t contains at least one token. If an enabled transition *occurs*, then this is tantamount to the following change of marking: a token is subtracted from each input place of t; a token is added to each output place of t; and no other places are affected.

We use the notation $M \xrightarrow{t} M'$ (or $(N, M) \xrightarrow{t} (N, M')$, to emphasize N) in order to express that M enables t and M' arises out of M according to the rule just described; M' is then said to be *directly reachable* from M, and the reflexive and transitive closure of this relation gives the overall *reachability* relation. A marking is called *safe* if no place contains more than one token, and a marked net is called *safe* if every marking reachable from the initial marking is safe.

Figure 1 shows an unmarked Petri net, N, and two marked Petri nets, (N, M_0) and (N, M_1). Notice that the underlying net (graph) is the same in all cases, only the markings are different. Moreover, the unmarked net could as well be interpreted as a marked net, namely as one which has the 'empty' marking assigning zero tokens to each place.

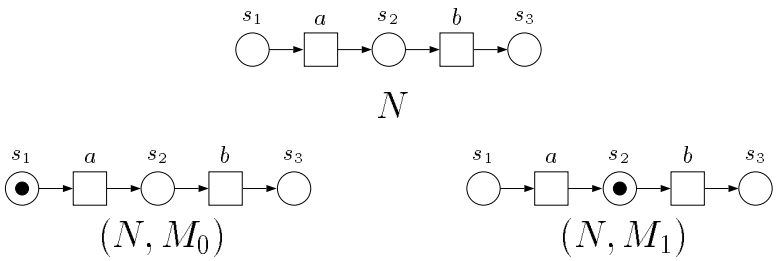


Fig. 1. A Petri net, and the same net with markings M_0 and M_1.

Formally, the net N of figure 1 is a triple (S, T, W) where S is the set of its places, $S = \{s_1, s_2, s_3\}$, T is the set of its transitions, $T = \{a, b\}$, and W assigns a number (in this case, only 0 or 1) to pairs (s, t) and (t, s), where $s \in S$ and $t \in T$, depending on whether an arrow leads from s to t or from t to s, respectively. For instance, $W(s_1, a) = 1$ and $W(b, s_2) = 0$ (i.e., 1 stands for 'arrow' and 0 stands for 'no arrow'). The marking M_0 shown

is a function satisfying $M_0(s_1) = 1$, $M_0(s_2) = 0$ and $M_0(s_3) = 0$, while M_1 is a function satisfying $M_1(s_1) = 0$, $M_1(s_2) = 1$ and $M_1(s_3) = 0$. The marking M_0 enables transition a which may occur; if it does, the resulting marking is M_1. Thus, $(N, M_0) \xrightarrow{a} (N, M_1)$.

Relationship between structure and behaviour In figure 1, N describes a sequence of two transitions, a and b, and it is therefore related to the CCS expression $a.(b.\mathsf{nil})$. But which of the three nets should correspond to the semantics of $a.(b.\mathsf{nil})$? The answer is clear: (N, M_0). The reason is that only this net has the same *behaviour* as $a.(b.\mathsf{nil})$. The net N by itself has the empty behaviour, and in (N, M_1), b can occur as the first action, but not a as required by $a.(b.\mathsf{nil})$. Consider now what happens when the action a is executed in either model. In CCS, the expression $a.(b.\mathsf{nil})$ is transformed into $b.\mathsf{nil}$. In the net domain, the marked net (N, M_0) is transformed into the net (N, M_1) with the same underlying unmarked net N. But if we were to define a Petri net corresponding to the expression $b.\mathsf{nil}$, then we would not even think of taking the net (N, M_1); rather, a natural choice would be (N, M_1) after deleting s_1 and a.

This example shows that the ways of generating behaviour in net theory and in CCS do not directly correspond to each other. In the latter, the structure of an expression may change, while in the former, only markings change but the structure of a net remains the same through any execution. The difference is a fundamental one, and stems from the underlying ideas of modelling systems. In net theory, when a transition has occurred, we may still recover its presence from the static structure of the system, even though it may never again be executed; there is thus a sharp distinction between the static and the dynamic aspects of a system. In CCS, when an action has occurred, we may forget it (unless it is in a loop), precisely because it may never be executed, and hence we may safely change the structure of the expression (expressions with loops may also be changed, for instance by making them longer); there is thus a very intimate relation between the static and dynamic aspects of process expressions.

Static and dynamic expressions The semantics of PBC will respect the division between static and dynamic aspects that can be found in the Petri net theory. The basic idea will be to introduce into the syntax of expressions a device that models markings. Thus, we will define expressions 'without markings' (called static expressions) and expressions 'with markings' (called dynamic expressions): static expressions correspond to unmarked nets, while dynamic expressions correspond to marked nets. Below, E and F will denote static expressions, and G and H will denote dynamic expressions.

For example, $E = a;b$ is a static PBC expression corresponding to the unmarked net N in figure 1. Making expressions dynamic consists of overbarring or underbarring (parts of) them. For instance, $\overline{E} = \overline{a;b}$ is a dynamic expression which corresponds to the same net as E, but with a marking that lets E be executed from its beginning, i.e., to the 'initial marking' of E. The net corresponding to $\overline{E}$ is (N, M_0). We will also consider – syntactically legal and meaningful – dynamic PBC expressions like $G = \underline{a};b$. It corresponds to the same underlying expression E, but in a state in which its first action a has just been executed, i.e., it shows the instant at which the final state of a has just been produced. Such a G corresponds to (N, M_1).

Structural similarity on expressions This syntactic device of overbarring and underbarring introduces what could, at first glance, be seen as a difficulty. For consider $E = a;b$ in a state in which its second part, b, is just about to be executed, $G' = a;\overline{b}$. Which marked net should this dynamic expression correspond to? There is only one reasonable answer to this question, namely the net (N, M_1) in figure 1 which, as already observed, corresponds to $G = \underline{a};b$. Thus, in general, we may have a many-to-one relationship between dynamic expressions and marked nets. Let us use the symbol $\equiv$ to relate dynamic expressions which are not necessarily syntactically equal, but are in the same relationship as G and G'. Then $G \equiv G'$ can be viewed as expressing that 'the state in which the first component of a sequence has terminated is the same as the state in which the second component may begin its execution'. As another example, we observe that $\overline{E} = \overline{a;b}$ also has a dynamic expression which is equivalent but not syntactically equal, namely $H = \overline{a};b$. And $\overline{E} \equiv H$ can be viewed as saying that 'the initial state of a sequence is the initial state of its first component'.

We do not see this many-to-one relationship as merely incidental, to be overcome, perhaps, by a better syntactic device than that of overbarring and underbarring subexpressions. Instead, we view it as evidence of a fundamental difference between Petri nets and process algebras: expressions of the latter come with in-built structure, while Petri nets do not. For instance, $E = a;b$ is immediately recognised as a sequence of a and b. In the example of figure 1, one may also recognise the structure to be a sequence. However, this is entirely due to the simplicity of the net. For the graph of an arbitrary net it is far from obvious how it may be seen as constructed from smaller parts, while it is always clear, for any well-formed expression in any process algebra, how it is made up from subexpressions. The fact that an expression of a process algebra always reveals its syntactic structure, has some highly desirable consequences. For example, it is often possible to conduct an argument 'by syntactic induction'; one may build proof systems 'by syntactic definition' around such an algebra; and the availability of operators for the modular construction of systems is usually appreciated by practitioners. On the other hand, one of the disadvantages of such a syntactic view is that some effort has to be invested into the definition of the behaviour of expressions. While this can usually be done inductively, it is still necessary to go through all operators one by one. A non-structured model such a Petri nets has an advantage in this respect; its behaviour (with respect to a marking) is defined by a single rule, namely the transition occurrence rule, which covers all the cases.

In this spectrum, the box algebra attempts to cover a middle position, and its expressions come with an in-built structure. On the other hand, the behavioural rules of (dynamic) expressions are, in fact, the Petri net transition rule in disguise. The necessity to consider equivalences, such as the *structural similarity* $\equiv$, can be viewed as being the price that has to be paid for being able to combine advantages of both Petri nets and process algebras.

1.2 Outline of this chapter

Section 2 is introductory, explaining the main distinctions between CCS and PBC (and thus also the generic box algebra). In section 3, the Petri net framework used in the rest of the chapter is presented. A general simultaneous refinement and relabelling operation, or net refinement, is defined in section 4. Section 5 describes a specific fixpoint theory developed in order to solve recursive equations on nets. In section 6, an analysis of nets obtained through refinements and recursive equations is conducted, based on the structural notion

of an S-invariant. In sections 7 and 8, the two approaches to defining the semantics of the generic box algebra are described, and the consistency results between them are stated. In section 9, the applicability of the theory is demonstrated by giving a compositional, Petri net and SOS, semantics for the PBC, and also by sketching a similar treatment for CCS, CSP and COSY. The last section contains brief explanations why the box algebra can readily yield a compositional semantics for an imperative concurrent programming language.

1.3 Formal notation

We use the standard mathematical notation throughout. Some particular notations are recalled below.

A *partial order* is a pair (X, R) where X is a set and $R \subseteq X \times X$ is a reflexive ($id_X \subseteq R$), transitive ($R \circ R \subseteq R$) and antisymmetric ($R \cap R^{-1} = id_X$) relation.

A *multiset* over a set A is a function $\mathsf{m}: A \to \mathbf{N}$. m is *finite* if the set of all $a \in A$ such that $\mathsf{m}(a) > 0$ is finite. The *cardinality* of a finite m is the number $|\mathsf{m}| = \sum_{a \in A} \mathsf{m}(a)$. The operations of *sum* and (nonnegative) *difference* for the multisets over A are defined so that, for every $a \in A$,

$$\begin{array}{ll} \text{Sum:} & (\mathsf{m} + \mathsf{m}')(a) = \mathsf{m}(a) + \mathsf{m}'(a) \\ \text{Difference:} & (\mathsf{m} - \mathsf{m}')(a) = \max(\mathsf{m}(a) - \mathsf{m}'(a), 0)\ . \end{array}$$

Note that if m and m' are finite then so are $\mathsf{m} + \mathsf{m}$ and $\mathsf{m} - \mathsf{m}'$. Multiset *inclusion*, $\mathsf{m} \subseteq \mathsf{m}'$, is defined by $\mathsf{m}(a) \leq \mathsf{m}'(a)$, for all $a \in A$. In the examples, we will write, e.g., $\{a, a, b\}$ for the multiset defined by $\mathsf{m}(a) = 2, \mathsf{m}(b) = 1$, and $\mathsf{m}(c) = 0$ for all $c \in A \backslash \{a, b\}$. The set of all finite multisets over A is denoted by $\mathsf{mult}(A)$.

2 CCS and PBC

In this section we further compare the basic differences between CCS and PBC. In doing so, we also sketch the syntax and semantics of PBC which will be the main instance of the generic model described in the subsequent sections.

Prefixing and sequential composition Instead of expressing sequential behaviour by CCS prefixing, PBC uses a true sequential operator, $E; F$, meaning that E is to be executed before F. The sequence can be simulated in CCS by a combination of prefixing, parallel operator and synchronisation. One reason for allowing full sequential composition is that in a majority of imperative languages, this is one of the basic operations. Another reason is that in PBC, the semantics of the full sequential composition is no more complicated than that of prefixing.

The CCS's nil process is no longer necessary in PBC. To see why, we compare the CCS way of deriving the behaviour of $a.(b.\mathsf{nil})$, and the PBC way of deriving the behaviour of $\overline{a;b}$:

$$\begin{array}{lccccccccc} \text{In CCS:} & a.(b.\mathsf{nil}) & & \xrightarrow{a} & & b.\mathsf{nil} & & \xrightarrow{b} & & \mathsf{nil}. \\ \text{In PBC:} & \overline{a;b} & \equiv & \overline{a};b & \xrightarrow{a} & \underline{a};b & \equiv & a;\overline{b} & \xrightarrow{b} & a;\underline{b} \quad \equiv \quad \underline{a;b}\ . \end{array}$$

In both cases, the action sequence ab is derived, but the PBC way of deriving this behaviour is more symmetric while the CCS way is shorter. In CCS, the fact that 'execution has ended'

is expressed by the derivation of nil, while in PBC by the derivation of an expression which is completely underbarred, such as $\underline{a;b}$.

In general, if E is a static expression, then we call $\overline{E}$ 'the initial state of E', and $\underline{E}$ 'the final (or terminal) state of E'. However, this terminology is slightly deceptive, for there may be expressions E for which no behaviour leads from $\overline{E}$ to $\underline{E}$.

Synchronisation As CCS, the PBC model is based upon the idea that in a distributed environment all complex interactions can be decomposed into primitive binary interactions.

Suppose that two people, A and B, wish to communicate by exchanging a handshake. In CCS, one could represent the action 'A extends his hand towards B' by a symbol a; the action of 'B extends her hand towards A' by another symbol $\widehat{a}$ (the 'conjugate'[3] of a); and the fact that A and B are standing face to face by the symbol $\mid$, yielding the CCS expression $(a.\mathsf{nil}) \mid (\widehat{a}.\mathsf{nil})$. This expression has the following possible activities: (i) A may extend his hand and withdraw it, leaving B with the same possibility; (ii) B may extend her hand and withdraw it, then A does the same; and (iii) both A and B extend their hands which results in a *handshake*, labelled by τ, the 'silent action'. Using the existing Petri net semantics of CCS expressions (e.g., in [30,58]), these three possible activities could be represented in Petri net terms as shown on the right hand side of figure 2.

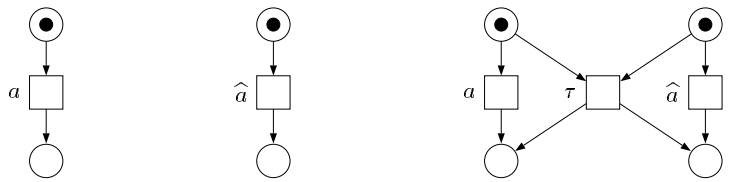


Fig. 2. Activity of a (left), activity of $\widehat{a}$ (middle) and activity of $a \mid \widehat{a}$ (right).

Now consider how one could describe a handshake between three people A, B and C. In terms of Petri nets, such an activity could be modelled by a three-way transition such as H in figure 3. A moment's reflection shows that a 3-way synchronisation transition like this cannot directly be modelled using the 2-way handshake mechanism of CCS; the reason is that τ cannot be synchronised with any other action (there is nothing like $\widehat{\tau}$ in CCS). While one might try to simulate the 3-way-handshake[4] using a series of binary handshakes, PBC proposes instead to extend the CCS framework so that a 3-way (and, in general, n-way) synchronisation could be expressed. One possibility, which is akin to the ACP approach [1, 2,27], might be to generalise the conjugation operation to a function or a relation that allows the grouping of more than just two actions; however, PBC takes a different approach, which

[3] In CCS, conjugation is usually denoted by overbarring; we use hatting instead, because overbarring has already been used to construct dynamic expressions.

[4] Although one might think that 3-way handshakes occur rarely in practice, we will argue in section 10 that multiway synchronisations make perfect sense in the modelling of programming languages, as does the particular way of describing them in PBC.

is still based on considering primitive synchronisations to be binary in nature. This works through the usage of structured actions, which we will describe next.

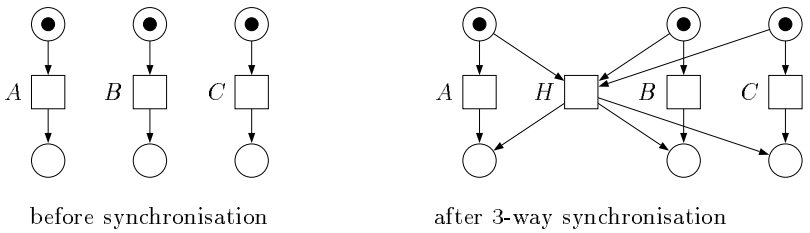


Fig. 3. Handshake between three people.

To understand the PBC way of describing multiway synchronisations, it is perhaps easiest to interpret a CCS pair of conjugate actions $(a, \widehat{a})$ in the original way [49] as denoting two links between two agents that fit together. In this view, a denotes a communication capability in search of a matching $\widehat{a}$, and similarly $\widehat{a}$ denotes a communication capability in search of a matching a. Once the communication takes place, i.e., a pair of a and $\widehat{a}$ is matched, this becomes a private synchronisation link (namely, τ) between two agents with no further externally visible communication capability. The PBC model extends this idea in the sense that the result of a synchronisation does not have to be completely internal or 'silent', but instead, may contain further communication capabilities that may be linked with other activities. This generalisation is achieved by allowing a transition to correspond to a whole (finite) set of communication capabilities, such as for instance, $\emptyset$, $\{a\}$, $\{\widehat{a}\}$, $\{a, b\}$, $\{\widehat{a}, b\}$, or even $\{a, \widehat{a}, b\}$. For instance, an activity $\{\widehat{a}\}$ may be synchronised with activity $\{a, b\}$ using the conjugate pair $(a, \widehat{a})$. This results in a synchronised activity which is not silent but rather, still has the communication capability $\{b\}$, as the pair $(a, \widehat{a})$ is internalised but b remains untouched. For instance, figure 4 describes a 3-way handshake in terms of the (Petri net view of) PBC. In the figure, a denotes 'A shakes hand with B', $\widehat{a}$ denotes 'B shakes hand with A', $\widehat{b}$ denotes 'B shakes hand with C' and b denotes 'C shakes hand with B'. Then, by the fact that B performs $\widehat{a}$ and $\widehat{b}$ simultaneously (expressed by the fact that $\{\widehat{a}, \widehat{b}\}$ is the label of a transition), the resulting activity 123 (in figure 4) describes a simultaneous handshake between all three people. Transition 12 describes the handshake only between A and B (with label $\{\widehat{b}\}$ and capability to link with C) and transition 23 describes the handshake only between B and C (with label $\{\widehat{a}\}$ and capability to link with A). Transition 123 can also be thought of either as a 2-way synchronisation between 1 and 23, or as a 2-way synchronisation between 12 and 3.

Let us now consider two actions $\{a, b\}$ and $\{\widehat{a}, b\}$. According to the PBC approach, two transitions, say 1 and 2, corresponding, respectively, to these actions can be synchronised using the conjugate pair $(a, \widehat{a})$. However, what synchronisation capability should the resulting transition 12 still possess? There are only two meaningful answers: either the set $\{b\}$ or the multiset $\{b, b\}$. The PBC model chooses the second alternative since only then we

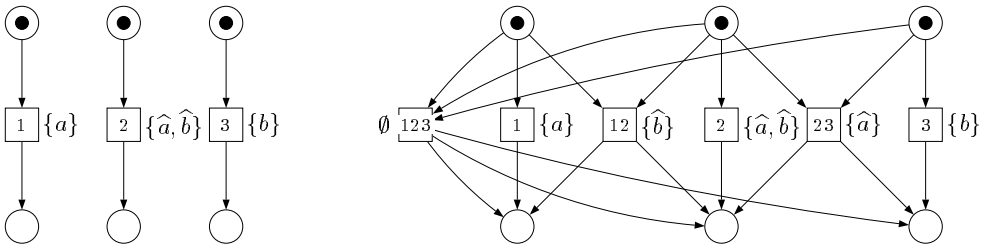


Fig. 4. 3-way handshake in terms of PBC.

obtain the (highly desirable and expected) property that the order of a series of consecutive synchronisations is irrelevant.

Therefore, PBC will use a set $\mathsf{A}_{\mathrm{PBC}}$ of primitive actions or *action particles*. There is a *conjugation* function $\widehat{\ }\colon \mathsf{A}_{\mathrm{PBC}} \to \mathsf{A}_{\mathrm{PBC}}$ with the properties of *involution* ($\widehat{\ } \circ \widehat{\ } = id_{\mathsf{A}_{\mathrm{PBC}}}$) and *discrimination* ($\widehat{a} \neq a$). This is the same basic setup as in CCS. However, unlike CCS, the PBC allows as the label of a transition any finite multiset over $\mathsf{A}_{\mathrm{PBC}}$. That is, the set of *labels* (or, for the more process algebraically inclined, *elementary expressions*, or, to emphasize the fact that more than one action may be combined in a single transition, the set of *multiactions*, or, if no confusion can arise with action particles, *actions*) is defined as $\mathsf{Lab}_{\mathrm{PBC}} = \mathsf{mult}(\mathsf{A}_{\mathrm{PBC}})$.

This specialises to the basic CCS framework as follows: the CCS a and $\widehat{a}$ correspond to $\{a\}$ and $\{\widehat{a}\}$ in PBC, respectively, and τ to $\emptyset$. No PBC action α, with $|\alpha| > 1$, has a direct CCS equivalent.

Parallel composition The CCS parallel composition performs the synchronisation for all possible conjugate pairs. For instance, the CCS expression $a.(a.(b.\mathsf{nil})) \,|\, \widehat{a}.(\widehat{a}.(\widehat{b}.\mathsf{nil}))$ creates five synchronisations (three of them being actually executable); the syntax of CCS does not allow one to express synchronisation using only the pair $(b, \widehat{b})$ but not the pair $(a, \widehat{a})$. For reasons that will become clear in section 10, it may be desirable to be able to say precisely which action names are being used for synchronisation.

The PBC approach therefore separates the synchronisation operator from parallel composition, $E \| F$, regarding synchronisation as a unary operator, E sy A where $A \subseteq \mathsf{A}_{\mathrm{PBC}}$. The latter makes it very similar to the restriction operator of CCS, but playing an opposite role: while synchronisation adds, restriction removes certain transitions.

Other operators Most of the remaining operators of the basic PBC (except iteration) are akin to their counterparts in CCS.

The choice between two expressions E and F, denoted by $E \,\square\, F$, specifies behaviour which is, in essence, the union of the behaviours of E and F. As its counterpart in CCS, $+$, the choice allows one to choose nondeterministically between two possible sets of behaviours. The iteration operator obeys the syntax $\langle E * F]$, where E is the looping part (whose execution

may be repeated arbitrarily many times), and F is the termination (which may be executed at most once, at termination).

Basic relabelling (simply called 'relabelling' in CCS) is defined with respect to a function f on action particles; $E[f]$ has all the behaviours of E, except that their constituents are relabelled according to f. Restriction, E rs A, is defined with respect to a set of action particles A, and has all the behaviours of E except those that involve the action particles in $A \cup \widehat{A}$, where $\widehat{A} = \{\widehat{a} \mid a \in A\}$. Scoping, $[A : E] = (E \text{ sy } A) \text{ rs } A$, is a derived operator; its importance stems from an application to the modelling of blocks in a programming language.

In terms of the Petri net semantics of PBC, it will be seen that the distinction between *control flow* operators and *communication interface* operators manifests itself in a very simple way: the latter (relabelling, synchronisation, restriction and scoping) modify transitions, while the former (sequence, choice, parallel composition and iteration) modify places.

Recursion allows process variables to be defined via equations. For example, $X \stackrel{\text{df}}{=} a; X$ specifies the behaviour of X to be an indefinite succession of the executions of a.

3 Petri Nets

There are several coherent behavioural semantics of Petri nets (e.g., [17,51,54]); in particular, concurrency semantics such as: trace semantics [48], step semantics [28], process semantics [6,33], and partial word semantics [34,57,59]. It is therefore often the case that a non-interleaving semantics of a process algebra is obtained by associating nets to process expressions. This technique has already been exploited for a number of existing process algebras [16,20,30,32,52,58], but in many cases only fragments of the theory have been successfully translated. Here we shall aim at a general translation which will also be fully compositional, due to the careful choice of the operators and very general mechanisms introduced to combine nets. This compositionality will be founded upon transition refinement, i.e., each operator on nets will be based on a net Ω whose transitions $t_1, t_2, \ldots$ are refined by the corresponding nets $\Sigma_1, \Sigma_2, \ldots$ in the process of forming a new net $\Omega(\Sigma_1, \Sigma_2, \ldots)$.

We shall employ labelled Petri nets, called *boxes*, whose interfaces are expressed by labellings of places and transitions. There are two main classes of boxes which we will be interested in, viz. plain boxes and operator boxes.

Plain boxes are the basic semantical objects of interest. They form the Petri net domain upon which various operators are defined, just as the process expressions form the domain of a process algebra upon which operators are defined. When giving the Petri net semantics of a process algebra (such as PBC or CCS), we will associate a plain box with every expression. The Σ_i above will be plain boxes.

Operator boxes are patterns (or functions) defining the ways of constructing new plain boxes out of given ones. When translating a process algebra into Petri nets, we will aim at associating an operator box with every operator of the process algebra. It is one of the characteristic features of our approach that the same type of nets – boxes – serve to describe two seemingly very different objects, namely the elements and the operators of the semantical domain. The Ω above will be an operator box.

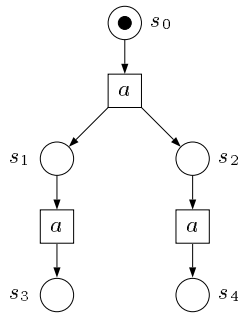


Fig. 5. A net corresponding to $\overline{a;(a\|a)}$.

We introduce the kind of nets we shall need using a simple example. A possible model for the dynamic expression $\overline{a;(a\|a)}$ might look like the (safe) net depicted in figure 5.[5] Clearly, the action a cannot be used to identify the transitions, since there are three of them, all corresponding to the same action a. As a result, to model static or dynamic expressions we need to employ Petri nets with transitions being labelled by action names. For Petri nets defining net operators, we shall also use transition labelling, but with labels corresponding to general relabellings, in order to allow combining together transitions coming from the composed nets. And, since action names may be treated as a special kind of (constant) relabellings, the latter may be used in full generality.

In figure 5, moreover, three different kinds of places can be identified. The place s_0 is special in the sense that it contains the token corresponding to the expression in its initial state. And, by symmetry, the places s_3 and s_4, when holding one token each, characterise the terminal state of the expression. The two remaining places, s_1 and s_2, may be considered as internal and contributing to intermediate markings corresponding to intermediate dynamic expressions. The different role of the places will be captured by a suitable labelling mechanism, with three possible values corresponding to the three different kinds of places.

Actions and relabellings We assume a set Lab of *actions* to be given. At this point Lab is an arbitrary set, but later in section 9 we shall consider Lab to be, for instance, the structured set $\mathsf{Lab}_{\mathrm{PBC}}$ of actions used in the PBC expressions. When executing a transition, or a set of concurrently enabled transitions, we shall be able to consider the action (or multiset of) actions with which it (or they) are labelled. A *relabelling* ϱ is a relation

$$\varrho \subseteq \mathsf{mult}(\mathsf{Lab}) \times \mathsf{Lab} \tag{1}$$

such that $(\emptyset,\alpha) \in \varrho$ if and only if $\varrho = \{(\emptyset,\alpha)\}$. The intuition behind a pair (Γ,α) belonging to ϱ is that it specifies a synchronisation followed by an interface change which can be applied to a (finite) group of transitions whose labels match the argument, i.e., the multiset of actions Γ. Since Γ, being a multiset, is an unordered object, the order of the transitions

[5] In the examples, we often change notation from $\{a\}$ as an action to a as an action for simplicity, and provided that synchronisation is not of concern.

in such a group does not matter, and we immediately obtain a kind of simple commutativity of the operation described by ϱ.

Three specific kinds of relabelling are of particular interest. A *constant* relabelling, $\varrho_\alpha = \{(\emptyset, \alpha)\}$, where α is an action in Lab, can be identified with α itself, so that we may consider the set of actions Lab to be embedded in the set of all relabellings. The *restriction* $\varrho_{\mathsf{Lab}'} = \{(\{\alpha\}, \alpha) \mid \alpha \in \mathsf{Lab}'\}$ only keeps the actions belonging to some set $\mathsf{Lab}' \subseteq \mathsf{Lab}$. And the *identity* relabelling, $\varrho_{id} = \{(\{\alpha\}, \alpha) \mid \alpha \in \mathsf{Lab}\}$ captures the 'keep things as they are' interface (non)change.

A non-constant relabelling will be called *transformational*; in such a case, the empty set will not be in its domain in order not to *create an action out of nothing*.

Relabellings have been defined in this generality for they are meant to encompass both normal labels Lab (then they are constant) and all communication interface operators such as restriction and synchronisation (then they are transformational). We will specify the transformational relabelling each time a concrete communication interface operator is defined.

3.1 Labelled nets and their semantics

A *labelled net* is a tuple

$$\Sigma \;=\; (S, T, W, \lambda, M)$$

such that: S and T are disjoint sets of respectively *places* and *transitions*; W is a *weight function* from the set $(S \times T) \cup (T \times S)$ to the set of natural numbers $\mathbf{N}$; λ is a *labelling* function for places and transitions such that $\lambda(s) \in \{\mathsf{e}, \mathsf{i}, \mathsf{x}\}$, for every place $s \in S$, and $\lambda(t)$ is a relabelling ϱ of the form (1), for every transition $t \in T$; and M is a *marking*, i.e., a mapping assigning a natural number to each place $s \in S$.[6]

We adopt the standard rules about representing nets as directed graphs, viz. places are represented as circles, transitions as rectangles, the flow relation generated by W is indicated by arcs annotated with the corresponding weights, and markings are shown by placing tokens within circles. As usual, the zero weight arcs will be omitted and the unit weight arcs (or unitary arcs) will be left unannotated. To avoid ambiguity, we will sometimes decorate the various components of Σ with the index Σ; thus, T_Σ denotes the set of transitions of Σ, etc. A net is *finite* if both S and T are finite sets. Figure 6 shows the graph of a labelled net $\Sigma_0 = (S_0, T_0, W_0, \lambda_0, M_0)$ defined thus:

$$\begin{aligned}
S_0 &= \{s_0, s_1, s_2, s_3\} \\
T_0 &= \{t_0, t_1, t_2\} \\
W_0 &= ((TS \cup ST) \times \{1\}) \;\cup\; (((S \times T) \setminus ST \;\cup\; (T \times S) \setminus TS) \times \{0\}) \\
\lambda_0 &= \{(s_0, \mathsf{e}), (s_1, \mathsf{i}), (s_2, \mathsf{x}), (s_3, \mathsf{e}), (t_0, a), (t_1, b), (t_2, a)\} \\
M_0 &= \{(s_0, 1), (s_1, 0), (s_2, 0), (s_3, 1)\}\ .
\end{aligned}$$

where $TS = \{(t_0, s_1), (t_1, s_2), (t_2, s_3)\}$ and $ST = \{(s_0, t_0), (s_1, t_1), (s_3, t_2)\}$.

If the labelling of a place s in a labelled net Σ is e then s is an *entry* place, if i then s is an *internal* place, and if x then s is an *exit* place. By convention, ${}^\circ\Sigma$, Σ° and $\ddot{\Sigma}$

[6] Sometimes we will treat M as a multiset, or even a set, if $M(S) \subseteq \{0, 1\}$.

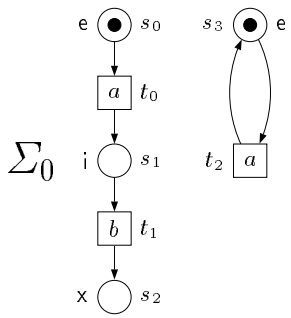


Fig. 6. A labelled net.

denote respectively the entry, exit and internal places of Σ. For every place (transition) x, we use ${}^\bullet x$ to denote is pre-set, i.e., the set of all transitions (places) y such that there is an arc from y to x, $W(y,x) > 0$. The post-set $x^\bullet$ is defined in a similar way. The pre- and post-set notation extends in the usual way to sets R of places and transitions, e.g., ${}^\bullet R = \bigcup\{{}^\bullet r \mid r \in R\}$. In what follows, all nets are assumed to be T-*restricted*, i.e., the pre- and post-sets of each transition are non-empty. No assumption of that kind is made for places. For the labelled net of figure 6 we have ${}^\circ\Sigma_0 = \{s_0, s_3\}$, $\Sigma_0^\circ = \{s_2\}$, ${}^\bullet s_0 = \emptyset$, $s_0^\bullet = \{t_0\}$ and $\{s_0, s_1\}^\bullet = \{t_0, t_1\} = {}^\bullet\{s_1, s_2\}$.

A labelled net Σ is *ex-restricted* if there is at least one entry and at least one exit place, ${}^\circ\Sigma \neq \emptyset \neq \Sigma^\circ$. Σ is *e-directed* (*x-directed*) if the entry (resp. exit) places are free from incoming (resp. outgoing) arcs, i.e., ${}^\bullet({}^\circ\Sigma) = \emptyset$ (resp. $(\Sigma^\circ)^\bullet = \emptyset$). Moreover, Σ is *ex-directed* if it is both e-directed and x-directed. The labelled net Σ_0 in figure 6 is ex-restricted and x-directed, but it is not e-directed.

A labelled net Σ is *simple* if W always returns 0 or 1, and *pure* if for all transitions $t \in T$, ${}^\bullet t \cap t^\bullet = \emptyset$. If Σ is not pure then there are $s \in S$ and $t \in T$ such that $W(s,t) \cdot W(t,s) > 0$; in such a case, the pair $\{s,t\}$ will be called a *side-loop*. The net in figure 6 is finite and simple, but is not pure as it contains a side-loop, $\{s_3, t_2\}$.

We will use three explicit ways of modifying the marking of $\Sigma = (S, T, W, \lambda, M_\Sigma)$. We define $\lfloor\Sigma\rfloor$ as $(S, T, W, \lambda, \emptyset)$; typically, this operation is used when $M_\Sigma \neq \emptyset$, since it erases all tokens. Moreover, we define $\overline{\Sigma}$ and $\underline{\Sigma}$ as, respectively, $(S, T, W, \lambda, {}^\circ\Sigma)$ and $(S, T, W, \lambda, \Sigma^\circ)$. We will call ${}^\circ\Sigma$ the *entry marking*, and Σ° the *exit marking* of Σ. Note also that $\lfloor . \rfloor$, $\overline{(.)}$ and $\underline{(.)}$ are syntactic operations having nothing to do with derivability (reachability) in the sense of the step sequence semantics defined next.

Step sequence semantics We adopt finite step sequence semantics for a labelled net $\Sigma = (S, T, W, \lambda, M)$, in order to capture the potential concurrency in the behaviour of the system modelled by Σ. A finite multiset of transitions U, called a *step*, is *enabled* by Σ if for every place $s \in S$,

$$M(s) \;\geq\; \sum_{t \in U} W(s,t) \cdot U(t) \;.$$

We denote this by $\Sigma[U\rangle$, or $M[U\rangle$ if the net is understood from the context. An enabled step U can be *executed*, leading to a follower marking M' defined, for every place $s \in S$, by

$$M'(s) \;=\; M(s) - \sum_{t \in U} W(s,t) \cdot U(t) + \sum_{t \in U} W(t,s) \cdot U(t) \;.$$

We will denote this by $M\ [U\rangle\ M'$ or $\Sigma\ [U\rangle\ \Theta$, where Θ is the labelled net (S, T, W, λ, M'). Transition labelling may be extended to steps, through the formula

$$\lambda(U) \;=\; \sum_{t \in U} U(t) \cdot \{\lambda(t)\} \;\in\; \mathsf{mult}(\mathsf{Lab}) \;.$$

Although we will use the same term 'step' to refer both to a finite set of transitions and to a finite multiset of labels, it will always be clear from the context which one is meant. The notation for action based steps will be $\Sigma\ [\Gamma\rangle_{\mathsf{lab}}\ \Theta$, where $\Gamma = \lambda(U)$.

A *finite step sequence* of Σ is a finite (possibly empty) sequence $\sigma = U_1 \ldots U_k$ of steps for which there are labelled nets $\Sigma_0, \ldots, \Sigma_k$ such that $\Sigma = \Sigma_0$ and for every $1 \leq i \leq k$, $\Sigma_{i-1}\ [U_i\rangle\ \Sigma_i$. Depending on the context, this shall be denoted by one of the following notations:

$$\Sigma\ [\sigma\rangle\ \Sigma_k \quad \text{or} \quad M_\Sigma\ [\sigma\rangle\ M_{\Sigma_k} \quad \text{or} \quad \Sigma_k \in [\Sigma\rangle \quad \text{or} \quad M_{\Sigma_k} \in [M_\Sigma\rangle \;.$$

Moreover, the marking M_{Σ_k} will be called *reachable* from M_Σ, and Σ_k *derivable* from Σ. The empty step will always be enabled, but it can be ignored when one considers a step sequence, since the empty step always relates a net to itself, i.e.,

$$\Sigma\ [\emptyset\rangle\ \Theta \iff \Sigma = \Theta \iff \Sigma\ [\varepsilon\rangle\ \Theta \;,$$

where ε denotes the empty sequence. A net exhibits *auto-concurrency* if some reachable marking enables a step which is not a set.

Consider the labelled net Σ_0 in figure 6. In the initial marking, transitions t_0 and t_2 are enabled concurrently and so $\{t_0, t_2\}$ is an enabled step. After this step has been executed, transitions t_1 and t_2 are enabled concurrently and, hence, $\{t_0, t_2\}\{t_1, t_2\}$ is a step sequence of the net from the marking shown in figure 6. A step does not need to be maximal. Thus, for instance, $\{t_0\}$ is also a step of Σ_0, and $\{t_0\}\{t_1\}\{t_2\}\{t_2\}$ and $\{t_0, t_2\}\{t_1\}\{t_2\}$ are step sequences. In terms of labelled steps, the step sequence $\{t_0, t_2\}\{t_1\}\{t_2\}$ corresponds to $\{a, a\}\{b\}\{a\}$. Notice that different step sequences may correspond to the same labelled step sequence. For example, both $\{t_0\}\{t_2\}\{t_1\}$ and $\{t_2\}\{t_0\}\{t_1\}$ correspond to $\{a\}\{a\}\{b\}$.

Safeness, cleanness and exclusiveness The marking M of Σ is *safe* if for all $s \in S$, $M(s) \in \{0, 1\}$. As already indicated, a safe marking can and will often be identified with the set of places to which it assigns 1. A marking is *clean* if it is not a proper super-multiset of ${}^\circ\Sigma$ or Σ°, i.e., if ${}^\circ\Sigma \subseteq M$ or $\Sigma^\circ \subseteq M$ implies ${}^\circ\Sigma = M$ or $\Sigma^\circ = M$, respectively. Finally, a marking is *ex-exclusive* if it does not simultaneously mark an entry place and an exit place, i.e., if $M \cap {}^\circ\Sigma = \emptyset$ or $M \cap \Sigma^\circ = \emptyset$.

The marking of the net in figure 6 is safe, clean and ex-exclusive. It would cease to be clean if we added a token to it, even if this new token was put on one of the entry places. A labbeled net is called *safe* (*clean*) if all its reachable markings are safe (resp. clean), *k-bounded* if no reachable marking puts more than k tokens on any place (so that 1-boundedness is the same as safeness), and *bounded* if there is some k such that it is k-bounded.

Complex markings In the operation of net refinement, transitions of one net will be replaced by other, possibly very complex, nets. Thus it is possible to consider that the execution of these transitions may take some (long) time or, indeed, may last indefinitely (if the nets replacing them deadlock or operate endlessly). This may be captured by a special kind of extended markings (the idea here resembles that used in dealing with behavioural equivalences in the context of action refinement [22, 29, 60]).

A *complex marking* of a labelled net Σ is a pair $\mathcal{M} = (M, Q)$ comprising a standard marking M of Σ and a finite multiset Q of *engaged* transitions of Σ. M may be considered as the *real* part, and Q as the *imaginary* part of $\mathcal{M}$. A standard marking M may then be identified with the complex marking $(M, \emptyset)$. The enabling and execution rules are extended thus. Let U, V and $W \subseteq Q$ be finite multisets of transitions. Then we will denote $\mathcal{M}\ [U + V^+ + W^-\rangle\ (M', Q')$ if, for every $s \in S$,

$$M(s) \geq \sum_{t \in U+V} W(s,t) \cdot (U(t) + V(t))$$

$$M'(s) = M(s) - \sum_{t \in U+V} W(s,t) \cdot (U(t) + V(t)) + \sum_{t \in U+W} W(t,s) \cdot (U(t) + W(t))$$

and, furthermore, $Q' = Q + V - W$. The notions of safeness, n-boundedness, cleanness, etc, extend to the case of complex markings. For instance, $\mathcal{M}$ is safe if $M(s) = 0$, for all $s \in {}^\bullet Q \cup Q^\bullet$.

3.2 Partial order semantics based on occurrence nets

The step semantics is not the only way to capture concurrency in Petri net behaviours. Another concurrency semantics of nets is based on the notion of a *process*.[7]

A process is an 'unfolding' of a net from the initial marking, modelling one of its evolutions (or a family of evolutions equivalent with respect to causality and concurrency) through the absorptions and productions of tokens. This is illustrated in figure 7, where the process π corresponds, among others, to the step sequences $\{v_1\}\{v_2, v_4\}\{v_2\}\{v_3\}$ and $\{v_1\}\{v_2\}\{v_2, v_4\}\{v_3\}$. In general, a processes is a labelled (occurrence) net satisfying some additional criteria.

Occurrence nets An *occurrence net* is a quadruple $\pi = (B, E, F, l)$ where B is the set of conditions (places), E is the set of events (transitions), F is the flow function (a weight function for arcs), and l is a labelling for $B \cup E$. The arc weights are binary, $F((B \times E) \cup (E \times B)) \subseteq \{0, 1\}$; for every condition b in B, $|{}^\bullet b| \leq 1 \geq |b^\bullet|$; and when considered as a directed graph, π is acyclic. The latter means that it induces a partial order $\mathsf{poset}(\pi)$ on the set of nodes:

$$\mathsf{poset}(\pi) \ = \ (B \cup E \ , \ \preceq_\pi)$$

[7] The word *process* is used here as a term from the Petri net theory; it has nothing to do with process algebras and process expressions. A general presentation of the process semantics and its relationship with the interleaving and step sequence semantics may be found in [6], where a special care has been devoted to infinite nets.

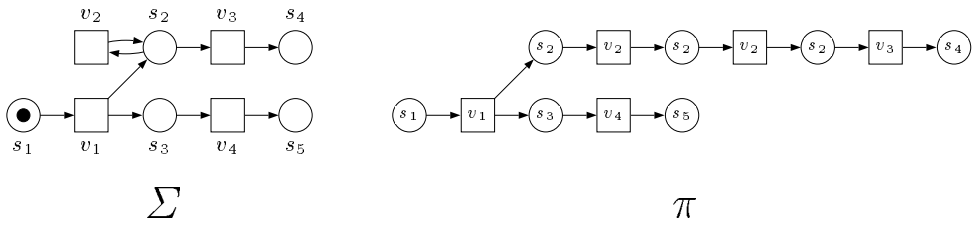


Fig. 7. A net and its processes.

where $\preceq_\pi$ is the transitive reflexive closure of $F^{-1}(1)$. The set of the minimal and maximal elements in $\mathsf{poset}(\pi)$ will be denoted by $\mathsf{Min}(\pi)$ and $\mathsf{Max}(\pi)$, respectively. Two nodes of $\mathsf{poset}(\pi)$ are *concurrent* if they are not ordered by $\preceq_\pi$. A *B-cut* is a maximal subset of concurrent conditions. The *predecessors* of a node are all the nodes preceding it in $\mathsf{poset}(\pi)$.

Processes For a finite occurrence net to be a valid process of a finite labelled net, some additional conditions must be fulfilled by the former in order to make it a faithful representation of an evolution of the latter. A finite occurrence net $\pi = (B, E, F, l)$ is a *process* of a finite labelled net $\Sigma = (S, T, W, \lambda, M_0)$ if the following hold.

- The conditions and events of π are labelled respectively by the places and transitions of Σ, $l(B) \subseteq S$ and $l(E) \subseteq T$. (sort preservation - SP)
- For every $e \in E$ and $s \in S$, $W(s, l(e)) = |l^{-1}(s) \cap {}^\bullet e|$ and $W(l(e), s) = |l^{-1}(s) \cap e^\bullet|$. (local coherence - LC)
- For every $s \in S$, $M_0(s) = |l^{-1}(s) \cap \mathsf{Min}(\pi)|$. (initial coherence - IC)

The conditions of π correspond to tokens occurring at some time in the places of Σ during evolution(s) corresponding to π, and the events of π correspond to transition occurrences. (LC) means that the correct number of tokens is absorbed and produced by each transition occurrence, and (IC) reflects the assumption that the system starts in the state given by the initial marking. The following properties establish a formal link between the processes of a labelled net and its behaviour:

- Every linearisation δ of a partition of the events of a process, compatible with the associated partial order,[8] leads through its labels to a valid finite step sequence $\sigma(\delta)$ occurring from the initial marking.[9] Conversely, for every valid finite step sequence σ occurring from the initial marking, there is a process from which this sequence can be obtained in the way we have just described. (linearisation - Lin)
- If M is the marking reached by the step sequence σ in (Lin), then, for every $s \in S$, $M(s) = |l^{-1}(s) \cap \mathsf{Max}(\pi)|$. (final coherence - FC)
- For every B-cut μ of the process, the restriction of the process to μ and the predecessors of its conditions is a valid process; in particular, there is a reachable marking M_μ

[8] That is, $\delta = E_1 E_2 \ldots E_k$ where $E = E_1 \cup E_2 \cup \ldots \cup E_k$ and, for all distinct $e \in E_i$ and $f \in E_j$, $e \preceq_\pi f$ implies $i < j$.

[9] That is, $\sigma(\delta) = l(E_1) l(E_2) \ldots l(E_k)$ and $M_0 \, [\sigma(\delta)\rangle$.

corresponding to μ such that $M_\mu(s) = |l^{-1}(s) \cap \mu|$, for every $s \in S$. Moreover, the part of the process comprising μ and all the nodes after μ is also a process, but for the net started from the marking M_μ. (cuts consistency - Cut)

- If some conditions are mutually concurrent, then there is a reachable marking comprising their labels (and respecting their multiplicity), and if some events are mutually concurrent, then there is a reachable marking allowing the step corresponding to their labels (with their label multiplicity). (concurrency - Con)

Infinite processes of finite nets To model infinite evolutions π of a finite net Σ, another condition has to be added to the definition of a process π:

- Every node has finitely many predecessors. (first finite accessibility - FFA)

This avoids a counterintuitive situation where tokens 'come' from an infinite past. For example, (SP), (LC) and (IC) are all fulfilled by the occurrence net π in figure 8, but (FFA) does not hold. And, indeed, intuitively, π does not correspond to any valid evolution of Σ.

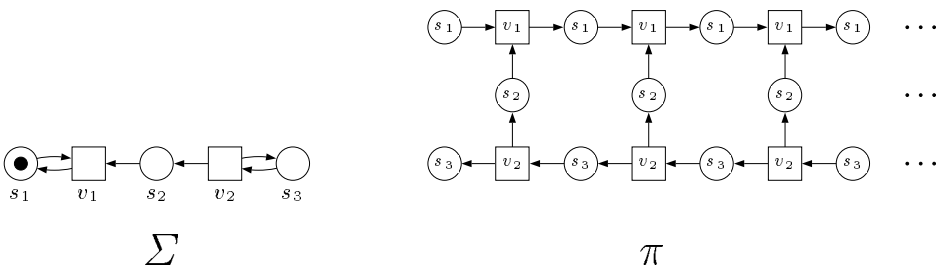


Fig. 8. An occurrence net which satisfies all process axioms except (FFA).

The link between the extended processes and (possibly infinite) net evolutions is similar as before: (Lin) holds after assuming that δ can be infinite and all the sets appearing in it are finite, moreover, σ may be infinite; (Cut) holds but M_μ exists only if μ is finite; (Con) holds without any changes for finite sets of conditions and events; and (FC) still holds for finite σ and π.

Processes of infinite nets with finite steps To deal with infinite labelled nets, (FFA) has to be modified. There are at least two ways of proceeding, depending on whether we intend to allow infinite steps or not. If not, we may require that:

- Every node has finitely many predecessor events. (second finite accessibility - SFA)

We ignore predecessor conditions since, in an infinite net, a single transition may insert/absorb tokens into/from infinitely many places. The properties of the new class of processes are similar to those stated above.

Processes of infinite nets with infinite steps If we want to allow infinite steps, the finite accessibility property can be formulated thus:

- For every node there is an upper bound on the length of the chains of nodes leading to it. (third finite accessibility - TFA)

The link between processes satisfying (TFA) and the possibly infinite evolutions of the net is again similar as before with one exception, namely, in (Lin) δ can contain infinite sets, and σ infinite steps.

(TFA) implies that every node is accessible from $\mathsf{Min}(\pi)$ and that every event may be accessed through a finite sequence of possibly infinite steps. In general, (TFA) is more liberal than (SFA).

3.3 Partial order semantics based on Mazurkiewicz traces

Since occurrence nets support the notion of a place while process expressions do not, we will also use Mazurkiewicz traces [48]. Such traces are a model of partial order behaviour based solely on transitions (and not on places).

Let A be a set and $\mathsf{ind} \subseteq A \times A$ be an irreflexive symmetric relation on A. The idea here is that A represents the set of all possible events in a concurrent system, and ind is an *independence* relation which asserts which events can be executed concurrently. With every sequence $\sigma = A_1 \ldots A_k$, where each A_i is a finite subset of A such that $(a,b) \in \mathsf{ind}$ for all distinct $a, b \in A_i$, we associate a partial order (poset), denoted by $\mathsf{poset}_{\mathsf{ind}}(\sigma)$, in the following way.

The set of *event occurrences* of σ, occ_σ, comprises all pairs $(a,l) \in A \times \mathbf{N}$ such that $a \in A_1 \cup \ldots \cup A_k$ and l is less or equal to the number of times a occurs within σ. Moreover, we denote by $\mathsf{idx}_{(a,l)}$ the index m such that A_m contains the l-th occurrence of a in σ. We then define a *precedence* relation on occ_σ, $\prec_\sigma$, by stipulating that $(a,l) \prec_\sigma (b,n)$ whenever $(a,b) \notin \mathsf{ind}$ and $\mathsf{idx}_{(a,l)} < \mathsf{idx}_{(b,n)}$. Then

$$\mathsf{poset}_{\mathsf{ind}}(\sigma) \;=\; (\mathsf{occ}_\sigma, \prec_\sigma^*)$$

where $\prec_\sigma^*$ is the transitive reflexive closure of $\prec_\sigma$.

For a labelled net Σ, let ind_Σ be a symmetric relation on its transitions, defined by:

$$\mathsf{ind}_\Sigma \;=\; \{(t,u) \in T_\Sigma \times T_\Sigma \mid ({}^\bullet t \cup t^\bullet) \cap ({}^\bullet u \cup u^\bullet) = \emptyset\} \;.$$

This relation is called the *independence relation*, because two distinct transitions belonging to ind_Σ have no impact on their respective environments. If they are both enabled individually, then they are also enabled simultaneously.

Then, with every finite step sequence σ of a safe labelled net Σ, $\Sigma\ [\sigma\rangle\ \Theta$, we can associate a partial order, $\mathsf{poset}_{\mathsf{ind}_\Sigma}(\sigma)$, in the way described above, and after taking A to be the transition set, $A = T_\Sigma$. For example, the step sequence

$$\sigma = \{t_0, t_2\}\emptyset\{t_1, t_2\}\{t_2\}\emptyset$$

of the labelled net Σ_0 in figure 6 generates a partial order $\mathsf{poset}_{\mathsf{ind}_{\Sigma_0}}(\sigma)$ whose Hasse diagram is shown on the left-hand side of figure 9. This partial order is often represented as a labelled

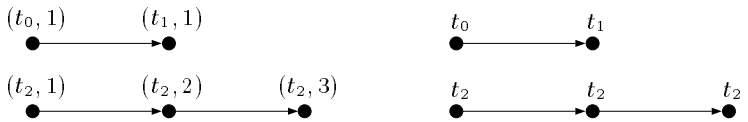


Fig. 9. Hasse diagrams of partial orders.

partial order with nodes being labelled by transitions, as shown on the right-hand side of figure 9.
The presence of a path between two nodes is interpreted as *causality*, and the lack of ordering as *concurrency*. Thus, in the example above, the occurrences of t_0 and t_1 are causally related, and both are concurrent with the three occurrences of t_2. Whenever $\Sigma\ [\sigma\rangle\ \Theta$, we will write $\Sigma\ [\mathsf{poset}_{\mathsf{ind}_\Sigma}(\sigma)\rangle_{\mathsf{po}}\ \Theta$ to indicate that Θ arises from Σ through the execution of the poset between the brackets $[\ldots\rangle$.

3.4 Equivalence notions

One may consider various behavioural equivalences for labelled nets. It may first be observed that the whole set of step sequences of a labelled net may be specified by defining its *full reachability graph*, whose nodes are all reachable markings (or equivalently, all derivable nets) and whose arcs are labelled with steps which transform one marking into another. For example, figure 10 represents the full reachability graph of the net Σ_0 shown in figure 6; the empty steps are left implicit, and the encircled white dot indicates the initial node (here and elsewhere). The arc labels may be transition steps (as in figure 10) or labelled steps. Any (finite) path in the above graph starting at the initial node specifies a legal (finite) step sequence of the net Σ_0, and vice versa.

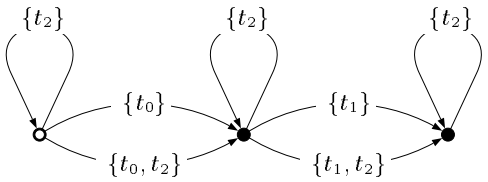


Fig. 10. Full reachability graph of Σ_0.

Transition systems Using the full reachability graph to represent the behaviour of a labelled net leads to problems in a compositional setting. In particular, isomorphism of reachability graphs is not preserved by, e.g., choice composition of nets, as demonstrated informally in figure 11. We address this problem by augmenting the behaviour of a labelled net Σ with two auxiliary moves, skip and redo, which can transform its initial state into

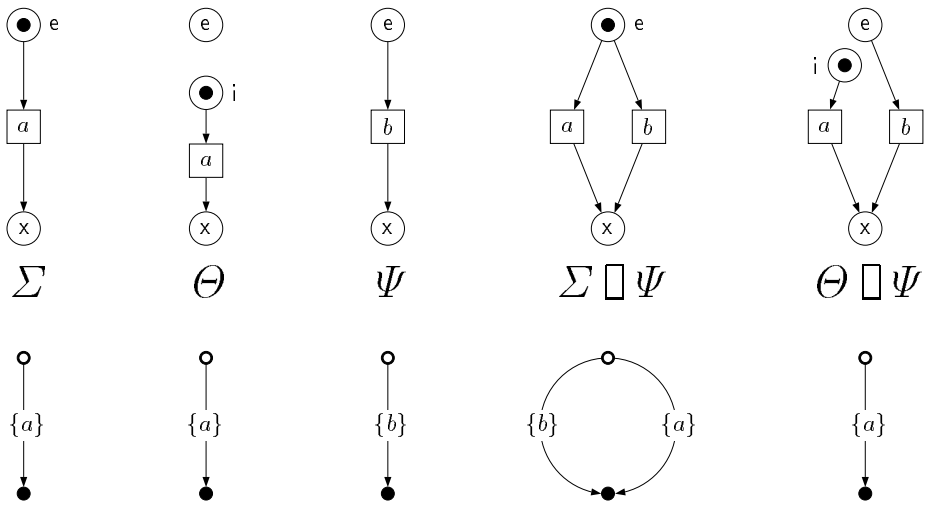


Fig. 11. Five nets and the corresponding (labelled) full reachability graphs demonstrating that isomorphism of reachability graphs is not preserved by choice composition.

the terminal state, and vice versa. We achieve the desired effect by artificially adding to Σ two fresh transitions, skip and redo, so that ${}^\bullet\mathsf{skip} = \mathsf{redo}^\bullet = {}^\circ\Sigma$, $\mathsf{skip}^\bullet = {}^\bullet\mathsf{redo} = \Sigma^\circ$, $\lambda(\mathsf{skip}) = \mathsf{skip}$ and $\lambda(\mathsf{redo}) = \mathsf{redo}$. Moreover, we assume that all the arcs adjacent to the skip and redo transitions are unitary, $\mathsf{redo}, \mathsf{skip} \notin \mathsf{Lab}$, and ${}^\circ\Sigma \neq \emptyset \neq \Sigma^\circ$ (i.e., the net is ex-restricted). The latter condition is needed to ensure that the two new transitions are T-restricted. The net Σ augmented with skip and redo will be denoted by Σ_{sr}.

The *transition system* of a marked net[10] Σ is defined as $\mathsf{ts}_\Sigma = (V, L, A, v_0)$ where: $V = \{\Theta \mid \mathsf{skip}, \mathsf{redo} \notin T_\Theta \ \wedge \ \Theta_{\mathsf{sr}} \in [\Sigma_{\mathsf{sr}}\rangle \ \}$ is the set of states; $v_0 = \Sigma$ is the initial state; $L = \mathsf{mult}(\mathsf{Lab} \cup \{\mathsf{redo}, \mathsf{skip}\})$ is the set of arc labels; and

$$A \ = \ \{(\Theta, \Gamma, \Psi) \in V \times L \times V \mid \Theta_{\mathsf{sr}} \ [\Gamma\rangle_{\mathsf{lab}} \ \Psi_{\mathsf{sr}}\}$$

is the set of arcs. In other words, ts_Σ is the labelled reachability graph of Σ_{sr} with all references to skip and redo in the nodes of the graph erased. The transition system of an unmarked net Σ is defined as $\mathsf{ts}_\Sigma = \mathsf{ts}_{\overline{\Sigma}}$.

The *full transition system* of a marked net Σ is defined as $\mathsf{fts}_\Sigma = (V, L', A', v_0)$ where: V is the set of states and v_0 is the initial state, both defined as above; L' is the set of all finite sets of transitions of Σ_{sr}; and

$$A' \ = \ \{(\Theta, U, \Psi) \in V \times L' \times V \mid \Theta_{\mathsf{sr}} \ [U\rangle \ \Psi_{\mathsf{sr}}\}$$

is the set of arcs. In other words, fts_Σ is the reachability graph of Σ_{sr} with all references to skip and redo in the nodes of the graph erased. For an unmarked net Σ, $\mathsf{fts}_\Sigma = \mathsf{fts}_{\overline{\Sigma}}$.

[10] Where Σ is *marked* if $M_\Sigma \neq \emptyset$, and *unmarked* otherwise.

Figure 12 shows how augmenting labelled nets with the redo and skip transitions allows one to discriminate between the nets Σ and Θ depicted in figure 11. In general, skip and redo allow for distinguishing the entry and exit states from the other ones, and for modelling the fact that if a net is left (through the exit state), it may later be possible to re-enter it (through the entry state).

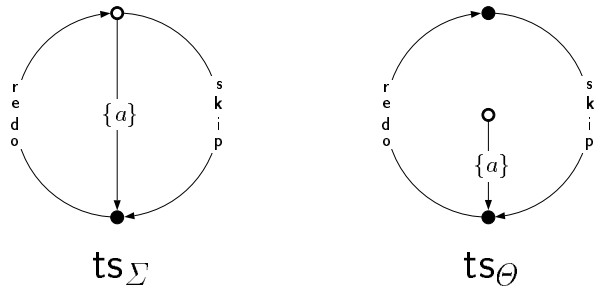


Fig. 12. Discriminating transition systems for the nets in figure 11.

Behavioural equivalences Let Σ and Θ be two labelled nets which are either both unmarked, or both marked. The nets are *ts-isomorphic*, $\Sigma \cong \Theta$, if ts_Σ and ts_Θ are isomorphic transition systems, and *strongly equivalent*, $\Sigma \approx \Theta$, if ts_Σ and ts_Θ are strongly equivalent transition systems, where two transition systems, (V, L, A, v_0) and (V', L', A', v_0'), are *strongly equivalent* if there is a relation $R \subseteq V \times V'$, itself called a *strong bisimulation*, such that $(v_0, v_0') \in R$, and if $(v, v') \in R$ then

$$\begin{aligned} (v,l,w) \in A \quad &\Longrightarrow \quad \exists w' \in V' : (v',l,w') \in A' \wedge (w,w') \in R \\ (v',l,w') \in A' \quad &\Longrightarrow \quad \exists w \in V \quad : (v,l,w) \in A \quad \wedge (w,w') \in R. \end{aligned}$$

Figure 13 shows two strongly equivalent labelled nets which are, however, not ts-isomorphic.

Structural equivalences What we have done is not the only way to define meaningful equivalence notions in the domain of labelled nets. In particular, since ts-isomorphism and strong equivalence are behavioural notions, it may be difficult to check if two nets are indeed equivalent. Now, since we are working with nets, it is also possible to define equivalences based on the structure of nets rather than on their behaviour.

Arguably the strongest structural equivalence, other than equality, is net isomorphism. Two labelled nets, Σ and Θ, are *isomorphic* if there is a bijective mapping $\psi: S_\Sigma \cup T_\Sigma \to S_\Theta \cup T_\Theta$ such that for all $s \in S_\Sigma$ and $t \in T_\Sigma$, $\psi(s) \in S_\Theta$, $\psi(t) \in T_\Theta$, $\lambda_\Theta(\psi(s)) = \lambda_\Sigma(s)$, $\lambda_\Theta(\psi(t)) = \lambda_\Sigma(t)$, $M_\Theta(\psi(s)) = M_\Sigma(s)$,

$$W_\Theta(\psi(s), \psi(t)) = W_\Sigma(s,t) \quad \text{and} \quad W_\Theta(\psi(t), \psi(s)) = W_\Sigma(t,s)\ .$$

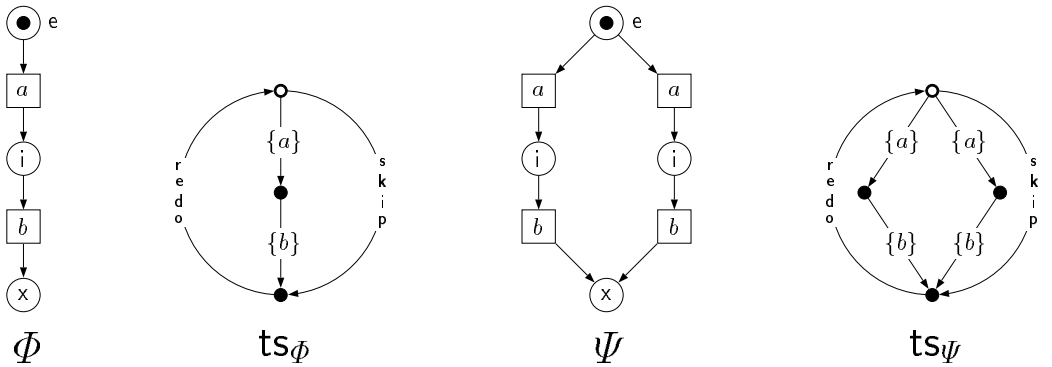


Fig. 13. Two strongly equivalent, but not ts-isomorphic, labelled nets.

We will denote this by Σ iso Θ, and call ψ an *isomorphism* for Σ and Θ.

Weaker equivalences are obtained by allowing the two nets differ only by duplicating places and transitions. Two places s and s' *duplicate* each other in a labelled net Σ if $\lambda_\Sigma(s) = \lambda_\Sigma(s')$, $M_\Sigma(s) = M_\Sigma(s')$, and for every transition t, $W_\Sigma(s,t) = W_\Sigma(s',t)$ and $W_\Sigma(t,s) = W_\Sigma(t,s')$; then, in any evolution of the net, the two places do not add anything with respect to each other. Similarly, two transitions t and t' *duplicate* each other if $\lambda_\Sigma(t) = \lambda_\Sigma(t')$, and for every place s, $W_\Sigma(s,t) = W_\Sigma(s,t')$ and $W_\Sigma(t,s) = W_\Sigma(t',s)$; then, in any label based evolution of the net, the two transitions do not add anything with respect to each other. Clearly, the relation of *duplication* is an equivalence relation for places and transitions; thus it is possible to replace places and/or transitions by their duplication equivalence classes obtaining nets with essentially the same behaviour.

Two labelled nets, Σ and Θ, will be called *place-duplicating / transition-duplicating / node-duplicating* if they lead to isomorphic nets when their places / transitions / nodes, are replaced by their duplication equivalence classes which inherit the labels, connectivity and markings of their elements. We will denote this by Σ $\mathsf{iso_S}$ Θ / Σ $\mathsf{iso_T}$ Θ / Σ $\mathsf{iso_{ST}}$ Θ, respectively. It is a straightforward observation that in the domain of ex-restricted labelled nets,

$$\mathsf{iso} \subset \mathsf{iso_S} \subset \mathsf{iso_{ST}} \quad \text{and} \quad \mathsf{iso} \subset \mathsf{iso_T} \subset \mathsf{iso_{ST}} \quad \text{and} \quad \mathsf{iso_{ST}} \subset \cong \subset \approx \ .$$

For the nets in figure 14, we have Σ_1 $\mathsf{iso_T}$ Σ_2 $\mathsf{iso_{ST}}$ Σ_3 $\mathsf{iso_S}$ Σ_1. Moreover, Σ_4 is not equivalent to any of the other three nets since, intuitively, Σ_4 has no reachable terminal state (the x-places cannot be marked concurrently).

3.5 Plain boxes

A *box* is an ex-restricted labelled net Σ. A box is *plain* if its transitions are all labelled by constant relabellings.[11] There are two main classes of plain boxes.

[11] Or labels, according to our convention of identifying constant relabellings with labels.

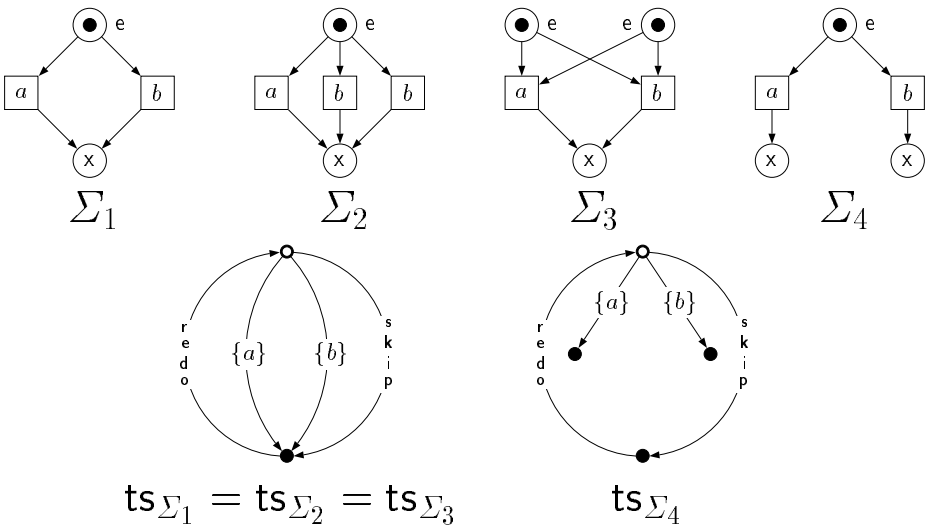


Fig. 14. Four labelled nets and their transition systems.

An unmarked plain box Σ will be called *static* if each marking reachable from ${}^\circ\Sigma$ or Σ° is safe and clean. Static boxes are net counterparts of static expressions, i.e., expressions without overbars and underbars. Later, we will see that unmarked boxes that are constructed by applying net operators described in sections 6 and 7 do satisfy these properties.

A marked plain box Σ is *dynamic* if each marking reachable from M_Σ or ${}^\circ\Sigma$ or Σ° is safe and clean. Dynamic boxes are net counterparts of dynamic expressions, i.e., expressions with active subexpressions indicated by overbars and underbars. Note that if Σ is a static box and Θ is derivable from $\overline{\Sigma}$ then Θ is a dynamic box.

When a box Σ or Σ_{sr} is marked, its reachable markings are always non-empty (due to T-restrictedness and ex-restrictedness). On the other hand, the empty marking of a box has no successor markings except itself. Thus, the distinction between static and dynamic boxes is invariant over behaviour. Moreover, if Σ is a static box then both $\overline{\Sigma}$ and $\underline{\Sigma}$ are dynamic boxes, and if Σ is a dynamic box then $\lfloor\Sigma\rfloor$ is a static box.

Proposition 1. *If Σ is a dynamic box then every net derivable from it is a dynamic box. Moreover, if U is a step enabled by the marking of Σ then $U \times U \subseteq \mathsf{ind}_\Sigma \cup id_{T_\Sigma}$ and $W_\Sigma(U \times S_\Sigma) \cup W_\Sigma(S_\Sigma \times U) \subseteq \{0,1\}$.* □

It could be objected that determining whether all the reachable markings are safe and clean may be hard, especially for infinite boxes. However, we shall see in section 6 that a simple structural argument may be used to prove that the boxes we shall construct have the required property, and in section 7 that a careful behavioural analysis leads to the same conclusion.

We further distinguish two special classes of boxes, called the *entry* and *exit* boxes, which comprise all dynamic boxes Σ such that M_Σ is, respectively, ${}^\circ\Sigma$ and Σ°. The sets of plain

static, dynamic, entry and exit boxes will, respectively, be denoted by Box^s, Box^d, Box^e, and Box^x. The entire set of plain boxes will be denoted by Box.

It is the static boxes Σ (or, equivalently, the entry boxes $\overline{\Sigma}$) which in our approach ultimately provide a denotational semantics of the static PBC expressions. Two behavioural conditions were imposed on the markings M reachable from the entry or exit marking of Σ. First, we require M to be safe in order to ensure that the semantics of the boxes is as simple as possible (in particular, that the boxes we consider do not exhibit auto-concurrency), and so we can directly use a partial order semantics of Petri nets in the style of Mazurkiewicz (cf. section 3.3). The second condition, that M is always a clean marking, is a consequence of the first condition and our intension to use iterative constructs in the algebra of nets (if $\overline{\Sigma}$ allows one to reach a non-clean marking then $\overline{\Sigma}_{\mathsf{sr}}$ is not safe). Although static boxes are of our primary interest, we also need to be able to represent intermediate markings. For this reason we introduced dynamic boxes which, by definition, include all marked nets Θ such that Θ is derivable from $\overline{\Sigma}$, for some static box Σ. Dynamic boxes will provide a denotational semantics for the dynamic PBC expressions, and hence will allow us to show, in section 7, that operational and Petri net semantics of PBC expressions are equivalent.

A box Σ will be called *ex-exclusive* if each marking reachable from M_Σ or ${}^\circ\Sigma$ or Σ° is ex-exclusive. Moreover, for every pair of non-empty disjoint sets of places S_e and S_x, the *ex-box*indexex-box $\mathsf{ex}(S_e, S_x)$ is a special simple box Σ such that ${}^\circ\Sigma = S_e$, $\Sigma^\circ = S_x$ and $\ddot{\Sigma} = T_\Sigma = \emptyset$.

4 Net refinement

The basis for providing plain boxes with an algebraic structure is a general *simultaneous refinement and relabelling meta-operator* (*net refinement*, for short), which embodies a mechanism combining transition refinement and interface change defined by relabellings. It is defined for a simple box Ω – an operator box – which serves as a pattern for gluing together a set of plain boxes $\mathbf{\Sigma}$ (one plain box for every transition in Ω) along their entry and exit interfaces. The relabellings annotating the transitions of Ω specify the interface changes for the boxes in $\mathbf{\Sigma}$.

4.1 Operator boxes

An *operator box* is a simple box Ω all of whose relabellings are transformational, and whose marking is a complex one, $\mathcal{M}_\Omega = (M, Q)$. Let $\mathbf{\Sigma} : T_\Omega \to \mathsf{Box}$ be a function from the transitions of Ω to plain boxes with real markings. We will refer to $\mathbf{\Sigma}$ as an *Ω-tuple* and denote $\mathbf{\Sigma}(v)$ by Σ_v, for every $v \in T_\Omega$. If the set of transitions of Ω is finite, we will assume that $T_\Omega = \{v_1, \ldots, v_n\}$ is implicitly ordered, and denote $\mathbf{\Sigma} = (\Sigma_{v_1}, \ldots, \Sigma_{v_n})$. For $T \subseteq T_\Omega$, we will use $\mathbf{\Sigma}_T$ to denote the restriction of $\mathbf{\Sigma}$ to the transitions in T.

The composition operation will be applicable to every Ω-tuple $\mathbf{\Sigma}$, provided $M_{\Sigma_v} \neq \emptyset$ if and only if $v \in Q$. This implies, in particular, that only finitely many Σ_v's are marked. However, we will sometimes need extra generality and allow infinite Q, with the restriction that, for every $s \in S_\Omega$,

$$|\{v \in {}^\bullet s \mid 0 < \sum_{x_v \in \Sigma_v^\circ} M_{\Sigma_v}(x_v)\}| + |\{v \in s^\bullet \mid 0 < \sum_{e_v \in {}^\circ\Sigma_v} M_{\Sigma_v}(e_v)\}| < \infty \ . \qquad (2)$$

Clearly, a finite Q fulfills this requirement.

4.2 Place and transition names of operator and plain boxes

As far as net refinement as such is concerned, the names (identities) of newly constructed transitions and places are irrelevant, provided that we always choose them fresh. However, in our approach to solving recursive definitions on boxes in section 5, the *names* of places and transitions are crucial since we use them to define the inclusion order on the domain of labelled nets.

A key to the construction of recursive nets is the use of labelled trees as place and transition names.[12] First of all, we shall assume that there are two disjoint infinite sets of place and transition names, $\mathsf{P}_{\mathsf{root}}$ and $\mathsf{T}_{\mathsf{root}}$. Each name $\eta \in \mathsf{P}_{\mathsf{root}} \cup \mathsf{T}_{\mathsf{root}}$ (or a pair (t, α), where t is a name in $\mathsf{T}_{\mathsf{root}}$ and α is a label in Lab) can be viewed as a special tree with a single root labelled with η (or (t, α)) which is also a leaf. Moreover, we shall allow more complicated trees as transition and place names, and use a linear notation to express such trees. To this end, the expression $x \lhd \mathcal{S}$, where x is a single root tree labelled with x (x is a name η or a pair (t, α)) and $\mathcal{S}$ is a multiset of trees, is a new tree where the trees of the multiset are appended (with their multiplicity) to the root. If $\mathcal{S} = \{p\}$ is a singleton multiset then $x \lhd \mathcal{S}$ will simply be denoted by $x \lhd p$, and if $\mathcal{S}$ is empty then $x \lhd \mathcal{S} = x$.

We shall further assume that in every operator box, all places and transitions are simply names (i.e., single root trees) from respectively $\mathsf{P}_{\mathsf{root}}$ and $\mathsf{T}_{\mathsf{root}}$. For the plain boxes, the trees used as names may be more complex. Each transition tree is a finite tree labelled with elements of $\mathsf{T}_{\mathsf{root}}$ (at the leaves) and $\mathsf{T}_{\mathsf{root}} \times \mathsf{Lab}$ (elsewhere), and each place tree is a possibly infinite (in depth and width) tree labelled with names from $\mathsf{P}_{\mathsf{root}}$ and $\mathsf{T}_{\mathsf{root}}$, which has the form:

$$t_1 \lhd t_2 \lhd \ldots \lhd t_n \lhd s \lhd \mathcal{S} \ ,$$

where $t_1, \ldots, t_n \in \mathsf{T}_{\mathsf{root}}$ ($n \geq 0$) are transition names and $s \in \mathsf{P}_{\mathsf{root}}$ is a place name (so that no confusion will be possible between transition-trees and place-trees: the latter always have a label from $\mathsf{P}_{\mathsf{root}}$ and the former never). We comprise all these trees (including the basic ones consisting only of a root as special cases) in our sets of allowed transition and place names, denoted respectively by $\mathsf{P}_{\mathsf{tree}}$ and $\mathsf{T}_{\mathsf{tree}}$. The definition of net refinement will be done in such a way that, provided all the names occurring in Ω are single root trees, and all the names occurring in $\mathbf{\Sigma}$ are in the sets of allowed names, then all the names in $\Omega(\mathbf{\Sigma})$ belong there, too.

4.3 Formal definition of net refinement $\Omega(\mathbf{\Sigma})$

Under the assumptions made about the operator box Ω and Ω-tuple of plain boxes $\mathbf{\Sigma}$, the result of a simultaneous substitution of boxes $\mathbf{\Sigma}$ for the transitions in Ω is a labelled net $\Omega(\mathbf{\Sigma})$ whose components are defined thus.

[12] We define such trees up to isomorphism.

Places The set of places of $\Omega(\mathbf{\Sigma})$ is defined as the (disjoint) union

$$\begin{aligned} S_{\Omega(\mathbf{\Sigma})} &= \bigcup_{v \in T_\Omega} \mathsf{ST}^v_{\mathsf{new}} \qquad\qquad \cup \bigcup_{s \in S_\Omega} \mathsf{SP}^s_{\mathsf{new}} \\ &= \bigcup_{v \in T_\Omega} \{\ v \lhd i \mid i \in \ddot{\Sigma}_v \} \cup \bigcup_{s \in S_\Omega} \{\ s \lhd (\{v \lhd x_v\}_{v \in {}^\bullet s} + \{w \lhd e_w\}_{w \in s^\bullet}) \mid \\ &\qquad\qquad\qquad\qquad\qquad\qquad x_v \in \Sigma_v^\circ \wedge e_w \in {}^\circ \Sigma_w \ \} \ . \end{aligned}$$

If s is an isolated place, ${}^\bullet s = s^\bullet = \emptyset$, then, by the definition of the tree appending operation, $\mathsf{SP}^s_{\mathsf{new}} = \{s\}$. Notice that the multiset of trees in the definition of a place

$$p \ = \ s \lhd (\{v \lhd x_v\}_{v \in {}^\bullet s} + \{w \lhd e_w\}_{w \in s^\bullet}) \ \in \ \mathsf{SP}^s_{\mathsf{new}} \tag{3}$$

is in fact a set, possibly infinite, because even in case there is a side-loop between s and $v = w$, then $x_v \neq e_w$ since x_v is an exit place and e_w is an entry place of Σ_v. The following notation is useful in manipulating the tree names upon which a newly constructed place is based.

Let y be a transition in Ω. Then for $p = v \lhd i$ in $\mathsf{ST}^v_{\mathsf{new}}$, we define $\mathsf{trees}^y(p) = \{i\}$ if $v = y$, and $\mathsf{trees}^y(p) = \emptyset$ if $v \neq y$; and, for $p \in \mathsf{SP}^s_{\mathsf{new}}$ as in (3), we define

$$\mathsf{trees}^y(p) = \begin{cases} \{x_y, e_y\} & \text{if } y \in {}^\bullet s \cap s^\bullet \\ \{x_y\} & \text{if } y \in {}^\bullet s \setminus s^\bullet \\ \{e_y\} & \text{if } y \in s^\bullet \setminus {}^\bullet s \\ \emptyset & \text{otherwise} \ . \end{cases}$$

Marking The marking of a place p in $\Omega(\mathbf{\Sigma})$ is defined in the following way:

$$M_{\Omega(\mathbf{\Sigma})}(p) \ = \ \begin{cases} M_{\Sigma_v}(i) & \text{if} \quad p = v \lhd i \in \mathsf{ST}^v_{\mathsf{new}} \\ M_\Omega(s) + \sum\limits_{v \in {}^\bullet s} M_{\Sigma_v}(x_v) + \sum\limits_{w \in s^\bullet} M_{\Sigma_w}(e_w) & \text{if} \quad p \in \mathsf{SP}^s_{\mathsf{new}} \ \text{ is as in (3)} \ . \end{cases} \tag{4}$$

Notice that if $s \in S_\Omega$ is an isolated place then $M_{\Omega(\mathbf{\Sigma})}(s) = M_\Omega(s)$, and that $M_{\Omega(\mathbf{\Sigma})}(p)$ is always a natural number due to (2).

Transitions The set of transitions of $\Omega(\mathbf{\Sigma})$ is defined as the union

$$T_{\Omega(\mathbf{\Sigma})} \ = \ \bigcup_{v \in T_\Omega} \mathsf{T}^v_{\mathsf{new}} \ = \ \bigcup_{v \in T_\Omega} \{(v, \alpha) \lhd R \mid R \in \mathsf{mult}(T_{\Sigma_v}) \wedge (\lambda_{\Sigma_v}(R), \alpha) \in \lambda_\Omega(v)\} \ .$$

Notice that the multiset R in $(v, \alpha) \lhd R$ will never be empty since no pair in $\lambda_\Omega(v)$ has the empty multiset as its left argument.

Similarly as for places, we will denote by $\mathsf{trees}(u)$ the multiset of transitions R upon which a newly constructed transition $u = (v, \alpha) \lhd R \in \mathsf{T}^v_{\mathsf{new}}$ is based.

Labelling The label of a node z in $\Omega(\mathbf{\Sigma})$ is defined in the following way:

$$\lambda_{\Omega(\mathbf{\Sigma})}(z) \ = \ \begin{cases} \mathsf{i} & \text{if} \quad z \in \mathsf{ST}^v_{\mathsf{new}} \\ \lambda_\Omega(s) & \text{if} \quad z \in \mathsf{SP}^s_{\mathsf{new}} \\ \alpha & \text{if} \quad z = (v, \alpha) \lhd R \in \mathsf{T}^v_{\mathsf{new}} \ . \end{cases}$$

Weights For a place p and transition u in $\mathsf{T}^v_{\mathsf{new}}$, the weight function is given by:

$$W_{\Omega(\mathbf{\Sigma})}(p,u) = \sum_{z \in \mathsf{trees}^v(p)} \sum_{t \in \mathsf{trees}(u)} W_{\Sigma_v}(z,t) \cdot \mathsf{trees}(u)(t)$$
$$W_{\Omega(\mathbf{\Sigma})}(u,p) = \sum_{z \in \mathsf{trees}^v(p)} \sum_{t \in \mathsf{trees}(u)} W_{\Sigma_v}(t,z) \cdot \mathsf{trees}(u)(t) \ .$$

Figure 15 shows a complete example of net refinement.

At this point, it is not certain that, e.g., if we start from static boxes, the result of a refinement will be a static box. Indeed, it will turn out that some extra conditions need to be introduced for this to hold. However, the mechanism we have described is compositional since the definition is based on, and respects, the structure of the components.

Proposition 2. *$\Omega(\mathbf{\Sigma})$ is a plain box which is unmarked if and only if so is Ω and each plain box in $\mathbf{\Sigma}$.* □

Notice that the definition of refinement is constructed in such a way that:

- The e/i/x-labelling of $\Omega(\mathbf{\Sigma})$ depends only on the e/i/x-labelling of Ω (and not on that of $\mathbf{\Sigma}$).
- The net structure (connectivity) of $\Omega(\mathbf{\Sigma})$ depends on the e/i/x-labelling of $\mathbf{\Sigma}$ (and, of course, on the structure of all participating nets), but not on the e/i/x-labelling of Ω. More precisely, the e and x labelling of Σ_v's places is used (only) for linking with the pre- and post-places of v in Ω.

4.4 Equivalence notions

Net refinement has a well formed graph-theoretic definition and, as a result, it is insensitive to taking isomorphic instances of nets. That is, isomorphism is preserved through net refinement. Below we consider two instances of net refinement, $\Omega(\mathbf{\Sigma})$ and $\Omega'(\mathbf{\Sigma}')$.

Theorem 1. *Let $\sim \in \{ \mathsf{iso} , \mathsf{iso_T} \}$ and ψ be an isomorphism from Ω to Ω' such that $\Sigma_v \sim \Sigma'_{\psi(v)}$, for every $v \in T_\Omega$. Then $\Omega(\mathbf{\Sigma}) \sim \Omega'(\mathbf{\Sigma}')$.* □

Theorem 2. *Let $\sim \in \{ \mathsf{iso_S} , \mathsf{iso_{ST}} \}$ and Ω $\mathsf{iso_S}$ Ω' and ψ be an isomorphism from Ω_0 to Ω'_0, where Ω_0 and Ω'_0 are obtained from respectively Ω and Ω' by replacing all places by their duplication equivalence classes. Moreover, let us assume that $\Sigma_v \sim \Sigma'_{\psi(v)}$, for every $v \in T_\Omega$. Then $\Omega(\mathbf{\Sigma}) \sim \Omega'(\mathbf{\Sigma}')$.* □

We do not consider $\mathsf{iso_T}$- nor $\mathsf{iso_{ST}}$-equivalence for operator boxes, since these equivalences do not preserve the arity (number of transitions) of operator boxes. Moreover, the behavioural equivalences, $\cong$ and $\approx$, will only be considered for concrete box algebras as they are highly sensitive to the particular choice of the operator and operand boxes.

5 Recursive Equations on Boxes

This section presents a method of solving recursive equations in the domain of boxes, following a common practice of using fixpoints and limit constructions. We will apply the standard methods, but also show that if they are applied to Petri nets, a number of specific problems arise which can be solved with help of the tree device for naming new places and transitions in net refinement.

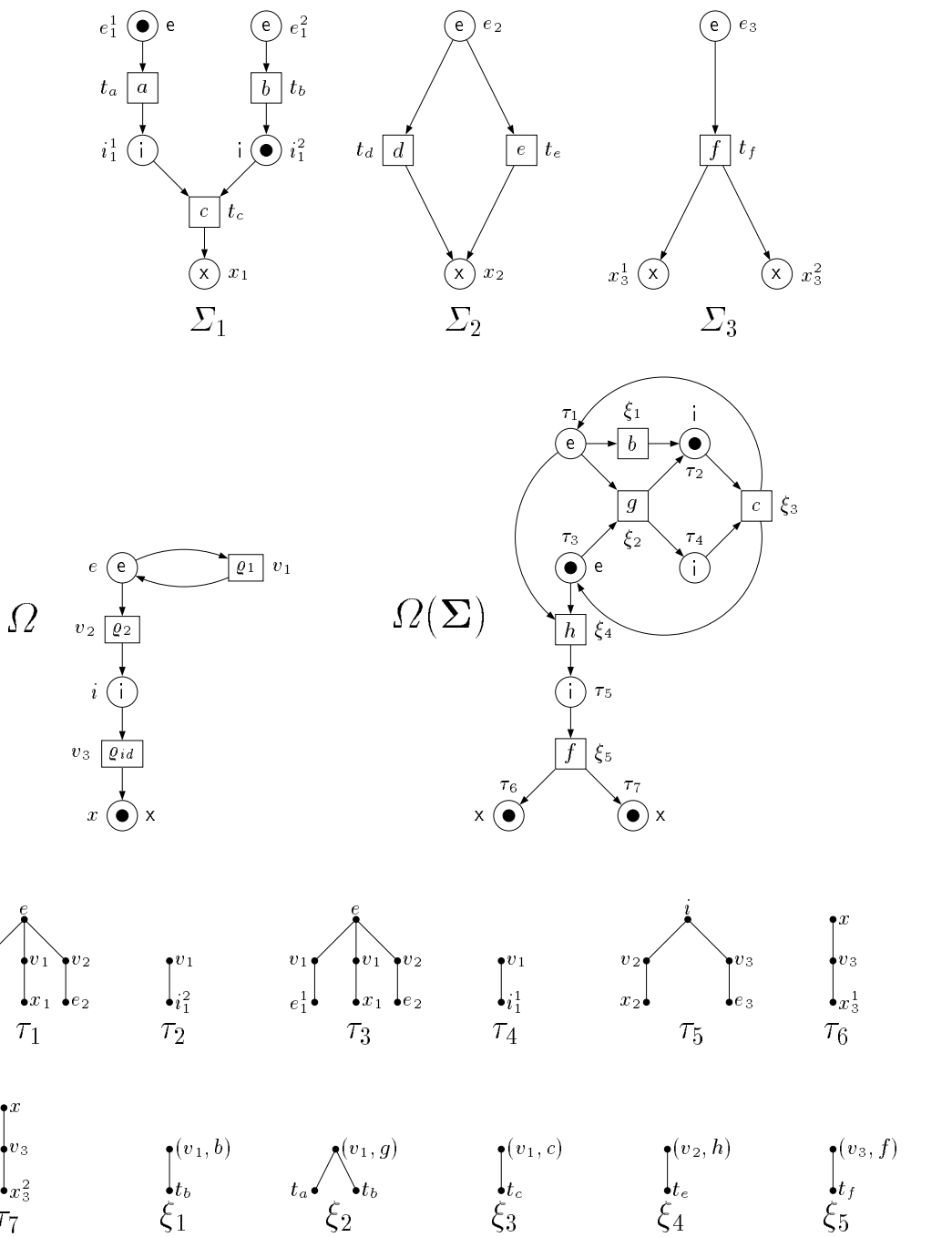


Fig. 15. Example of net refinement where the (complex) marking of Ω is $\mathcal{M}_\Omega = (\{x\}, \{v_1\})$ and $\varrho_1 = \varrho_{id} \backslash \{(\{a\}, a)\} \cup \{(\{a, b\}, g)\}$ and $\varrho_2 = \varrho_{id} \backslash \{(\{e\}, e), (\{d\}, d)\} \cup \{(\{e\}, h)\}$.

Inclusion order on labelled nets We equip the set of labelled nets with an *ordering relation*, $\sqsubseteq$. For two labelled nets Σ and Θ, we denote $\Sigma \sqsubseteq \Theta$ if $S_\Sigma \subseteq S_\Theta$, $T_\Sigma \subseteq T_\Theta$, $W_\Sigma = W_\Theta|_{(S_\Sigma \times T_\Sigma) \cup (T_\Sigma \times S_\Sigma)}$, $\lambda_\Sigma = \lambda_\Theta|_{S_\Sigma \cup T_\Sigma}$ and $M_\Sigma = M_\Theta|_{S_\Sigma}$. The relation $\sqsubseteq$ is a partial order, and if $\Sigma \sqsubseteq \Theta$ then ${}^\circ\Sigma \subseteq {}^\circ\Theta$ and $\Sigma^\circ \subseteq \Theta^\circ$. A $\sqsubseteq$-*chain* is an infinite sequence

$$\chi \;=\; \Sigma^0 \;\; \Sigma^1 \;\; \Sigma^2 \;\; \ldots$$

of labelled nets such that $\Sigma^k \sqsubseteq \Sigma^{k+1}$ for every $k \geq 0$. Each such chain has a unique least upper bound or *limit*, denoted by $\bigsqcup \chi$; we will call the Σ^k's *approximations* of $\bigsqcup \chi$. The places and transitions of $\bigsqcup \chi$ are respectively the union of the places and transitions of all labelled nets in the chain. For any two elements x and y of $\bigsqcup \chi$ (one of them a place, the other a transition), the weight $W(x,y)$ is defined as the weight between x and y in the first Σ^k to which both of them belong (note that, by the definition of $\sqsubseteq$, this weight does not change in subsequent approximations). Finally, for a place or transition x in $\bigsqcup \chi$, the label $\lambda(x)$ is defined as the label of x in the first and hence in all Σ^k containing x, and for a place s in $\bigsqcup \chi$, the marking $M(s)$ is defined as the marking of s in the first and hence in all Σ^k containing s. In other words,

$$\bigsqcup \chi \;=\; (\bigcup_{k=0}^{\infty} S_{\Sigma^k} \;,\; \bigcup_{k=0}^{\infty} T_{\Sigma^k} \;,\; \bigcup_{k=0}^{\infty} W_{\Sigma^k} \;,\; \bigcup_{k=0}^{\infty} \lambda_{\Sigma^k} \;,\; \bigcup_{k=0}^{\infty} M_{\Sigma^k}) \;.$$

Let $\mathcal{C}$ be a property of labelled nets. We will say that $\mathcal{C}$ is *limit-preserved* if for every $\sqsubseteq$-chain χ of labelled nets satisfying $\mathcal{C}$, it is the case that $\bigsqcup \chi$ satisfies $\mathcal{C}$ as well.

Proposition 3. *The following properties of labelled nets are limit-preserved.*

(1) All markings reachable from the initial marking are safe. *(safeness)*
(2) All markings reachable from the initial marking are clean. *(cleanness)*
(3) There is at least one entry place and at least one exit place. *(ex-restrictedness)*
(4) There is no arc incoming to an entry place. *(e-directedness)*
(5) There is no arc outgoing from an exit place. *(x-directedness)*
(6) There is no marking reachable from the initial marking which simultaneously marks an entry place and an exit place. *(ex-exclusiveness)*
(7) Every transition has a non-empty pre-set and non-empty post-set. *(T-restrictedness)*
(8) The initial marking is empty. *(being a static net)*
(9) All transition labels are constant relabellings. *(being a plain net)*

□

5.1 Solving recursive equations

Let $\mathcal{X}$ be a non-empty, possibly infinite, set of *recursion variables*. We consider a system of *equations on boxes* of the following form (one equation per variable X in $\mathcal{X}$):

$$X \;\stackrel{\text{df}}{=}\; \Omega_X(\mathbf{\Delta}_X) \tag{5}$$

where Ω_X is an unmarked operator box, and $\mathbf{\Delta}_X$ is an Ω_X-tuple of recursion variables and unmarked plain boxes, called *rec-boxes*, Box^{rec}. That is, $\mathbf{\Delta}_X \;:\; T_{\Omega_X} \to \mathcal{X} \cup \mathsf{Box}^{rec}$. Now,

given a mapping $\mathsf{sol} : \mathcal{X} \to \mathsf{Box}^{rec}$ assigning a rec-box to every variable in $\mathcal{X}$, we will denote by $\mathbf{\Delta}_X[\mathsf{sol}]$ the Ω_X-tuple obtained from $\mathbf{\Delta}_X$ by replacing each variable Y by $\mathsf{sol}(Y)$. That is, $\mathbf{\Delta}_X[\mathsf{sol}] \ : \ T_{\Omega_X} \to \mathsf{Box}^{rec}$ is such that for every $v \in T_{\Omega_X}$,

$$\mathbf{\Delta}_X[\mathsf{sol}](v) \ = \ \begin{cases} \mathsf{sol}(Y) \text{ if } & \Delta_{X,v} = Y \in \mathcal{X} \\ \Delta_{X,v} \ \text{ if } & \Delta_{X,v} \in \mathsf{Box}^{rec} \ . \end{cases}$$

A *solution* of the system of equations (5) is an assignment sol such that, for every variable $X \in \mathcal{X}$, $\mathsf{sol}(X) \ = \ \Omega_X(\mathbf{\Delta}_X[\mathsf{sol}])$. Notice that since all the nets in $\mathbf{\Delta}_X[\mathsf{sol}]$ are unmarked, the application of Ω_X to $\mathbf{\Delta}_X[\mathsf{sol}]$ satisfies the condition (2) stated in section 4.1.

To simplify the presentation, we shall first only consider a *single* recursive equation for $\mathcal{X} = \{X\}$:

$$X \ \stackrel{\text{df}}{=} \ \Omega(\mathbf{\Delta}) \ . \tag{6}$$

In this case, we identify an assignment $\mathsf{sol} : \{X\} \to \mathsf{Box}^{rec}$ with the rec-box $\Sigma = \mathsf{sol}(X)$ and denote $\mathbf{\Delta}[\Sigma]$ rather than $\mathbf{\Delta}[\mathsf{sol}]$, calling Σ a *solution* of (6) whenever $\Sigma = \Omega(\mathbf{\Delta}[\Sigma])$. Without loss of generality, we shall also assume that for every rec-box Δ_v in $\mathbf{\Delta}$, it is the case that $T_{\Delta_v} \subseteq \mathsf{T_{root}} \backslash T_\Omega$ and $S_{\Delta_v} \subseteq \mathsf{P_{root}} \backslash S_\Omega$.[13]

A recursive equation may have different, even non-isomorphic, solutions;[14] each such solution being uniquely identified by the sets of entry and exit places.

Theorem 3. *Let Σ and Θ be solutions of (6).*

(1) The sets of transitions of Σ and Θ and their labellings are the same.
(2) $\Sigma \sqsubseteq \Theta$ if and only if ${}^\circ\Sigma \subseteq {}^\circ\Theta$ and $\Sigma^\circ \subseteq \Theta^\circ$.
(3) $\Sigma = \Theta$ if and only if ${}^\circ\Sigma = {}^\circ\Theta$ and $\Sigma^\circ = \Theta^\circ$. □

Using fixpoints The right-hand side of (6) induces a mapping $\mathbf{F} : \mathsf{Box}^{rec} \to \mathsf{Box}^{rec}$ defined, for every rec-box Σ, by

$$\mathbf{F}(\Sigma) \ = \ \Omega(\mathbf{\Delta}[\Sigma]) \ .$$

Hence solving the equation (6) amounts to finding the fixpoints of $\mathbf{F}$. Following the usual treatment of fixpoint theory, we now proceed by trying to establish the monotonicity and continuity of $\mathbf{F}$. We say that $\mathbf{F}$ is $\sqsubseteq$-*monotonic* if, for all rec-boxes Σ and Θ, $\Sigma \sqsubseteq \Theta$ implies $\mathbf{F}(\Sigma) \sqsubseteq \mathbf{F}(\Theta)$, and $\sqsubseteq$-*continuous* if for every $\sqsubseteq$-chain $\chi = \Sigma^0\Sigma^1 \ldots$ of rec-boxes,

$$\mathbf{F}(\bigsqcup \chi) \ = \ \bigsqcup \mathbf{F}(\Sigma^0)\mathbf{F}(\Sigma^1) \ldots$$

It turns out that $\mathbf{F}$ is $\sqsubseteq$-monotonic, but it is not $\sqsubseteq$-continuous without making an additional assumption. More precisely, $\mathbf{F}$ is $\sqsubseteq$-continuous if and only if the recursive equation (6) is X-*finitary* which means that, for every place s in Ω, $|\{v \in {}^\bullet s \cup s^\bullet \mid \Delta_v = X\}| < \infty$.[15]

[13] Recall that, by the definition of an operator box, $T_\Omega \subseteq \mathsf{T_{root}}$ and $S_\Omega \subseteq \mathsf{P_{root}}$.
[14] For a recursive system in the box algebra this does not matter since all the solutions will have *identical* behaviour.
[15] That is, if every place in the operator box has 'access' to only finitely many instances of the variable X.

Although the $\sqsubseteq$-continuity of $\mathbf{F}$ does not hold in the general case, there is a closely related and for our purposes sufficient, property of $\mathbf{F}$ obtained by strengthening the inclusion order. For two rec-boxes Σ and Θ, we shall denote $\Sigma \mathrel{\widetilde{\sqsubseteq}} \Theta$ if $\Sigma \sqsubseteq \Theta$, ${}^\circ\Sigma = {}^\circ\Theta$ and $\Sigma^\circ = \Theta^\circ$. We then define $\widetilde{\sqsubseteq}$-chains, $\widetilde{\sqsubseteq}$-monotonicity and $\widetilde{\sqsubseteq}$-continuity in the same way as the corresponding notions for $\sqsubseteq$. And since $\widetilde{\sqsubseteq}$-chains are also $\sqsubseteq$-chains, proposition 3 and other properties enjoyed by the latter hold for $\widetilde{\sqsubseteq}$-chains as well. Crucially, we have that

Theorem 4. *$\mathbf{F}$ is $\widetilde{\sqsubseteq}$-monotonic and $\widetilde{\sqsubseteq}$-continuous.* □

Thus one can attempt to solve equation (6) by constructing a $\widetilde{\sqsubseteq}$-chain of rec-boxes formed through successive applications of the mapping $\mathbf{F}$ to some initial approximation Υ.

For a rec-box Υ, we denote by $\mathbf{F}^*(\Upsilon)$ the infinite sequence $\mathbf{F}^0(\Upsilon)\mathbf{F}^1(\Upsilon)\mathbf{F}^2(\Upsilon)\ldots$ where $\mathbf{F}^0(\Upsilon) = \Upsilon$ and $\mathbf{F}^n(\Upsilon) = \mathbf{F}(\mathbf{F}^{n-1}(\Upsilon))$, for every $n \geq 1$. We then call Υ a $\widetilde{\sqsubseteq}$-*seed* for the equation (6) if $\Upsilon \mathrel{\widetilde{\sqsubseteq}} \mathbf{F}(\Upsilon)$; in such a case, by the $\widetilde{\sqsubseteq}$-monotonicity of $\mathbf{F}$ and proposition 3(*3,7,8,9*), we have that $\mathbf{F}^*(\Upsilon)$ is a $\widetilde{\sqsubseteq}$-chain of rec-boxes. And such a chain always yields a solution.

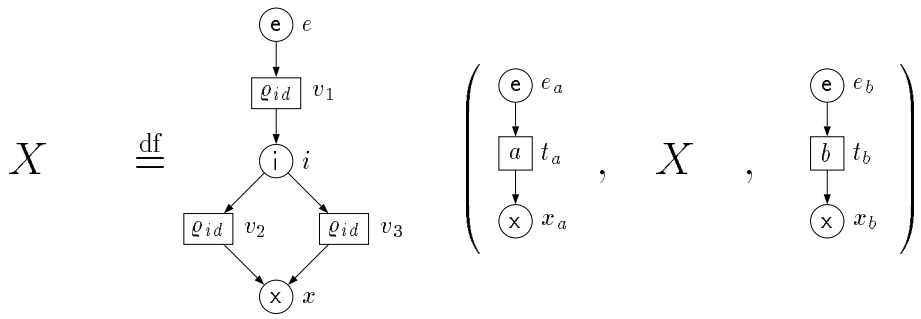
Theorem 5. *If Υ is a $\widetilde{\sqsubseteq}$-seed then $\bigsqcup \mathbf{F}^*(\Upsilon)$ is a solution of (6).* □

Moreover, it suffices to consider ex-boxes as the only seeds.

Theorem 6. *If Σ is a solution of (6) then the ex-box $\Upsilon = \mathsf{ex}({}^\circ\Sigma, \Sigma^\circ)$ is a $\widetilde{\sqsubseteq}$-seed such that $\bigsqcup \mathbf{F}^*(\Upsilon) = \Sigma$.* □

Before looking closer at the seed boxes, we present an example.

Example of the limit construction Consider the following equation which intuitively corresponds to $X \stackrel{\text{df}}{=} a;(X \,[]\, b)$:



Let us examine how a solution of the net equation (7) might be constructed using the limit construction. Suppose that there exists already an approximation Υ^0 given as a starting point, a hypothetical seed box. Then inserting that approximation in the place of the X on the right-hand side of (7) yields a new box $\Upsilon^1 = \Omega(\Sigma_a, \Upsilon^0, \Sigma_b)$, where Ω is the operator box in this equation and Σ_a and Σ_b are the two plain boxes, which is a slightly better approximation.

We then construct another approximation $\Upsilon^2 = \Omega(\Sigma_a, \Upsilon^1, \Sigma_b)$ by the same principle, and so on. We already know that if Υ^0 is a $\widetilde{\sqsubseteq}$-seed then the limit of the sequence $\Upsilon^0\Upsilon^1\ldots$

so constructed exists and solves the equation (7). We now claim that the ex-box Υ^0 shown in figure 16 (with these particular place names) is a good seed for (7). Figure 16 shows also the first three approximations resulting from this seed and, finally, it shows $\Sigma = \bigsqcup \mathbf{F}^*(\Upsilon^0)$, which is the limit of the infinite series of boxes whose first three elements are depicted. The reader may check that this box satisfies (7) in the strictest possible sense of the word. To see this, replace the variable X on the right-hand side of the equation by the box Σ shown in figure 16, perform net refinement $\Omega(\Sigma_a, \Sigma, \Sigma_b)$, and check that the result is not just isomorphic but even *identical* to Σ, i.e., it has the same places and the same transitions with the same connections and the same labels.

Thus, once an appropriate starting point Υ^0 has been chosen, the approximation of a solution is straightforward (we just keep refining successive approximations into the right-hand side of the equation under consideration and compute the limit) and leads to a solution in all cases.

Deriving seed boxes To find a $\widetilde{\sqsubseteq}$ -seed Υ for the recursive equation (6), we need to ensure that $\Upsilon \sqsubseteq \mathbf{F}(\Upsilon)$, ${}^\circ\Upsilon = {}^\circ\mathbf{F}(\Upsilon)$ and $\Upsilon^\circ = \mathbf{F}(\Upsilon)^\circ$ hold for Υ and the next approximation $\mathbf{F}(\Upsilon)$. One might ask why the rec-box Υ^0 in figure 16 was a suitable starting point for solving the equation (7). First, it is a good idea to start always with an ex-box, because these are minimal with respect to the number of transitions and because, as a consequence of theorem 6, all solutions can be so obtained. Moreover, one only needs to ensure that ${}^\circ\Upsilon = {}^\circ\mathbf{F}(\Upsilon)$ and $\Upsilon^\circ = \mathbf{F}(\Upsilon)^\circ$, since then $\Upsilon \sqsubseteq \mathbf{F}(\Upsilon)$ always holds. It is even better (if possible) to start with an ex-box containing only one entry and one exit place, as such boxes are minimal with respect to the ordering $\sqsubseteq$. Let us now fix a hypothetical starting ex-box

$$\Upsilon \;=\; \mathsf{ex}(\mathcal{P}, \mathcal{R})$$

and develop conditions for it to be a $\widetilde{\sqsubseteq}$ -seed. First, $\mathcal{P}$ and $\mathcal{R}$ should be non-empty and disjoint. Second, with a view to ensure that ${}^\circ\Upsilon = {}^\circ\mathbf{F}(\Upsilon)$ and $\Upsilon^\circ = \mathbf{F}(\Upsilon)^\circ$, some conditions need to be imposed on $\mathcal{P}$ and $\mathcal{R}$, essentially by performing the refinement at the entry and exit places. As both place sets are sets of trees, these conditions can be translated into equalities over sets of place trees.

We define a mapping $\mathbf{f} : 2^{\mathsf{P}_{\mathsf{tree}}} \times 2^{\mathsf{P}_{\mathsf{tree}}} \to 2^{\mathsf{P}_{\mathsf{tree}}} \times 2^{\mathsf{P}_{\mathsf{tree}}}$ associated with $\mathbf{F}$, and through $\mathbf{F}$ with the recursive equation (6) and thus also with Ω, so that for all $P, R \subseteq \mathsf{P}_{\mathsf{tree}}$,

$$\mathbf{f}(P,R) \;=\; \Big(\bigcup_{s \in {}^\circ\Omega} \mathbf{f}_s(P,R) \quad , \quad \bigcup_{s \in \Omega^\circ} \mathbf{f}_s(P,R) \Big) \tag{8}$$

where each $\mathbf{f}_s(P,R)$ comprises all place trees $s \lhd (\{v \lhd p_v\}_{v \in s^\bullet} + \{w \lhd r_w\}_{w \in {}^\bullet s})$ such that:

$$p_v \;\in\; \begin{cases} P & \text{if} \quad \Delta_v = X \\ {}^\circ\Delta_v & \text{if} \quad \Delta_v \neq X \end{cases} \qquad \text{and} \qquad r_w \;\in\; \begin{cases} R & \text{if} \quad \Delta_w = X \\ \Delta_w^\circ & \text{if} \quad \Delta_w \neq X \end{cases} .$$

A pair of sets $(P,R) \in 2^{\mathsf{P}_{\mathsf{tree}}} \times 2^{\mathsf{P}_{\mathsf{tree}}}$ is a *positive fixpoint* of $\mathbf{f}$ if $\mathbf{f}(P,R) = (P,R)$ and $P \neq \emptyset \neq R$. The function $\mathbf{f}$ can be used to calculate the entry and exit places of $\mathbf{F}(\Upsilon)$ since $\mathbf{f}(\mathcal{P},\mathcal{R}) = ({}^\circ\mathbf{F}(\Upsilon), \mathbf{F}(\Upsilon)^\circ)$. As a result, $\Upsilon = \mathsf{ex}(\mathcal{P},\mathcal{R})$ is a $\widetilde{\sqsubseteq}$ -seed if and only if $(\mathcal{P},\mathcal{R})$ is a positive fixpoint of $\mathbf{f}$.

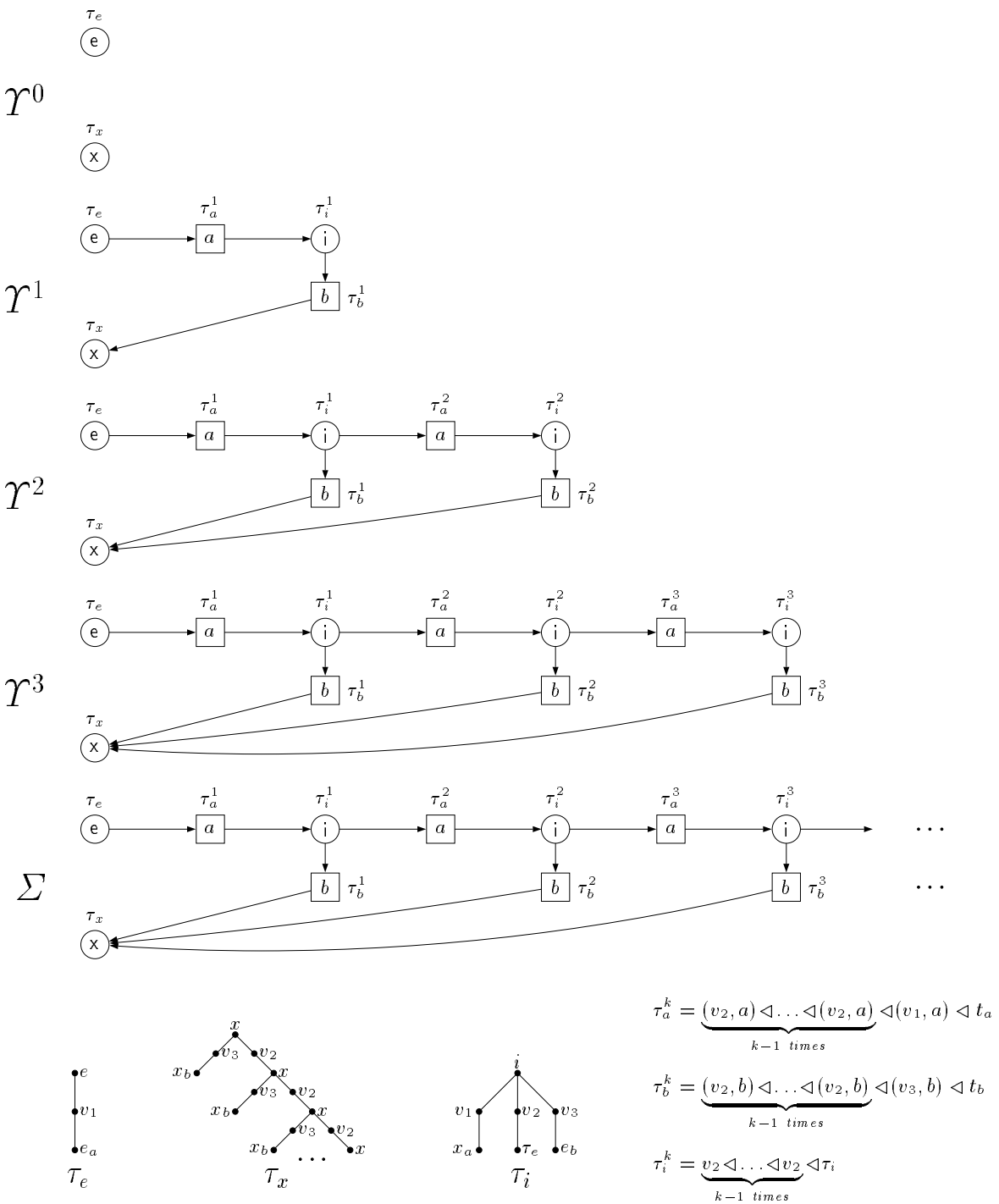


Fig. 16. Four approximations for equation (7) and the resulting solution.

Thus deriving $\widetilde{\sqsubseteq}$ -seeds for (6) amounts to finding positive fixpoints of $\mathbf{f}$. Using the notation introduced in (8), the latter amounts to finding non-empty sets $\mathcal{P}$ and $\mathcal{R}$ solving the following *tree equations*:

$$\begin{array}{rcl} \mathcal{P} & = & \bigcup_{s \in {}^\circ\Omega} \ \mathbf{f}_s(\mathcal{P},\mathcal{R}) \\ \mathcal{R} & = & \bigcup_{s \in \Omega^\circ} \ \mathbf{f}_s(\mathcal{P},\mathcal{R}) \ . \end{array} \tag{9}$$

The function $\mathbf{f}$ is a monotonic mapping on the domain $(2^{\mathsf{P}_{\mathsf{tree}}} \times 2^{\mathsf{P}_{\mathsf{tree}}}, \subseteq)$ which is a complete lattice with set union as the 'join' and intersection as the 'meet' operation.[16] It therefore follows that Tarski's theorem [19] can be applied.

Theorem 7. *The set of fixpoints of* $\mathbf{f}$ *is a complete lattice whose maximal element,* $\mathsf{fix}_{\mathsf{max}}$*, is positive. Moreover, there is at least one positive fixpoint which is minimal among all the positive fixpoints of* $\mathbf{f}$*.* □

This leads (although not immediately) to the main result of this section.

Theorem 8. *The recursive equation (6) has at least one solution. Moreover, the set of solutions has a unique maximal element and at least one minimal element (which may not be unique).* □

One might be tempted to always take the minimal fixpoint of $\mathbf{f}$, $\mathsf{fix}_{\mathsf{min}}$, and treat the corresponding solution as *the* solution of (6) since, by theorem 3(2), such a solution would be minimal. Unfortunately, the minimal fixpoint may not be positive making it unsuitable for our purposes because boxes are ex-restricted. In general, we have the following.

If fix is a unique positive fixpoint of $\mathbf{f}$ then (6) has a unique solution which can be derived from $(\mathcal{P},\mathcal{R}) = \mathsf{fix}$. If $\mathsf{fix}_{\mathsf{min}}$ is positive then (6) has a unique minimal solution with respect to $\sqsubseteq$ derived from the minimal fixpoint, $(\mathcal{P},\mathcal{R}) = \mathsf{fix}_{\mathsf{min}}$. There is *always* a unique maximal solution of (6) (cf. theorem 7) which can be derived from $(\mathcal{P},\mathcal{R}) = \mathsf{fix}_{\mathsf{max}}$. This solution is of particular importance (see section 6.4) and may also be given in closed form (see [8]). If fix is a fixpoint of $\mathbf{f}$ which is minimal among all the positive fixpoints of $\mathbf{f}$ then (6) has a minimal solution which can be derived from $(\mathcal{P},\mathcal{R}) = \mathsf{fix}$.

There is a simple structural condition guaranteeing that $\mathbf{f}$ is a constant mapping and therefore trivially has a unique fixpoint. The recursive equation (6) is *front-guarded* if no entry place of Ω has a direct access to the variable X; that is, if $\Delta_v \neq X$, for every $v \in {}^\bullet({}^\circ\Omega) \cup ({}^\circ\Omega)^\bullet$. Similarly, it is *rear-guarded* if $\Delta_v \neq X$, for every $v \in {}^\bullet(\Omega^\circ) \cup (\Omega^\circ)^\bullet$. And (6) is *guarded* if it is both front- and rear-guarded.

Theorem 9. *If (6) is guarded then it has a unique solution.* □

5.2 Finitary equations

If the recursive equation (6) is X-finitary then $\mathbf{f}$ is continuous. We can then calculate some positive fixpoints of $\mathbf{f}$ by taking $(\mathcal{P},\mathcal{R}) = \bigcup_{i=0}^{\infty} \mathbf{f}^{\,i}(\{\tau_e\},\{\tau_x\})$ provided that τ_e and τ_x are place trees satisfying $(\{\tau_e\},\{\tau_x\}) \subseteq \mathbf{f}(\{\tau_e\},\{\tau_x\})$. And there always exists at least one pair of such trees (for instance, τ_e and τ_x in figure 16). Moreover, since (6) is X-finitary, the

[16] All operations, including $\subseteq$, are carried out component-wise.

mapping $\mathbf{F}$ is $\sqsubseteq$-continuous rather than only $\widetilde{\sqsubseteq}$-continuous. An immediate consequence of this fact is that the definition of a seed box can be relaxed, by decreeing that a rec-box Υ is a for equation (6) if $\Upsilon \sqsubseteq \mathbf{F}(\Upsilon)$. And the results obtained previously for $\widetilde{\sqsubseteq}$-seed boxes carry over to the modified setting.

Theorem 10. *If Υ is a $\sqsubseteq$-seed for an X-finitary (6) then $\bigsqcup \mathbf{F}^*(\Upsilon)$ is a solution.* □

The search for $\sqsubseteq$-seeds can be carried out similarly as in section 5.1. In particular, Υ is a $\sqsubseteq$-seed if and only if $\mathcal{P}$ and $\mathcal{R}$ are non-empty and $(\mathcal{P},\mathcal{R}) \subseteq \mathbf{f}(\mathcal{P},\mathcal{R})$. What is more, as we already mentioned, there is always a pair of *singleton* sets $\mathcal{P} = \{\tau_e\}$ and $\mathcal{R} = \{\tau_x\}$ satisfying $(\mathcal{P},\mathcal{R}) \subseteq \mathbf{f}(\mathcal{P},\mathcal{R})$. It therefore follows that in the X-finitary case it is possible to obtain a solution of (6) by starting from a very small ex-box.

Theorem 11. *If (6) is X-finitary then it has a solution which can be approximated from a $\sqsubseteq$-seed with exactly two places and no transitions.* □

5.3 Two further recursion examples

Let us first return to the equation (7). In this case, to find the fixpoints of $\mathbf{f}$ we need to solve the following instance of the tree equations (9):

$$\begin{aligned}\mathcal{P} &= \{e \lhd v_1 \lhd e_a\}\\ \mathcal{R} &= \{x \lhd \{v_2 \lhd t \ , \ v_3 \lhd x_b\} \mid t \in \mathcal{R}\} \ .\end{aligned}$$

There are exactly two solutions, $\mathsf{fix}_{\mathsf{min}} = (\{\tau_e\},\emptyset)$ and $\mathsf{fix}_{\mathsf{max}} = (\{\tau_e\},\{\tau_x\})$, where τ_e and τ_x are as in figure 16. Since $\mathsf{fix}_{\mathsf{min}}$ is not positive, there is only one solution of the equation (7) which can be obtained from the $\widetilde{\sqsubseteq}$-seed $\Upsilon = \mathsf{ex}(\{\tau_e\},\{\tau_x\})$, shown in figure 16.

The first recursion example was front-guarded, but not rear-guarded. The next example is neither front- nor rear-guarded:

$$X \quad \stackrel{\text{df}}{=} \quad \left(\begin{array}{ccc} \mathsf{e}\ e_{\|}^1 & \mathsf{e}\ e_{\|}^2 & \mathsf{e}\ e_a \\ \boxed{\varrho_{id}}\ v_{\|}^1 & \boxed{\varrho_{id}}\ v_{\|}^2 & \boxed{a}\ t_a \\ \mathsf{x}\ x_{\|}^1 & \mathsf{x}\ x_{\|}^2 & \mathsf{x}\ x_a \end{array} \ , \quad X \right) \tag{10}$$

Intuitively, this corresponds to $X \stackrel{\text{df}}{=} a \| X$. To construct a $\widetilde{\sqsubseteq}$-seed, we need to solve the following instance of the tree equations for the auxiliary function $\mathbf{f}$:

$$\begin{aligned}\mathcal{P} &= \{\, e_{\|}^1 \lhd v_{\|}^1 \lhd e_a \,\} \cup \{\, e_{\|}^2 \lhd v_{\|}^2 \lhd t \ \mid \ t \in \mathcal{P} \,\}\\ \mathcal{R} &= \{\, x_{\|}^1 \lhd v_{\|}^1 \lhd x_a \,\} \cup \{\, x_{\|}^2 \lhd v_{\|}^2 \lhd t \ \mid \ t \in \mathcal{R} \,\}.\end{aligned} \tag{11}$$

And there is a one-to-one correspondence between the positive solutions of (11) and the solutions of (10). But now we have several positive solution; for example, the minimal fixpoint

containing only finite trees, and the maximal fixpoint which contains infinite trees as well. More precisely, we have $\mathsf{fix}_{\mathsf{min}} = (P, R)$ and $\mathsf{fix}_{\mathsf{max}} = (P \cup \{\tau_e\}, R \cup \{\tau_x\})$, where:

$$P = \{\tau_e^1, \tau_e^2, \tau_e^3, \ \ldots\} \qquad \tau_e = e_{\|}^2 \lhd v_{\|}^2 \lhd e_{\|}^2 \lhd v_{\|}^2 \lhd e_{\|}^2 \lhd v_{\|}^2 \ \cdots$$
$$R = \{\tau_x^1, \tau_x^2, \tau_x^3, \ \ldots\} \qquad \tau_x = x_{\|}^2 \lhd v_{\|}^2 \lhd x_{\|}^2 \lhd v_{\|}^2 \lhd x_{\|}^2 \lhd v_{\|}^2 \ \cdots$$

where $\tau_e^1 = e_{\|}^1 \lhd v_{\|}^1 \lhd e_a$ and $\tau_x^1 = x_{\|}^1 \lhd v_{\|}^1 \lhd x_a$ and $\tau_e^{k+1} = e_{\|}^2 \lhd v_{\|}^2 \lhd \tau_e^k$ and $\tau_x^{k+1} = x_{\|}^2 \lhd v_{\|}^2 \lhd \tau_x^k$, for $k \geq 1$. In all, there are four positive fixpoints which determine four starting boxes to begin the approximation of a solution of (10): Σ^{min} using $\mathsf{fix}_{\mathsf{min}}$, $\Sigma^{max:min}$ using $(P \cup \{\tau_e\}, R)$, $\Sigma^{min:max}$ using $(P, R \cup \{\tau_x\})$ and Σ^{max} using $\mathsf{fix}_{\mathsf{max}}$. Depending on the choice, one of the four different solutions of (10) is obtained, shown in figure 17. It is not hard to check that all four boxes are indeed solutions of (10) in the same strict sense as before.

The interleaving behaviour — and also behaviour in terms of finite steps — of the four solutions is the same. However, if infinite steps were allowed then the minimal solution, Σ^{min}, would have a terminating behaviour from the initial marking while the other three would not (since the initial token on an isolated entry place cannot be removed, as well as it is impossible to put a token onto an isolated exit place). Thus, for instance, if X was the first part of a sequential composition, then the second part of that composition could be executed using the minimal solution but not using the remaining three. Even more puzzling is that if we allowed infinite steps then $\overline{\Sigma^{max:min}}$ would not be clean, and no longer safe if we added the redo transition. We will return to this problem in section 6.4.

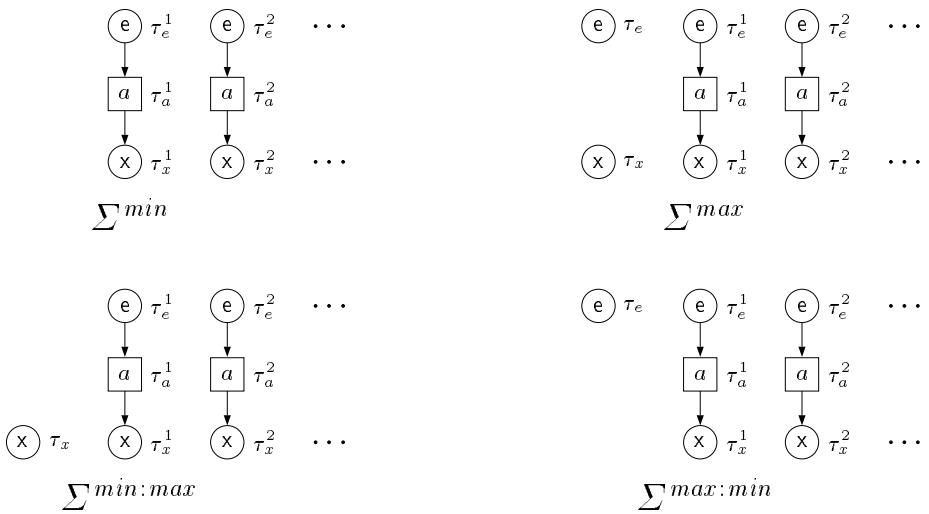


Fig. 17. The solutions of equation (10).

As the next equation – which informally corresponds to $X \stackrel{\text{df}}{=} X$ – consider the following:

$$X \quad \stackrel{\text{df}}{=} \quad \begin{array}{c} (\mathsf{e})\ e \\ \downarrow \\ \boxed{\varrho_{id}}\ v \\ \downarrow \\ (\mathsf{x})\ x \end{array} \quad \left(\quad X \quad \right) \tag{12}$$

It leads to the following tree equations for the entry and exit places of the starting box:

$$\mathcal{P} = \{\, e \lhd v \lhd t \;\mid\; t \in \mathcal{P} \,\}$$
$$\mathcal{R} = \{\, x \lhd v \lhd t \;\mid\; t \in \mathcal{R} \,\}$$

which means that $\mathbf{f}$ has a unique positive fixpoint

$$(\mathcal{P}\ ,\ \mathcal{R}) \;=\; (\ \{\, e \lhd v \lhd e \lhd v \lhd \ldots \}\ \ ,\ \ \{\, x \lhd v \lhd x \lhd v \lhd \ldots \}\ )\ .$$

In this case, the approximations never change anything, and the box solving the equation is $\mathsf{ex}(\mathcal{P}, \mathcal{R})$. Note that despite the colloquial meaning of $X \stackrel{\text{df}}{=} X$, no box with a transition solves (12). Thus solving recursive equation by boxes with least behaviour is already in-built in our approach.

We feel that this gives an intuitive semantics to the 'recursive call' $X \stackrel{\text{df}}{=} X$ because we do not model procedure entry and procedure exit as separate (silent) actions. Thus, $X \stackrel{\text{df}}{=} X$ essentially means that X is continually 'rewritten' by itself and effects no behaviour at all, be it silent or observable. This is in contrast to the view that $X \stackrel{\text{df}}{=} X$ implies continual entry to a procedure and thus an infinite succession of silent moves; to model such behaviour, we would rather use $X \stackrel{\text{df}}{=} \emptyset; X$ (or even $X \stackrel{\text{df}}{=} \emptyset; X; \emptyset$, to model exit as well as entry, even though none of the transitions coming from the $\emptyset$ following X will be executable).

5.4 Equivalence notions

As in the case of net refinement, various kinds of structural equivalence are well behaved with respect to the solutions of a recursive equation on boxes. Below we consider two instances of the recursive equation (6), $X \stackrel{\text{df}}{=} \Omega(\mathbf{\Delta})$ and $X \stackrel{\text{df}}{=} \Omega'(\mathbf{\Delta}')$.

Theorem 12. *Let $\sim \in \{\, \mathsf{iso}\ ,\ \mathsf{iso_T}\ ,\ \mathsf{iso_S}\ ,\ \mathsf{iso_{ST}} \,\}$ and ψ be an isomorphism from Ω to Ω' such that $\Delta_v = X = \Delta'_{\psi(v)}$ or $\Delta_v \sim \Delta'_{\psi(v)}$, for every $v \in T_\Omega$. Then, for every solution Σ of the first equation, there is a solution Σ' of the second equation such that $\Sigma \sim \Sigma'$.* □

Theorem 13. *Let $\sim \in \{\, \mathsf{iso_S}\ ,\ \mathsf{iso_{ST}} \,\}$ and $\Omega\ \mathsf{iso_S}\ \Omega'$ and ψ be an isomorphism from Ω_0 to Ω'_0, where Ω_0 and Ω'_0 are obtained from respectively Ω and Ω' after replacing all places by their duplication equivalence classes. Moreover, let us assume that $\Delta_v = X = \Delta'_{\psi(v)}$ or $\Delta_v \sim \Delta'_{\psi(v)}$, for every $v \in T_\Omega$. Then, for every solution Σ of the first equation, there is a solution Σ' of the second equation such that $\Sigma \sim \Sigma'$.* □

5.5 Solving systems of recursive equations

The definitions presented, and the results obtained, for a single recursive equation are easily lifted to the general case, using standard techniques on Cartesian product spaces (for a finite set of equations) or on function spaces (for an infinite set of equations). The reader is referred to [8] for details.

6 S-Invariant Analysis

This section examines the extent to which S-invariant analysis – an indigenous Petri net technique – could be used within the compositional framework described in the two previous sections. We first define S-invariants and point out that they are related to safeness and other behavioural net properties. Then we consider how to construct an S-invariant of a refined net from S-invariants of the operator and operand boxes. Afterwards, we briefly discuss systems of recursive equations on nets. As an application of the S-invariant analysis, we explain how finite precedence properties may eradicate some problems related to infinite behaviours.

6.1 S-invariants and S-aggregates

In the literature [21, 46, 54], an S-invariant is defined as a weighting of the places of a (finite) Petri net satisfying certain consistency properties. Here, since we allow infinite nets, care is needed to provide meaningful operations on weights over possibly infinite sets of places. An *S-invariant* of a labelled net Σ is a mapping $\theta : S_\Sigma \cup T_\Sigma \to \Re^+$ such that, for every $t \in T_\Sigma$,[17]

$$\sum_{s \in S_\Sigma} W_\Sigma(s,t) \cdot \theta(s) \;=\; \theta(t) \;=\; \sum_{s \in S_\Sigma} W_\Sigma(t,s) \cdot \theta(s) \;.$$

A node z of Σ is *covered* by θ if $\theta(z) > 0$, and we say that Σ is *S-covered* (*T-covered*) by S-invariants of a certain kind, if for each place (transition) there is an S-invariant of that kind covering it. The S-invariant θ may be extended to real and complex markings of Σ, in the following way:

$$\theta(M) \;=\; \sum_{s \in S_\Sigma} M(s) \cdot \theta(s) \qquad \text{and} \qquad \theta(M,Q) \;=\; \sum_{s \in S_\Sigma} M(s) \cdot \theta(s) \;+\; \sum_{t \in T_\Sigma} Q(t) \cdot \theta(t) \;.$$

In this section, we consider labelled nets Σ with both real and complex markings, and call $\theta(M_\Sigma)$ the *weight* of θ. Figure 18 shows three different S-invariants of the labelled net $\Sigma_{\mathbf{bLs}}$; we shall refer to them as the 'bold', 'large' and 'small' S-invariants.

The defining condition for an S-invariant simply means that the number of tokens, weighted by θ, absorbed by any transition is finite and the same as that produced by it, i.e., if $\mathcal{M}$ is a real or complex marking and $\mathcal{M}'$ is reachable from $\mathcal{M}$, then $\theta(\mathcal{M}) = \theta(\mathcal{M}')$. This property is the cornerstone of the invariant analysis in the Petri net theory.

An S-invariant θ of an ex-restricted labelled net Σ will be called *conservative* if $\theta(^\circ\Sigma) = \theta(\Sigma^\circ) < \infty$. The interest in such a condition derives from the observation that a conservative

[17] In this section, we use generalised sums and products, defined in the standard way, for possibly infinite sets of non-negative real numbers.

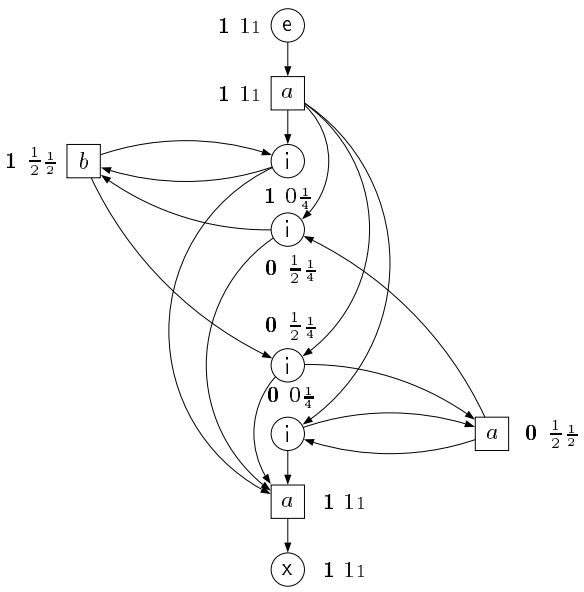


Fig. 18. Three S-invariants of a labelled net, Σ_{bLs}.

θ is an S-invariant of the net Σ_{sr} with $\theta(\mathsf{redo}) = \theta(\mathsf{skip}) = \theta(^\circ\Sigma)$. Notice that only if the entry and exit markings have the same weight (not necessarily finite), it may be possible for Σ to evolve from the entry to the exit marking.

An S-invariant θ is *1-conservative* if $\theta(^\circ\Sigma) = \theta(\Sigma^\circ) = 1$. Notice that the three S-invariants in figure 18 are all 1-conservative. In general, we will also require that $\theta(M_\Sigma) = \theta(^\circ\Sigma)$, so that it may be possible that M_Σ is reachable from the entry marking of Σ.

S-invariants are directly related to various behavioural properties of labelled nets.

Theorem 14. *Let Σ be a labelled net.*

(1) If for every place s there is an S-invariant θ of weight n such that $n \geq \theta(s) > \frac{n}{2}$, then Σ is safe and exhibits no auto-concurrency.
(2) If for every transition t there is an S-invariant θ of weight n such that $n \geq \theta(t) > \frac{n}{2}$, then Σ exhibits no auto-concurrency.
(3) If Σ is S-covered by 1-conservative S-invariants of weight 1, then Σ is clean. □

An *S-aggregate* of a labelled net Σ is a 1-conservative S-invariant θ with natural weights, $\theta(S_\Sigma \cup T_\Sigma) \subseteq \mathbf{N}$. Then exactly one entry (exit) place has the weight 1, all the remaining entry (exit) places having the zero weight. And, if the marking of Σ has the weight 1, then exactly one marked place or transition (if the marking is complex) has the weight 1, all the remaining ones having the zero weight.

Theorem 15. *Let Σ be a labelled net.*

(1) If Σ is S-covered by S-aggregates of weight 1, then it is safe, clean, and exhibits no auto-concurrency.
(2) If Σ is T-covered by S-aggregates of weight 1, then it exhibits no auto-concurrency.
(3) If Σ is a plain box with the empty marking which is S-covered by S-aggregates, then Σ is a static box.
(4) If Σ is a plain box with a non-empty real marking which is S-covered by S-aggregates of weight 1, then Σ is a dynamic box. □

6.2 The synthesis problem for refinement

When studying the compositionality properties, it seems natural to ask whether is it possible to construct S-invariants for a refined net from those defined for the operator box and the plain box operands. However, the example in figure 19 shows that it does not seem reasonable to insist on constructing *all* S-invariants. The operator box there, Ω, has a single 1-conservative S-invariant, and each 1-conservative S-invariant of Σ is characterised by a real number r $(0 \leq r \leq 1)$ and the weight distribution shown in figure 19. Now, in the refined net $\Omega(\Sigma)$, each 1-conservative S-invariant is characterised by two real numbers, p and q $(0 \leq p, q \leq 1)$, and the weight distribution shown in figure 19. Hence, unlike in Σ, there is no link between the distribution of weights for the exit and entry places. But, if the relabelling of Ω was the identity relabelling, ϱ_{id}, then the refinement would be isomorphic to Σ and thus have the same S-invariants. This is not compatible with a purely structural construction of all S-invariants of a refined net based solely on the connectivity in the nets involved in the refinement, without taking into account the (possibly complex) properties of transition labellings. Having said that, it is possible to devise a general scheme for constructing a sufficiently rich set of S-invariants for nets resulting from net refinement.

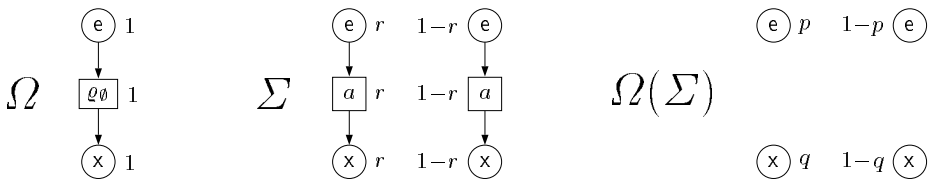


Fig. 19. S-invariant explosion phenomenon (where $\varrho_\emptyset = \emptyset$).

Until the end of section 6.2, we shall consider a fixed refinement $\Omega(\mathbf{\Sigma})$, relying on the notation used in section 4. In particular, we shall assume that Ω is an operator box with a complex marking (M, Q), and each Σ_v in $\mathbf{\Sigma}$ is a plain box with a real marking which is non-empty if and only if $v \in Q$. Moreover, we shall assume that θ is an S-invariant of Ω, and θ_v is an S-invariant of Σ_v, for every $v \in T_\theta$, where $T_\theta \subseteq T_\Omega$ and $S_\theta \subseteq S_\Omega$ are the nodes of Ω covered by θ. For every place s in Ω, we will use the following notation. If $v \in {}^\bullet s$ and $q \in \Sigma_v^\circ$ (or, symmetrically, if $v \in s^\bullet$ and $q \in {}^\circ\Sigma_v$) then

$$\mathsf{SP}^s_{\mathsf{new}}[vq] \;=\; \{p \;\in\; \mathsf{SP}^s_{\mathsf{new}} \mid v \lhd q \in \mathsf{subtrees}(p)\}$$

where $\mathsf{subtrees}(p) = \{v \lhd q \mid p = s \lhd (\{v \lhd q\} + \mathcal{S})\}$. That is, $\mathsf{SP}^s_{\mathsf{new}}[vq]$ is the set of all place trees p in $\mathsf{SP}^s_{\mathsf{new}}$ which have $v \lhd q$ appended to the root. It follows from the definition of net refinement that if $v \in {}^\bullet s$ and $w \in s^\bullet$ then

$$\mathsf{SP}^s_{\mathsf{new}} \;=\; \bigcup_{q \in \Sigma^\circ_v} \mathsf{SP}^s_{\mathsf{new}}[vq] \;=\; \bigcup_{q \in {}^\circ\Sigma_w} \mathsf{SP}^s_{\mathsf{new}}[wq]$$

and that both unions are in fact disjoint unions.

Composing S-invariants There may be many different ways in which one could meaningfully combine the S-invariants defined for Ω and $\mathbf{\Sigma}_{T_\theta}$ into a valid S-invariant of $\Omega(\mathbf{\Sigma})$. The aim here is to describe a general scheme of achieving this.

A *distribution function* is a mapping $\partial_s : \mathsf{SP}^s_{\mathsf{new}} \to \Re^+$ defined for a place s in S_θ. If s is an isolated place, then $\mathsf{SP}^s_{\mathsf{new}} = \{s\}$ and $\partial_s(s) = 1$. If s is not isolated then we require that, for all $v \in {}^\bullet s$ and $q \in \Sigma^\circ_v$ as well as for all $v \in s^\bullet$ and $q \in {}^\circ\Sigma_v$,

$$\theta_v(q) \;=\; \sum_{p \,\in\, \mathsf{SP}^s_{\mathsf{new}}[vq]} \partial_s(p)\ .$$

The above condition is well formed, as θ_v exists due to ${}^\bullet S_\theta = S_\theta^\bullet = T_\theta$. The name of ∂_s derives from the observation that it distributes the weight of 1 over all places in $\mathsf{SP}^s_{\mathsf{new}}$, i.e., $\sum_{p \,\in\, \mathsf{SP}^s_{\mathsf{new}}} \partial_s(p) = 1$.

Given a set of distribution functions for the places in S_θ, $\partial = \{\partial_s\}_{s \in S_\theta}$, we define a mapping $\theta_\partial : S_{\Omega(\mathbf{\Sigma})} \cup T_{\Omega(\mathbf{\Sigma})} \to \Re^+$, in such a way that for every node z in $\Omega(\mathbf{\Sigma})$,

$$\theta_\partial(z) \;=\; \begin{cases} \theta(v) \cdot \sum_{t \in U} U(t) \cdot \theta_v(t) & \text{if} \quad z = (v,\alpha) \lhd U \in \mathsf{T}^v_{\mathsf{new}} \quad \text{and} \quad v \in T_\theta \\ \theta(v) \cdot \theta_v(i) & \text{if} \quad z = v \lhd i \in \mathsf{ST}^v_{\mathsf{new}} \quad \text{and} \quad v \in T_\theta \\ \theta(s) \cdot \partial_s(z) & \text{if} \quad z \in \mathsf{SP}^s_{\mathsf{new}} \quad \text{and} \quad s \in S_\theta \\ 0 & \text{otherwise}\ . \end{cases}$$

Theorem 16. *The mapping θ_∂ is an S-invariant of $\Omega(\mathbf{\Sigma})$. It is 1-conservative if so is θ. Moreover, if the weight of each θ_v is $Q(v)$, then the weight of θ_∂ is that of θ.* □

Multiplicative distribution functions The above definition of an S-invariant of $\Omega(\mathbf{\Sigma})$ is not fully constructive without saying how to find the distribution functions ∂_s.[18] A possible scheme for constructing distribution functions can be based on the following definition. For a non-isolated place s in S_θ, let $\partial^{\mathsf{mult}}_s$ be a mapping such that, for every $p \in \mathsf{SP}^s_{\mathsf{new}}$,

$$\partial^{\mathsf{mult}}_s(p) \quad = \prod_{v \lhd q \in \mathsf{subtrees}(p)} \theta_v(q)$$

provided that the generalised product does exist. Then, in all those cases that $\partial^{\mathsf{mult}}_s$ is a valid distribution function,[19] we will call it the *multiplicative* distribution function for s.

[18] It may happen that such functions do not exist due to a particular choice of the S-invariants of Ω and $\mathbf{\Sigma}_{T_\theta}$ [8].

[19] Even a real-valued $\partial^{\mathsf{mult}}_s$ may fail to be a distribution function [8].

We will now present two specific cases where $\partial_s^{\mathsf{mult}}$ is guaranteed to be a distribution function.

Proposition 4. *Let $s \in S_\theta$ be a non-isolated place which is* degree-finite*, which means that both ${}^\bullet s$ and $s^\bullet$ are finite sets. Then $\partial_s^{\mathsf{mult}}$ is a distribution function.* □

The bold and small S-invariants in figure 18 can both be derived through multiplicative distribution functions. To see this, observe that the net there can be obtained as the result of the refinement using the nets in the figure 20. The chosen S-invariants are the constant 1-valued functions for θ and θ_{v_1} and θ_{v_3}, and the two (bold and small) S-invariants indicated in figure 20 for θ_{v_2}.

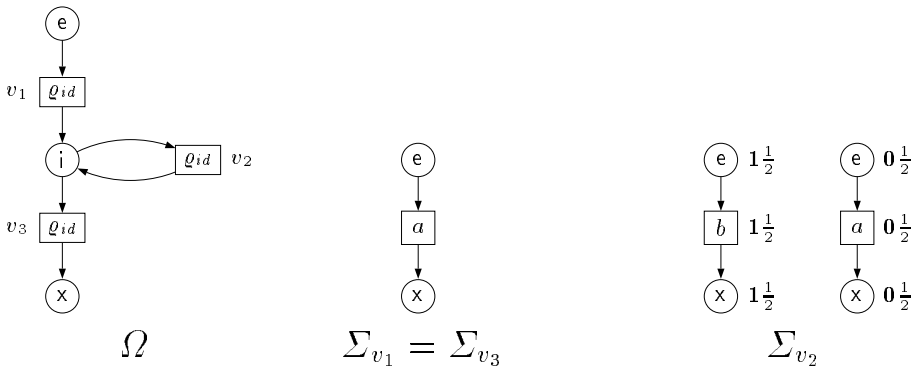


Fig. 20. Boxes and S-invariants for the exmaple in figure 18.

The particular interest in multiplicative distribution functions is due to the fact that in the translation of a typical process algebra into Petri nets, all operator boxes would have degree-finite places (and most of them are just finite), perhaps with the sole exception of the infinitary choice. Another interesting consequence of the last result is a characterisation of the S-coverings and T-coverings of $\Omega(\mathbf{\Sigma})$.

Proposition 5. *Let all the places in Ω be degree-finite. If Ω and $\mathbf{\Sigma}$ are all T-covered (S-covered), then so is $\Omega(\mathbf{\Sigma})$.* □

Next we partially lift the degree-finiteness condition required in proposition 4. An S-invariant θ of a labelled net Σ is *ex-binary* if $\theta({}^\circ\Sigma \cup \Sigma^\circ) \subseteq \{0, 1\}$. For a 1-conservative θ, in such a case there is exactly one entry place and one exit place with weight 1, all remaining entry and exit places having zero weight. Notice also that all S-aggregates are ex-binary.

Proposition 6. *Let s be a non-isolated place in S_θ such that θ_v is an ex-binary 1-conservative S-invariant, for every $v \in {}^\bullet s \cup s^\bullet$. Then $\partial_s^{\mathsf{mult}}$ is a distribution function.* □

We then combine propositions 4 and 6 into a single result.

Theorem 17. *Let s be a non-isolated place in S_θ such that all but finitely many of the S-invariants θ_v, for $v \in {}^\bullet s \cup s^\bullet$, are ex-binary. Then $\partial_s^{\mathsf{mult}}$ is a distribution function.* □

An important consequence of the existence of multiplicative distribution functions is that T-covering by S-aggregates is a stable property for refinements.

Proposition 7. *Let Ω be T-covered by S-aggregates of weight n, and Σ_v be T-covered by S-aggregates of weight $Q(v)$, for every $v \in T_\Omega$. Then $\Omega(\mathbf{\Sigma})$ is T-covered by S-aggregates of weight n.* □

A similar property for S-coverings may not hold. For example, the nets in figure 20 are all S-covered by S-aggregates while the result of their combination, shown in figure 18, is not S-covered by S-aggregates, since it is not safe from the entry marking. However, in the absence of side-loops in the operator box, we have the following.

Theorem 18. *Let Ω be a pure operator box.*

(1) If Ω is unmarked, S-covered by S-aggregates and the nets in $\mathbf{\Sigma}$ are unmarked and S-covered by S-aggregates, then $\Omega(\mathbf{\Sigma})$ is a static box, S-covered by S-aggregates.
(2) If Ω is marked, S-covered by S-aggregates of weight 1, the nets in $\mathbf{\Sigma}_{T_\Omega \setminus Q}$ are unmarked and S-covered by S-aggregates, and those in $\mathbf{\Sigma}_Q$ are S-covered by S-aggregates of weight 1, then $\Omega(\mathbf{\Sigma})$ is a dynamic box, S-covered by S-aggregates of weight 1. □

The last result has important consequences for the Petri net in algebras like that used to model PBC expressions.

Theorem 19. *Let $\mathcal{O}$ be a set of pure operator boxes, and $\mathcal{P}$ be a set of plain boxes.*

(1) If all the boxes in $\mathcal{O}$ and $\mathcal{P}$ are unmarked and S-covered by S-aggregates, then all boxes obtained through successive refinements from $\mathcal{O}$ and $\mathcal{P}$ are static plain boxes S-covered by S-aggregates.
(2) If all the boxes in $\mathcal{O}$ and $\mathcal{P}$ are marked and S-covered by S-aggregates of weight 1, then all boxes obtained through successive refinements from $\mathcal{O}$ and $\mathcal{P} \cup \lfloor \mathcal{P} \rfloor$, are dynamic plain boxes S-covered by S-aggregates of weight 1. □

Rational groupings The large S-invariant in figure 18 cannot be obtained using multiplicative distribution functions from any 1-conservative S-invariants of the nets shown in figure 20. For the only non-trivial 1-conservative S-invariants for Ω, Σ_{v_1} and Σ_{v_3}, have the constant weights 1, and each 1-conservative S-invariant for Σ_{v_2} is characterised by a real number r ($0 \leq r \leq 1$) such that r is the weight for the nodes on the left, and $1-r$ is the weight for the nodes on the right. Using the formula characterising multiplicative distribution functions, these would lead to the following weights for the places of the net in figure 18 (taken from top to bottom): 1, r^2, $r(1-r)$, $r(1-r)$, $(1-r)^2$ and 1. Thus no r could possibly yield: 1, 0, $\frac{1}{2}$, $\frac{1}{2}$, 0 and 1.

But multiplicative distribution functions are not the only distribution functions. Let us re-phrase the condition for being a distribution function for a place s in S_θ. Suppose that $s^\bullet = \{\ldots, v_i, \ldots\}$ and ${}^\bullet s = \{\ldots, w_j, \ldots\}$. Defining a distribution function ∂_s for s amounts to constructing a table:

weight	$\ldots$	v_i	$\ldots$	w_j	$\ldots$	
∂_1	$\ldots$	$e_{i,1}$	$\ldots$	$x_{j,1}$	$\ldots$	row corresponding to $s \lhd \{\ldots, v_i \lhd e_{i,1}, \ldots, w_j \lhd x_{j,1}, \ldots\}$
∂_2	$\ldots$	$e_{i,2}$	$\ldots$	$x_{j,2}$	$\ldots$	row corresponding to $s \lhd \{\ldots, v_i \lhd e_{i,2}, \ldots, w_j \lhd x_{j,2}, \ldots\}$
$\vdots$		$\vdots$		$\vdots$		$\vdots$
$\underbrace{\quad}_{\partial_s}$						$\underbrace{\quad}_{\mathsf{SP}^s_{\mathsf{new}} \setminus \partial_s^{-1}(0)}$

where each ∂_k is a positive real; each $e_{i,k}$ belongs to ${}^\circ\Sigma_{v_i} \setminus \theta_{v_i}^{-1}(0)$; each $x_{j,k}$ belongs to $\Sigma^\circ_{w_j} \setminus \theta_{w_j}^{-1}(0)$; each place in ${}^\circ\Sigma_{v_i} \setminus \theta_{v_i}^{-1}(0)$ and $\Sigma^\circ_{w_j} \setminus \theta_{w_j}^{-1}(0)$ must appear at least once in the column corresponding to v_i and w_j, respectively. Moreover, the following hold, for all i, j and k:

$$\theta_{v_i}(e_{i,k}) \;=\; \sum_{\substack{l \text{ such that} \\ e_{i,l} = e_{i,k}}} \partial_l \qquad \text{and} \qquad \theta_{w_j}(x_{j,k}) \;=\; \sum_{\substack{l \text{ such that} \\ x_{j,l} = x_{j,k}}} \partial_l \;.$$

The table may be constructed if, for example, the distribution of non-zero weights in the entry and exit places of the nets contributing to $\mathsf{SP}^s_{\mathsf{new}}$ is the same. More precisely, assume that there is a finite or infinite set $K = \{1, 2, 3, \ldots\} \subseteq \mathbf{N}$ and a set $\{\partial_k \mid k \in K\}$ of positive reals such that, for every $v_i \in s^\bullet$, the entry places covered by θ_{v_i} can be enumerated as $\{e_{i,k}\}_{k \in K}$ and $\theta_{v_i}(e_{i,k}) = \partial_k$, for every $k \in K$; and, similarly, for every $w_j \in {}^\bullet s$, the exit places covered by θ_{w_j} can be enumerated as $\{x_{j,k}\}_{k \in K}$ and $\theta_{w_j}(x_{j,k}) = \partial_k$, for every $k \in K$. If we use these ∂_k's, $e_{i,k}$'s and $x_{j,k}$'s to construct the table, then each $e_{i,k}$ or $x_{j,k}$ occurs exactly once in the corresponding column, and the resulting table yields a distribution function for s.

Similar effect may be achieved even though the distributions of weights are not identical for all v_i's and w_j's. In particular, if all weights for the places in $(\bigcup_{v \in s^\bullet} {}^\circ\Sigma_v) \cup (\bigcup_{w \in {}^\bullet s} \Sigma^\circ_w)$ belong to $\mathbf{N}/n$, i.e., are rational numbers with the same base n, we can take $\partial_1 = \cdots = \partial_n = \frac{1}{n}$, and choose $\{e_{i,k}\}_{k=1,\ldots,n}$ and $\{x_{j,k}\}_{k=1,\ldots,n}$ under the assumption that repetitions are allowed. That is, one can construct a table of the following form where we allow repetitions of rows:

weight	$\ldots$	v_i	$\ldots$	w_j	$\ldots$	
$\frac{1}{n}$	$\ldots$	$e_{i,1}$	$\ldots$	$x_{j,1}$	$\ldots$	row corresponding to $s \lhd \{\ldots, v_i \lhd e_{i,1}, \ldots, w_j \lhd x_{j,1}, \ldots\}$
$\frac{1}{n}$	$\ldots$	$e_{i,2}$	$\ldots$	$x_{j,2}$	$\ldots$	row corresponding to $s \lhd \{\ldots, v_i \lhd e_{i,2}, \ldots, w_j \lhd x_{j,2}, \ldots\}$
$\vdots$		$\vdots$		$\vdots$		$\vdots$
$\frac{1}{n}$	$\ldots$	$e_{i,n}$	$\ldots$	$x_{j,n}$	$\ldots$	row corresponding to $s \lhd \{\ldots, v_i \lhd e_{i,n}, \ldots, w_j \lhd x_{j,n}, \ldots\}$
$\underbrace{\quad}_{\partial_s}$						$\underbrace{\quad}_{\mathsf{SP}^s_{\mathsf{new}} \setminus \partial_s^{-1}(0)}$

If some $e_{i,h}$ or $x_{j,h}$ has the weight $\frac{m}{n}$ in the net it originates from, it will occur exactly m times in the corresponding column; and, if a row occurs m times in the table, the corresponding place will have the weight $\frac{m}{n}$ in ∂_s. Such a construction is always possible, and generally in many different ways since we may reorder independently each column.

The last case has an interesting consequence. Let us call a 1-conservative S-invariant θ of box Σ a *semi-aggregate* if $\theta(S_\Sigma \cup T_\Sigma) \in \mathbf{N}/2$. Then, a box with the empty marking

is *semi-static* if it is T-covered by S-aggregates, its entry and exit places are covered by S-aggregates, and its internal places are covered by semi-aggregates. Similarly, a box with a non-empty marking is *semi-dynamic* if it is T-covered by S-aggregates of weight 1, its entry and exit places are covered by S-aggregates of weight 1, and its internal places are covered by semi-aggregates of weight 1.

A semi-dynamic box, as well as a semi-static box, when started from ${}^\circ\Sigma$ or M_Σ is: clean, exhibits no auto-concurrency, its entry and exit places are safe, and its internal places are 2-bounded although not necessarily safe. In a semi-static or semi-dynamic box, each pair of entry/exit places is also covered by semi-aggregates of weight 1; one simply can form the average between an S-aggregate covering the entry place and an S-aggregate covering the exit place. Finally, a static box S-covered by S-aggregates is clearly semi-static as well; a dynamic box S-covered by S-aggregates of weight 1 is clearly semi-dynamic; and the tokenless version of a semi-dynamic box is semi-static.

Proposition 8. *Let no entry or exit place in Ω be part of a side-loop.*

(1) If Ω is an unmarked operator box S-covered by S-aggregates and, for every $v \in T_\Omega$, Σ_v is a semi-static plain box, then $\Omega(\mathbf{\Sigma})$ is a semi-static plain box.
(2) If Ω is a marked operator box S-covered by S-aggregates of weight 1 and, for every $v \in T_\Omega$, Σ_v is a semi-dynamic plain box if $v \in Q$, and a semi-static plain box otherwise, then $\Omega(\mathbf{\Sigma})$ is a semi-dynamic plain box. □

And then we can generalise theorem 19.

Theorem 20. *Let $\mathcal{O}$ be a set of operator boxes such that no entry or exit place is part of a side-loop, and $\mathcal{P}$ be a set of plain boxes.*

(1) If all the boxes in $\mathcal{O}$ are unmarked and S-covered by S-aggregates, and all the boxes in $\mathcal{P}$ are semi-static, then all boxes obtained through successive refinements from $\mathcal{O}$ and $\mathcal{P}$ are semi-static plain boxes.
(2) If all the boxes in $\mathcal{O}$ are marked and S-covered by S-aggregates of weight 1, and all the boxes in $\mathcal{P}$ are semi-dynamic, then all boxes obtained through successive refinements from $\mathcal{O}$ and $\mathcal{P} \cup \lfloor \mathcal{P} \rfloor$, are semi-dynamic plain boxes. □

6.3 The synthesis problem for recursive systems

We now briefly mention the problem of constructing S-invariants of the maximal solution $\mathbf{\Sigma}$ of the system of recursive equations (5) introduced in section 5.1. Again, we shall only consider the unmarked case.

As noted in section 5.2, non-maximal solutions of the recursive system (5) may be non-clean if infinite steps are allowed in the evolutions, which is related to there being no covering by S-invariants (in figure 17, the isolated places in $\Sigma^{min:max}$ and $\Sigma^{max:min}$ are not covered by any 1-conservative S-invariant). We shall therefore concentrate on the maximal solution.

In the treatment of the synthesis problem for (5), one aims at constructing S-invariants for all boxes in the maximal solution $\mathbf{\Sigma}$. It turns out that the techniques developed for dealing with net refinement can be lifted to the level of recursive systems [8]. For example, we obtain the following counterpart of proposition 4.

Theorem 21. *Let (5) be such that all the place trees in its maximal solution* $\mathbf{\Sigma}$ *are finite. If all the boxes in (5) are (S-covered) T-covered by 1-conservative S-invariants, then all the boxes* $\mathbf{\Sigma}$ *are (S-covered) T-covered by 1-conservative S-invariants.* □

Most importantly, the results of sections 6.2 and 6.3 can be extended to the current framework.

Theorem 22. *Let* $\mathcal{O}$ *be a set of pure unmarked operator boxes and* $\mathcal{P}$ *be a set of unmarked ex-directed plain boxes, all S-covered by S-aggregates. Then all boxes obtained through successive refinements or (maximal) solutions of recursive systems from* $\mathcal{O}$ *and* $\mathcal{P}$ *are static plain boxes S-covered by S-aggregates.* □

Theorem 23. *Let* $\mathcal{O}$ *be a set of unmarked ex-directed operator boxes S-covered by S-aggregates, and* $\mathcal{P}$ *be a set of semi-static plain boxes. Then all boxes obtained through successive refinements or (maximal) solutions of recursive systems from* $\mathcal{O}$ *and* $\mathcal{P}$ *are semi-static plain boxes.* □

To summarise, we have described very general tools to construct S-invariants for nets constructed through operator boxes, from S-invariants of the composed nets. These can be used to obtain coverability results, and thus to derive important behavioural characteristics, of whole families of nets. This method could be contrasted with considering separately each relevant behavioural property (such as safeness, cleanness, lack of auto-concurrency) and proving its validity by structural induction; here all these follow directly from the coverability properties.

6.4 Partial orders and finite precedence property

Recursion generally leads to infinite nets. And, despite having well-formed systematic structure, they may still exhibit some of the problems characteristic of infinite Petri nets. The semantics of infinite nets has been thoroughly examined in [6], where it has been observed that various non-equivalent 'natural' extensions of the concurrent evolution rule defined for finite nets may be justifiably considered. Take, for example, the minimal net model of the expression $E = X;b$ where X is defined by the recursive equation $X \stackrel{\text{df}}{=} a\|X$, shown in figure 21 (see also figure 17).

Consider the transition t which, if we start from the entry marking, needs the completion of the infinitely many t_i transitions ($i = 0, 1, 2, \ldots$) to occur. If the concurrent semantics of infinite nets was defined through finite steps, t would be inaccessible. But, as noticed in [6], it is possible to allow infinite steps, by executing simultaneously infinitely many concurrent transitions. This would allow evolutions of the net which are not equivalent to any interleaving sequences of transitions, unless one is prepared to allow transfinite evolutions, nor can be obtained as 'limits' of some infinite evolutions. For instance, if the infinite step rule was adopted, t would be allowed to occur after the infinite step $\{t_0, t_1, t_2, \ldots\}$. Thus the range of possible evolutions depends on the choice of the basic occurrence rule.

By contrast, the picture changes if we consider the maximal box model for the expression E, shown in figure 21 (see also figure 17). Here, it is the places e_∞ and i_∞ which deserve our particular attention. Apart from arising naturally in the search for unique solutions of recursive systems of equations on nets (see section 5.3), they also have another interesting

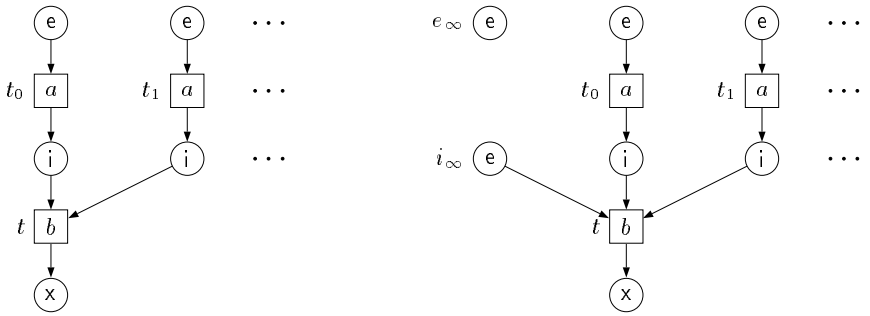


Fig. 21. The minimal and maximal box models of $E = X;b$ where $X \stackrel{\text{df}}{=} a \| X$.

property: with e_∞ and i_∞, even after concurrently executing all t_i's in a single step, t will still be missing the token in i_∞. In fact, this is an instance of a general property of the maximal solutions of recursive systems. As a consequence, using the maximal solution, the possible evolutions will not fundamentally rely on allowing or disallowing infinite steps, and infinite steps may always be regarded as limits of successive finite steps or even individual transition occurrences. The interleaving semantics will not lose any relevant evolutions, and the previous results based on the S-invariant analysis may be applied in full generality, i.e., also for the semantics based on infinite steps.

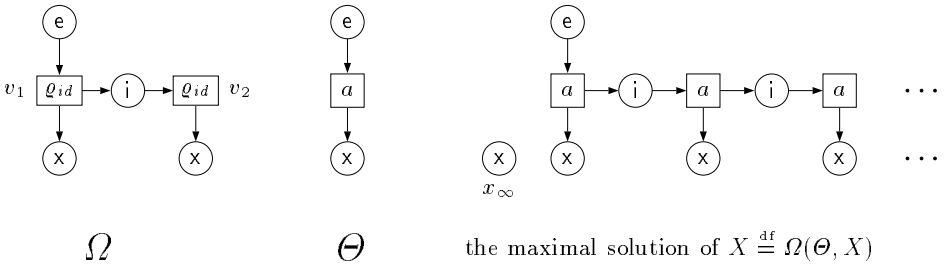


Fig. 22. Terminated evolution and finite precedence.

Another observation of the same nature concerns the terminated evolutions, i.e., those after which all the exit places are concurrently marked. Consider, for instance, the nets in figure 22. The recursive equation $X \stackrel{\text{df}}{=} \Omega(\Theta, X)$ has two infinite net solutions, the minimal and maximal one, Σ^{min} and Σ^{max}. The latter is shown in figure 22, and Σ^{min} can be obtained from Σ^{max} by deleting the isolated exit place x_∞. Both solutions are covered by S-aggregates, which is also true of the operator box Ω and the operand box Θ. Let us first consider the minimal solution, Σ^{min}. It has a unique infinite evolution which may not be reduced to a finite succession of steps as the net is sequential. The corresponding 'limit' marking has one token in each of the exit places, so that the exit marking may be

considered as reachable at a limit after infinitely many sequential transition occurrences. This has a worrying consequence. Indeed, suppose that a box B is obtained by arranging two component boxes B_1 and B_2 in sequence; one would expect that the valid finite or infinite evolutions of B (from its entry marking) may be obtained by taking an evolution of B_1, or by putting in sequence a complete evolution of B_1 and an evolution of B_2. But if the complete evolution of B_1 may involve an infinite sequence of actions, the result will not always be a valid evolution of B [15].

Again, the situation is radically different for the maximal solution, Σ^{max}. With the isolated exit place x_∞, while the limit marking corresponding to the same as above unique infinite sequential evolution only contains exit places, the evolution is not complete since there will never be any token in the isolated place x_∞. And, again, this is a general property, i.e., infinite evolutions may never be complete.

The two properties we have just discussed point at a rather special role of the maximal solutions of recursive systems; not only are they the only way to define the solutions uniquely, but also exhibit crucial properties of finite precedence of infinite (sequential or concurrent) evolutions.

We are now ready to formulate the already announced result about the finite precedence of events. We shall do this in the process framework since it allows one to directly delineate the causality, i.e., the precedence characteristics of any event (transition occurrence). We adopt the most liberal definition of finite accessibility, the third finite accessibility (TFA), since one of our aims is to show that even being so liberal is of no consequence here. Moreover, we only treat safe nets and assume that all nets we start from are finite (so that they themselves respect the finite precedence property).

Theorem 24. *Let Σ be a plain box obtained, through refinements and maximal solutions of recursive systems, from a set of finite pure unmarked ex-directed operator boxes and a set of finite unmarked ex-directed plain boxes, all covered by S-aggregates. Then every event of every process of $\overline{\Sigma}$ only has finitely many predecessor events.* □

Thus in the context of the theory developed here, the two finite accessibility constraints (TFA) and (SFA) are equivalent. Moreover, allowing infinite steps does not fundamentally change the semantics with respect to that based on finite steps. What may happen though is that some configurations of tokens are finitely reachable instead of being reachable 'at the limit'. And, as the example in figure 21 shows, theorem 24 crucially depends on using the maximal, rather than any other, solutions of recursive equations.

The case of complete processes, i.e., by definition, those π for which $\mathsf{Min}(\pi)$ and $\mathsf{Max}(\pi)$ correspond, respectively, to the entry to exit markings of Σ, is treated next.

Theorem 25. *Let Σ be a plain box obtained, through refinements and maximal solutions of recursive systems, from a set of finite pure unmarked ex-directed operator boxes and a set of finite unmarked ex-directed plain boxes, all covered by S-aggregates. Then every process of $\overline{\Sigma}$ with conditions corresponding to all exit places of Σ, has finitely many events.* □

The proofs of the last two theorems are based on the results formulated earlier in this section, providing a non-trivial application of the S-invariant analysis of compositional boxes.

6.5 Equivalence notions

If Σ and Θ are isomorphic labelled nets, then they have up to isomorphism the same S-invariants. If two transitions t and t' in a net duplicate each other (more generally, if they have the same connectivity as their labels are not important here), then for every S-invariant θ, it follows that $\theta(t) = \theta(t')$. Hence, if two unmarked nets are iso_T-equivalent, then they have the same S-invariants, up to isomorphism and transition duplication. The same is true for labelled nets marked with real markings. The case of the iso_S-equivalence is more complicated. For consider a set $S' \subseteq S$ of duplicate places and an S-invariant θ. It is then possible to freely re-distribute the weights inside S' without destroying the S-invariance and 1-conservativeness properties, provided that the total weight of S' and the weight of the marking on S' remain unchanged. Hence two iso_S-equivalent nets will have the same S-invariants up to isomorphism and re-distribution of weights inside each duplication equivalence class. The case of the iso_{ST}-equivalence combines the discussions of both iso_T-equivalence and iso_S-equivalence.

7 An Algebra of Nets

In this and the next section, we present a generic framework for defining process algebras with two consistent semantics, based respectively on Petri nets and SOS rules. This section deals with the first of the two algebras.

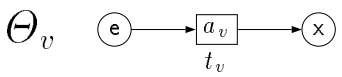
7.1 sos–operator boxes

The first aim is to delineate a class of operator boxes that could be applied to static and dynamic boxes in such a way that a counterpart of the process algebra SOS rule could also be formulated in the domain of nets. We will define operator boxes satisfying this property, called sos-operator boxes, by imposing two conditions on a generic sos-operator box $\Omega = (S, T, W, \lambda, \emptyset)$. To simplify the argument, we initially assume that only the identity relabelling ϱ_{id} is used to annotate the transitions in Ω. Another point arises from the observation that here a concrete net operator Ω aims at modelling a corresponding syntactic operator op_Ω on process expressions, to be used in prefix, postfix, infix or functional mode. But, whereas Ω might in principle be marked, we have no means to specify explicitly the marking of a symbol representing a net operator, as it is a mere lexical construct or symbol. From the static or dynamic nature of the various operands it is, however, possible to associate a marking to an expression, but such a marking will be a fiction of imagination. Hence Ω and, indeed, all operator boxes considered in this section will have markings which will simply be left implicit.

The two conditions defining an sos-operator box, (OpNet1) and (OpNet2) below, will be derived by applying Ω to suitably chosen tuples of static boxes $\mathbf{\Theta}$, and then considering some properties that the composition $\Omega(\mathbf{\Theta})$ ought to satisfy. Our first requirement will be that:

> Operations on static boxes should not lead outside
> the semantic domain of static boxes. (Req1)

To analyse its consequences, consider an Ω-tuple of very simple[20] static boxes $\mathbf{\Theta}$ defined thus, for every $v \in T$:



where $a_v \neq a_w$ for $v \neq w$. An easy observation is that, due to the simple form of the refining nets, $\Omega(\mathbf{\Theta})$ and Ω are isomorphic up to transition labelling. In other words, we can think of the net $\Omega(\mathbf{\Theta})$ as though it were Ω with each transition v being labelled by α_v. And so (Req1) implies that $\Omega(\mathbf{\Theta})$, and thus Ω itself, should be a static box[21], i.e., all markings reachable from ${}^\circ\Omega$ or Ω° should be safe and clean:

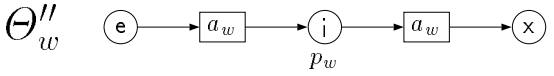
Ω is a static box. (OpNet1)

In the light of the discussion carried out in the previous sections, such a condition should not be too surprising. What is worth noting, however, is that it has been derived using a very general argument without, for instance, relying on the covering of Ω by S-invariants.

(Req2) is directly motivated by our wish to obtain an SOS rule for compositionally defined nets:

The net obtained as a result of an evolution of a net composed from other nets, should itself be a composition of nets related to the original ones. (Req2)

We interpret this as saying that no matter how $\overline{\Omega(\mathbf{\Theta})}$ (or $\Omega(\mathbf{\Theta})$) evolves, the resulting net Σ should be derivable as a composition $\Sigma = \Omega(\mathbf{\Theta}')$ where $\lfloor \overline{\mathbf{\Theta}'} \rfloor = \lfloor \mathbf{\Theta} \rfloor$. It turns out that this leads to a perhaps unexpected property of the complex markings reachable from the entry or exit marking of Ω. For suppose that Ω is an operator box satisfying (OpNet1) and (M, Q) is an arbitrary complex marking reachable from the entry marking ${}^\circ\Omega$ (the case where (M, Q) is reachable from the exit marking is symmetric). Then, from the safeness of Ω, it follows that M and Q are sets and $M \cap ({}^\bullet Q \cup Q^\bullet) = \emptyset$. Consider now another Ω-tuple $\mathbf{\Theta}''$ of simple static boxes obtained from the previously used $\mathbf{\Theta}$ by replacing, for every $w \in Q$, the Θ_w by Θ''_w defined thus:



$\Omega(\mathbf{\Theta})''$ can now be thought of as Ω with every transition $v \in T \backslash Q$ being labelled by a_v, and every transition $w \in Q$ being split into the 'beginning of w' and 'end of w', both transitions being labelled by a_w and linked by a single place p_w.

[20] Actually, we implicitly assumed that one should always be able to use these boxes, as well as those used later in this section to motivate (OpNet2), as refining nets.

[21] Strictly speaking, a static box is, by definition, plain. Here we extend this notion, which is purely marking-based, to operator boxes as well.

It is easy to see that $\overline{\Omega(\mathbf{\Theta})''}$ is a safe net which can evolve into the net Σ whose (safe) marking is $M \cup \{p_w \mid w \in Q\}$. Since we should be able to represent Σ as $\Omega(\mathbf{\Theta}')$ where $\lfloor \mathbf{\Theta}' \rfloor = \lfloor \mathbf{\Theta}'' \rfloor$, it follows from the definition of net refinement and safeness of Σ, that for every $v \in T \backslash Q$, Θ'_v is either Θ_v or $\overline{\Theta_v}$ or $\underline{\Theta_v}$, and for every $w \in Q$, Θ'_w is Θ''_w with exactly one token inside the place p_w. And, crucially, M is the disjoint union of the pre-sets of transitions v such that $\Theta'_v = \overline{\Theta_v}$ and the post-sets of transitions v such that $\Theta'_v = \underline{\Theta_v}$. In other words, the 'real' part of the complex marking (M, Q) can be factored onto pre- and post-sets of some transitions in T.

Following this observation, we shall say that a pair $\mu = (\mu_e, \mu_x)$ of sets of transitions of Ω is a *factorisation* of a set of places $M \subseteq S$ if

$$M \;=\; \biguplus_{v \in \mu_e} {}^\bullet v \;\uplus\; \biguplus_{v \in \mu_x} v^\bullet \;.$$

Moreover, a quadruple of sets of transitions of Ω, $\mu = (\mu_e, \mu_d, \mu_x, \mu_s)$, will be called a *factorisation* of a complex safe marking (M, Q) of Ω if $\mu_d = Q$, $\mu_s = T \backslash (\mu_e \cup \mu_d \cup \mu_x)$, and (μ_e, μ_x) is a factorisation of M. Then Ω itself will be called *factorisable* if for every safe complex marking (M, Q) reachable from the entry or exit marking of Ω, there is at least one factorisation. And a second condition to be imposed on an sos-operator box is that

Ω is factorisable. (OpNet2)

Not all static operator boxes are factorisable, as illustrated by the right box in figure 23. For in the marking obtained from the entry marking after the occurrences of v_1 and v_3, the left token can be factored out using v_3 and the right token can be factored out using v_2, but the middle token is 'orphaned'. Notice, however, that this non-factorisable static operator box is S-covered by S-aggregates. Factorisability of an operator is behavioural in nature since it is necessary to check whether every reachable marking can be factored. Fortunately, most operators used in practice are small, or have a straightforward symmetric structure, so that this is not normally a real problem.

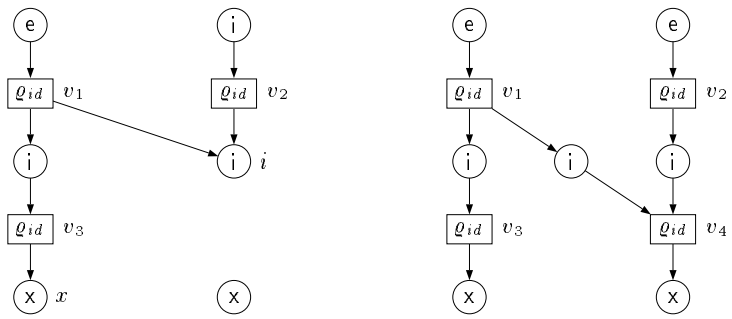


Fig. 23. An sos-operator box with a dead transition, and a non-factorisable static box.

To summarise, in this and the next section we shall only consider operators fulfilling the conditions (OpNet1) and (OpNet2), i.e., simple static (hence safe and clean) factorisable

operator boxes, called *sos-operator boxes*. For an sos-operator box Ω, we will denote by fact_Ω the set comprising all factorisations of all complex markings reachable from the entry or exit marking of Ω, as well as the only factorisation $(\emptyset, \emptyset, \emptyset, T_\Omega)$ of the empty marking of Ω.

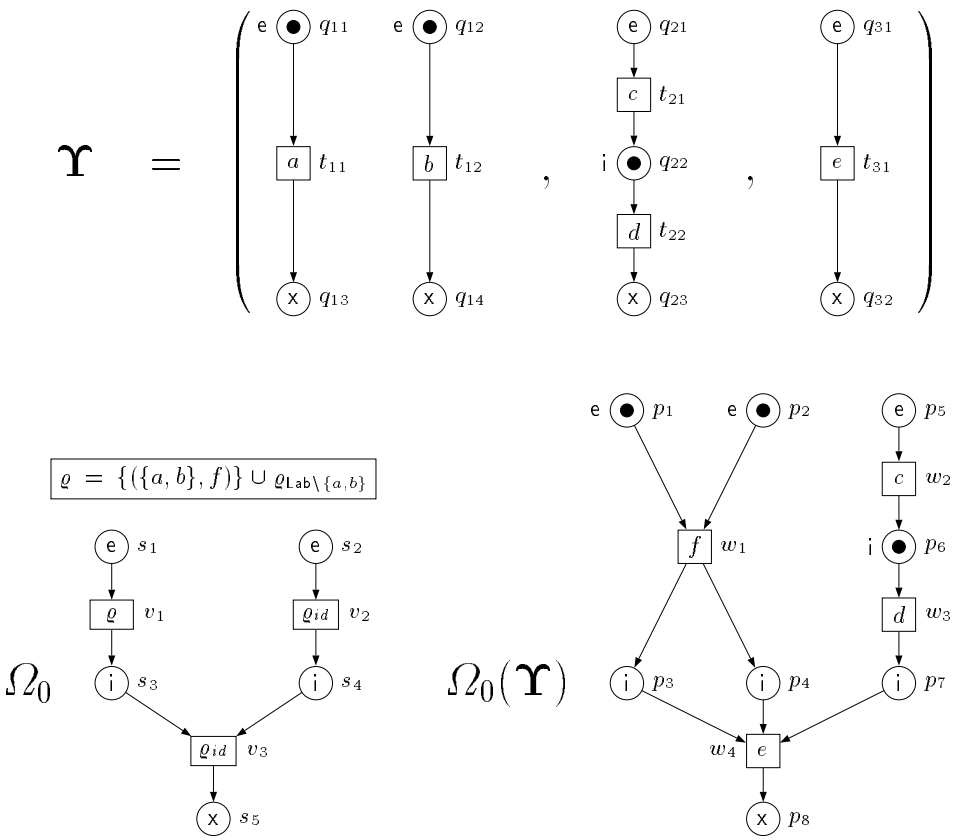
A running example Figure 24 shows an sos-operator box Ω_0 which will serve as a running example. Clearly, Ω_0 is a static box. A justification that Ω_0 is also factorisable is provided by table 1 which lists all complex markings (M, Q) reachable from the entry or exit marking of Ω_0, and their possible factorisations $(\mu_e, \mu_d, \mu_x, \mu_s)$. Thus fact_{Ω_0} comprises thirteen different factorisations, as the only factorisation of the empty marking, $(\emptyset, \emptyset, \emptyset, \{v_1, v_2, v_3\})$, also belongs to fact_{Ω_0}.

M	Q	μ_e	μ_d	μ_x	μ_s
$\{s_1, s_2\}$	$\emptyset$	$\{v_1, v_2\}$	$\emptyset$	$\emptyset$	$\{v_3\}$
$\{s_1\}$	$\{v_2\}$	$\{v_1\}$	$\{v_2\}$	$\emptyset$	$\{v_3\}$
$\{s_2\}$	$\{v_1\}$	$\{v_2\}$	$\{v_1\}$	$\emptyset$	$\{v_3\}$
$\{s_1, s_4\}$	$\emptyset$	$\{v_1\}$	$\emptyset$	$\{v_2\}$	$\{v_3\}$
$\{s_2, s_3\}$	$\emptyset$	$\{v_2\}$	$\emptyset$	$\{v_1\}$	$\{v_3\}$
$\emptyset$	$\{v_1, v_2\}$	$\emptyset$	$\{v_1, v_2\}$	$\emptyset$	$\{v_3\}$
$\{s_3\}$	$\{v_2\}$	$\emptyset$	$\{v_2\}$	$\{v_1\}$	$\{v_3\}$
$\{s_4\}$	$\{v_1\}$	$\emptyset$	$\{v_1\}$	$\{v_2\}$	$\{v_3\}$
$\{s_3, s_4\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{v_1, v_2\}$	$\{v_3\}$
		$\{v_3\}$	$\emptyset$	$\emptyset$	$\{v_1, v_2\}$
$\emptyset$	$\{v_3\}$	$\emptyset$	$\{v_3\}$	$\emptyset$	$\{v_1, v_2\}$
$\{s_5\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{v_3\}$	$\{v_1, v_2\}$

Table 1. Factorisations of the reachable markings for Ω_0.

Properties of factorisations A factorisation of a reachable marking (M, Q) is essentially a way of representing its real part, M, as the disjoint union of the pre-sets of a set of transitions, μ_e, and the post-sets of another set of transitions, μ_x. Intuitively, μ_e are transitions which can be concurrently executed at (M, Q), and μ_x are (possibly) transitions which have just been executed. However, such an interpretation may be somewhat misleading since, besides the fact that μ_e and μ_x may be infinite, it may happen that although (M, Q) is a reachable marking and $v \in \mu_x$, neither $(M \backslash v^\bullet, Q \cup \{v\})$ nor $((M \backslash v^\bullet) \cup {}^\bullet v, Q)$ are reachable from the entry or exit marking of Ω; it may even happen that v is a dead transition, whose sole role is to ensure the factorisability of Ω.

This is illustrated in figure 23 where the left operator is factorisable but the transition v_2 is dead from the entry and exit markings. Yet if we dropped this dead (hence supposedly useless) transition, the operator would become unfactorisable since after the occurrences of v_1 and v_3, leading to the marking $(\{i, x\}, \emptyset)$, the internal place i could no longer be factored



$p_1 = s_1 \lhd v_1 \lhd q_{11}$ $\quad p_2 = s_1 \lhd v_1 \lhd q_{12}$

$p_3 = s_3 \lhd \{v_1 \lhd q_{13}, v_3 \lhd q_{31}\}$ $\quad p_4 = s_3 \lhd \{v_1 \lhd q_{14}, v_3 \lhd q_{31}\}$

$p_5 = s_2 \lhd v_2 \lhd q_{21}$ $\quad p_6 = v_2 \lhd q_{22}$

$p_7 = s_4 \lhd \{v_2 \lhd q_{23}, v_3 \lhd q_{31}\}$ $\quad p_8 = s_5 \lhd v_3 \lhd q_{32}$

$w_1 = (v_1, f) \lhd \{t_{11}, t_{12}\}$ $\quad w_2 = (v_2, c) \lhd t_{21}$

$w_3 = (v_2, d) \lhd t_{22}$ $\quad w_4 = (v_3, e) \lhd t_{31}$

Fig. 24. A tuple of boxes $\boldsymbol{\Upsilon} = (\Upsilon_{v_1}, \Upsilon_{v_2}, \Upsilon_{v_3})$ and the sos-operator box Ω_0 for the running example, and the result of an application of Ω_0 to $\boldsymbol{\Upsilon}$.

out. Notice that, contrary to what we have said about μ_x, if $v \in \mu_e$ then both $(M \backslash {}^\bullet v, Q \cup \{v\})$ and $((M \backslash {}^\bullet v) \cup v^\bullet, Q)$ are reachable from the entry or exit marking of Ω.

In what follows, we will call a transition v of an sos-operator box Ω *reversible* if, for every complex marking (M, Q) reachable from ${}^\circ\Omega$ or Ω°, $v^\bullet \subseteq M$ implies that $(M \setminus v^\bullet, Q \cup \{v\})$ is also a marking reachable from ${}^\circ\Omega$ or Ω°. For the left box in figure 23, v_1 and v_3 are both reversible, while v_2 is not. In the operator box Ω_0 in figure 24, all three transitions are reversible.

Domain of application Let Ω be an sos-operator box and $\mathbf{\Sigma}$ be an Ω-tuple of static and dynamic boxes. The *factorisation* of $\mathbf{\Sigma}$ is the quadruple $\mu = (\mu_e, \mu_d, \mu_x, \mu_s)$ such that, for $\delta \in \{e, d, x, s\}$,

$$\mu_\delta = \begin{cases} \{v \in T_\Omega \mid \Sigma_v \in \mathsf{Box}^\delta\} & \text{if} \quad \delta \in \{e, x, s\} \\ \{v \in T_\Omega \mid \Sigma_v \in \mathsf{Box}^d \backslash (\mathsf{Box}^e \cup \mathsf{Box}^x)\} & \text{if} \quad \delta = d \ . \end{cases}$$

The *domain of application* of Ω, denoted by dom_Ω, is then defined as the set comprising every Ω–tuple of static and dynamic boxes $\mathbf{\Sigma}$ whose factorisation belongs to fact_Ω, and such that, for every $v \in T_\Omega$:

- If ${}^\bullet v \cap v^\bullet \neq \emptyset$ then Σ_v is ex-exclusive. (Dom1)
- If v is not reversible then Σ_v is x-directed. (Dom2)

The condition (Dom1) is meant to prevent the non-safeness of refinements in which a box is refined into a transition which is part of a side-loop (see the formula (4) for the marking in net refinement). The other condition, (Dom2), is related to the possible presence of dead transitions in an operator box. Referring again to the left sos-operator box in figure 23, one may observe that if we were allowed to refine the transition v_2 with a non-x-directed box Σ_{v_2}, then some behaviours originating from Σ_{v_2} could be possible from the entry marking of the refined net, while no behaviour beginning at the entry nor exit marking of the operator box can possibly involve v_2.[22]

Example (continued) Taking the sos-operator box in figure 24 and the factorisations in table 1, we obtain:

$$\mathsf{dom}_{\Omega_0} \ = \ \mathsf{Box}^d \times \mathsf{Box}^d \times \mathsf{Box}^s \ \cup \ \mathsf{Box}^s \times \mathsf{Box}^s \times \mathsf{Box}^d \ \cup \ \mathsf{Box}^s \times \mathsf{Box}^s \times \mathsf{Box}^s \ .$$

In figure 24, $\mathbf{\Upsilon} = (\Upsilon_{v_1}, \Upsilon_{v_2}, \Upsilon_{v_3})$ is an Ω_0-tuple with the factorisation, $(\{v_1\}, \{v_2\}, \emptyset, \{v_3\})$, which belongs to fact_{Ω_0}. Hence $\mathbf{\Upsilon}$ belongs to the domain of application of the sos-operator box Ω_0.[23] The result of performing the refinement $\Omega_0(\mathbf{\Upsilon})$ is shown in figure 24.

[22] Actually, one could have required that no transition be enabled at $\Sigma^\circ_{v_2}$, but x-directedness is a simple structural sufficient condition for this.

[23] Notice that (Dom1) and (Dom2) are satisfied in this case since Ω_0 is pure and all its transitions are reversible.

7.2 Structured operational semantics of composite boxes

A typical structured operational semantics rule in process algebras stipulates that if a process term $expr = op(expr_1, \ldots, expr_n)$ can make a move

$$expr \stackrel{act}{\longrightarrow} expr'$$

then act can be composed out of some individual moves of the sub-terms $expr_1, \ldots, expr_n$. And conversely, if the sub-terms can individually make moves satisfying some prescribed conditions, then the whole term can make a combined move. We will now apply this scheme to nets obtained using sos-operator boxes.

Let Ω be an sos-operator box and $\mathbf{\Sigma}$ be an Ω–tuple in its domain, dom_Ω. When dealing with the operational semantics of the compositional box $\Omega(\mathbf{\Sigma})$, we shall use the notation

$$(\Omega \ : \ \mathbf{\Sigma}) \ \stackrel{U}{\longrightarrow} \ (\Omega \ : \ \mathbf{\Theta}) \tag{13}$$

to mean that the boxes $\mathbf{\Sigma}$ can individually make moves which, when combined, yield step U and lead to new boxes $\mathbf{\Theta}$. By definition, this will be the case whenever U is a finite set of transitions of $\Omega(\mathbf{\Sigma})$ and, for every transition v in Ω, $U \cap \mathsf{T}^v_{\mathsf{new}} = \{u_1, \ldots, u_k\}$ is a finite set of transitions such that

$$\Sigma_v \ [\ \mathsf{trees}(u_1) + \cdots + \mathsf{trees}(u_k) \ \rangle \ \Theta_v \ . \tag{14}$$

In fact, the multiset sum in (14) is a disjoint union of k sets. Moreover, $\Sigma_v = \Theta_v$ for all but finitely many v.

Instead of expressing behaviours in terms of transitions, it is also possible to express them using actions, through the labelling function

$$\lambda_{\Omega(\mathbf{\Sigma})}(U) \ = \ \sum_{u \in U} \{\lambda_{\Omega(\mathbf{\Sigma})}(u)\}$$

which returns multisets rather than sets since different transitions may have the same label.

Example (continued) For the boxes in figure 24 we have

$$(\Omega_0 \ : \ \mathbf{\Upsilon}) \ \stackrel{\{w_1, w_3\}}{\longrightarrow} \ (\Omega_0 \ : \ \mathbf{\Theta}) \tag{15}$$

where $\mathbf{\Theta} = (\Theta_{v_1}, \Theta_{v_2}, \Theta_{v_3}) = (\Upsilon_{v_1}, \Upsilon_{v_2}, \Upsilon_{v_3})$. Indeed, by setting $U = \{w_1, w_3\}$ we obtain: $U \cap \mathsf{T}^{v_1}_{\mathsf{new}} = \{w_1\}$ and $U \cap \mathsf{T}^{v_2}_{\mathsf{new}} = \{w_3\}$ and $U \cap \mathsf{T}^{v_3}_{\mathsf{new}} = \emptyset$. Moreover, $\mathsf{trees}(w_1) = \{t_{11}, t_{12}\}$ and $\mathsf{trees}(w_3) = \{t_{22}\}$. And, finally,

$$\Upsilon_{v_1} \ [\{t_{11}, t_{12}\}\rangle \ \Theta_{v_1} \quad \text{and} \quad \Upsilon_{v_2} \ [\{t_{22}\}\rangle \ \Theta_{v_2} \quad \text{and} \quad \Upsilon_{v_3} \ [\emptyset\rangle \ \Theta_{v_3} \ .$$

Notice that $\{w_1, w_3\}$ is also a valid step for the box $\Omega_0(\mathbf{\Upsilon})$ and $\Omega_0(\mathbf{\Upsilon}) \ [\{w_1, w_3\}\rangle \ \Omega_0(\mathbf{\Theta})$.

SOS rule for boxes The transformation captured by (13) is sound with respect to the standard net semantics.

Theorem 26. *Let $\mathbf{\Sigma}$ be a tuple in the domain of an sos-operator box Ω. If $\mathbf{\Theta}$ and U are as in (13), then $\mathbf{\Theta} \in \mathsf{dom}_\Omega$ and $\Omega(\mathbf{\Sigma})\ [U\rangle\ \Omega(\mathbf{\Theta})$.* □

A direct converse of theorem 26 does not, in general, hold true. For if we consider the tuple of boxes $\mathbf{\Theta}$ as in (15), then $\Omega_0(\mathbf{\Theta})\ [\{w_4\}\rangle$, yet no move at all of the form

$$(\Omega_0\ :\ \mathbf{\Theta}) \xrightarrow{U} (\Omega_0\ :\ \mathbf{\Theta}')$$

for a non-empty U is possible. This is so because, when composing the nets, the tokens contributed by Θ_{v_1} and Θ_{v_2} are inserted into the composed net in such a way that they could all have been contributed by Θ_{v_3} as well. More precisely, we have $\Omega(\mathbf{\Theta}) = \Omega(\mathbf{\Psi})$, where $\Psi_{v_1} = \lfloor \Upsilon_{v_1} \rfloor$ and $\Psi_{v_2} = \lfloor \Upsilon_{v_2} \rfloor$ and $\Psi_{v_3} = \overline{\Upsilon_{v_3}}$. And the situation is now different since a move

$$(\Omega_0\ :\ \mathbf{\Psi}) \xrightarrow{\{w_4\}} (\Omega_0\ :\ \mathbf{\Psi}')$$

is possible with $\Psi'_{v_1} = \Psi_{v_1}$, $\Psi'_{v_2} = \Psi_{v_2}$ and $\Psi'_{v_3} = \underline{\Psi_{v_3}}$. A conclusion to be drawn from this example is that the markings in a tuple of boxes $\mathbf{\Sigma}$ may need to be re-arranged before attempting to derive a move which is admitted by their composition $\Omega(\mathbf{\Sigma})$. Such a re-arrangement is formalised by a similarity relation $\equiv_\Omega$ on Ω–tuples of boxes.[24]

Let Ω be an sos-operator box and $\mathbf{\Sigma}$ and $\mathbf{\Theta}$ be Ω–tuples of static and dynamic boxes whose factorisations are respectively μ and κ. Then $\mathbf{\Sigma} \equiv_\Omega \mathbf{\Theta}$ if μ and κ are factorisations of the same complex marking of Ω and $\lfloor \mathbf{\Sigma} \rfloor = \lfloor \mathbf{\Theta} \rfloor$ and $\Sigma_v = \Theta_v$, for every $v \in \mu_d = \kappa_d$.

$\equiv_\Omega$ is an equivalence relation which is closed in the domain of application of Ω, and relates tuples which yield the same boxes through refinement, $\Omega(\mathbf{\Sigma}) = \Omega(\mathbf{\Theta})$. A partial converse of the latter property also holds: if $\mathbf{\Sigma}$ and $\mathbf{\Theta}$ are tuples in the domain of Ω such that $\lfloor \mathbf{\Sigma} \rfloor = \lfloor \mathbf{\Theta} \rfloor$ and $\Omega(\mathbf{\Sigma}) = \Omega(\mathbf{\Theta})$, then $\mathbf{\Sigma} \equiv_\Omega \mathbf{\Theta}$.

With help of the similarity relation $\equiv_\Omega$ it is possible to reverse theorem 26.

Theorem 27. *Let $\mathbf{\Psi}$ be a tuple in the domain of an sos-operator box Ω. If $\Omega(\mathbf{\Psi})\ [U\rangle\ \Sigma$ then there are $\mathbf{\Sigma}$ and $\mathbf{\Theta}$ in dom_Ω such that $\mathbf{\Psi} \equiv_\Omega \mathbf{\Sigma}$, $\Omega(\mathbf{\Theta}) = \Sigma$ and (13) holds.* □

Various important consequences may be derived from theorems 26 and 27. In particular, they imply that (Req1) and (Req2) are satisfied. That is, $\Omega(\mathbf{\Sigma})$ is a static or dynamic box, and if Σ is a net derivable from $\overline{\Omega(\mathbf{\Sigma})}$ or $\underline{\Omega(\mathbf{\Sigma})}$, then there is a tuple $\mathbf{\Theta}$ in the domain of Ω such that $\lfloor \mathbf{\Sigma} \rfloor = \lfloor \mathbf{\Theta} \rfloor$ and $\Sigma = \Omega(\mathbf{\Theta})$. As a summary, the SOS rule for boxes can be presented in the following way:

$$\frac{\left(\Omega\ :\ \mathbf{\Theta}\right) \xrightarrow{U} \left(\Omega\ :\ \mathbf{\Theta}'\right)}{\Omega(\mathbf{\Sigma})\ [U\rangle\ \Omega(\mathbf{\Sigma}')} \qquad \mathbf{\Sigma} \equiv_\Omega \mathbf{\Theta}\ ,\ \mathbf{\Sigma}' \equiv_\Omega \mathbf{\Theta}'$$

[24] Notice the analogy between $\equiv_\Omega$ and $\equiv$ (discussed in section 1.1), which is of course no accident.

Domain restrictions and recursion In the next definition, we will use Box^{xcl}, Box^{edir} and Box^{xdir} to denote, respectively, the sets of ex-exclusive, e-directed and x-directed boxes.

Consider again the recursive system of equations (5) from section 5, under the assumption that each box occurring in it is an sos-operator box or a static plain box. Let $\mathcal{X}^{wf}$, $\mathcal{X}^{xcl}$, $\mathcal{X}^{edir}$ and $\mathcal{X}^{xdir}$ be the largest subsets of $\mathcal{X}$ – called respectively the *well formed*, *ex-exclusive*, *e-directed* and *x-directed* variables[25] – such that the following hold, where we assume that a variable $X \in \mathcal{X}$ has the recursive equation $X \stackrel{\mathrm{df}}{=} \Omega(\mathbf{\Delta})$:

- All ex-exclusive, e-directed and x-directed variables are well formed. (Var1)
- If $X \in \mathcal{X}^{wf}$ then Ω is an sos-operator box and, for every $v \in T_\Omega$:
  - $\Delta_v \in \mathsf{Box}^s \cup \mathcal{X}^{wf}$
  - if ${}^\bullet v \cap v^\bullet \neq \emptyset$ then $\Delta_v \in \mathsf{Box}^{xcl} \cup \mathcal{X}^{xcl}$
  - if v is not reversible then $\Delta_v \in \mathsf{Box}^{xdir} \cup \mathcal{X}^{xdir}$. (Var2)
- If $X \in \mathcal{X}^{xcl}$ then Ω is ex-exclusive and, for every $v \in T_\Omega$ satisfying

$$ {}^\bullet v \cap {}^\circ\Omega \neq \emptyset \neq v^\bullet \cap \Omega^\circ \quad \text{or} \quad v^\bullet \cap {}^\circ\Omega \neq \emptyset \neq {}^\bullet v \cap \Omega^\circ \ , $$

  it is the case that $\Delta_v \in \mathsf{Box}^{xcl} \cup \mathcal{X}^{xcl}$. (Var3)
- If $X \in \mathcal{X}^{edir}$ then Ω is e-directed and, for every $v \in ({}^\circ\Omega)^\bullet$, $\Delta_v \in \mathsf{Box}^{edir} \cup \mathcal{X}^{edir}$. (Var4)
- If $X \in \mathcal{X}^{xdir}$ then Ω is x-directed and, for every $v \in {}^\bullet(\Omega^\circ)$, $\Delta_v \in \mathsf{Box}^{xdir} \cup \mathcal{X}^{xdir}$. (Var5)

Theorem 28. *The system of recursive equations (5) has at least one solution in the domain of plain boxes with empty markings. Moreover, the following hold for its every solution.*

- *If X is a well formed variable then the box corresponding to X is a static box.*
- *If $X \in \mathcal{X}^\delta$ then the box corresponding to X belongs to Box^δ, for $\delta \in \{xcl, edir, xdir\}$.*

□

7.3 Further restrictions on application domain

The conditions (Dom1) and (Dom2) imposed on the domain of an sos-operator box are sufficient for the main compositionality results; in particular, for showing the full consistency between the net semantics and operational semantics in the process algebra presented in section 8.4. So far, however, this is unrelated to any process algebraic considerations, except that the shape of the SOS rule is inspired by process algebra theory. Starting with this section, we shall discuss relationships between operations on nets (realised by operator boxes) and process algebraic operators. In the present section, we address a behavioural consistency issue that pertains to such a discussion.

Consider the operator box Ω^* on the left hand side of figure 25, whose terminated behaviours are modelled by the expression $v_2^* v_1$ (in other words, any number of repetitions of the net refining v_2 followed by a behaviour of the net refining v_1). Allowing operator boxes like Ω^* raises the danger of creating composite nets which are not in agreement with

[25] In this and other syntactic definitions, the superscripts indicate the relevant properties of the corresponding boxes; more precisely, *wf* indicates well-formedness, *xcl* indicates ex-exclusiveness, *edir* indicates e-directedness, and *xdir* indicates x-directedness.

an intuitive behavioural meaning of operations specified by operator boxes. For consider an expression such as $\Omega_{\square}(\mathsf{N}_\alpha, \Omega^*(\mathsf{N}_\beta, \mathsf{N}_\gamma))$ (using the boxes in figure 27), whose behaviour is, intuitively: do either α, or $\beta^*\gamma$. According to our intuition about the operation specified by Ω^*, one might expect that the net corresponding to this expression could be constructed by first applying Ω^* to N_β and N_γ, and then putting the result in a choice with N_α, yielding the net shown on the right-hand side of figure 25. This net, however, allows evolutions such as $\{\beta\}\{\beta\}\{\alpha\}$, which do not correspond to what one expects from a choice construct. This phenomenon is not particular to our approach, nor to Petri nets in general (cf. [1,30]), and it does not invalidate our coherence results (see section 8.4) since the sos-semantics yields exactly the same 'undesirable' behaviour.

If we imposed ex-directedness on all boxes, then the problem would disappear. However, such a strict solution is not always desirable, because boxes such as Ω^* are useful in modelling guarded while-loops in programming languages (basically, v_1 corresponds to the negation of the guard(s), and v_2 to the repetitive behaviour). On the other hand, if it is ascertained that a loop does not occur initially in an enclosing choice, or in another enclosing loop, then the problem also disappears, and ex-directedness is no longer required. To our experience, in practical programming languages, such a case (i.e., a loop occurring initially in enclosing loops or choices) does not arise, and we therefore formulate a set of conditions in such a way that it is excluded, and Ω^* still is an allowed operator box.

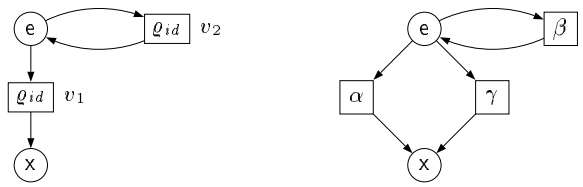


Fig. 25. Operator box Ω^* and a net modelling (tentatively) $\Omega_{\square}(\mathsf{N}_\alpha, \Omega^*(\mathsf{N}_\beta, \mathsf{N}_\gamma))$.

Generalising the above observation, we formulate five additional conditions on an Ω-tuple $\mathbf{\Sigma}$ in the domain of application of an sos-operator box Ω.

- If $v \neq w$ and ${}^\bullet v \cap {}^\bullet w \neq \emptyset$, then $\Sigma_v, \Sigma_w \in \mathsf{Box}^{edir}$. (Beh1)
- If $v \neq w$ and $v^\bullet \cap w^\bullet \neq \emptyset$, then $\Sigma_v, \Sigma_w \in \mathsf{Box}^{xdir}$. (Beh2)
- If $v^\bullet \cap {}^\bullet w \neq \emptyset$, then $\Sigma_v \in \mathsf{Box}^{xdir}$ or $\Sigma_w \in \mathsf{Box}^{edir}$. (Beh3)
- If ${}^\bullet v \cap \Omega^\circ \neq \emptyset$, then $\Sigma_v \in \mathsf{Box}^{edir}$. (Beh4)
- If $v^\bullet \cap {}^\circ\Omega \neq \emptyset$, then $\Sigma_v \in \mathsf{Box}^{xdir}$. (Beh5)

Adding (Beh1-5) to the existing conditions makes the problems like that encountered earlier on disappear. With (Beh1), it will not be possible to start the execution of Σ_v and later, without finishing it, to enter Σ_w for a conflicting w because we came back to the initial state of Σ_v (notice that, again, the problem really concerns the possibility of coming back to the initial state and e-directedness is a simple structural condition excluding this kind of behaviour). (Beh2) prohibits to finish the execution of Σ_v and afterwards to enter Σ_w from the rear (notice that here the problem concerns the possibility of executing something from

an exit marking which, perhaps unexpectedly, makes the situation rather different from the previous one). (Beh3) excludes a situation, combining the previous two problems, where we start the execution of Σ_w and later, without finishing it, we come back to the initial state and (re)enter Σ_v from the rear. (Beh4) excludes the situation whereby we reach the exit marking, start the execution of Σ_v and return to the exit marking without finishing Σ_v, so that a completed history is not composed out of completed histories of the components corresponding to a completed history of Ω. Finally, (Beh5) excludes the situation whereby, from the entry marking of the composed net, we can execute a behaviour originating from Σ_v° which bears no relationship to any possible behaviour of Ω (this intuitively corresponds to the backward reachability in Ω, which is not allowed). Note that (Beh1-5) are truly structural conditions, as opposed to (Dom1-2) and (OpNet1-2).

8 A Process Algebra And Its Two Consistent Semantics

This section describes an algebra of process expressions based on the sos-operator boxes, called the *box algebra*. The box algebra is in fact a *meta-model* parameterised by two non-empty, possibly infinite, sets of Petri nets: a set $\mathsf{ConstBox}$ of static and dynamic plain boxes which provide a denotational semantics of simple process expressions, including constants, and a disjoint set OpBox of sos-operator boxes which provide interpretation for the connectives. The only assumption about the sos-operator boxes in OpBox and the static boxes in $\mathsf{ConstBox}$ is that they have disjoint sets of simple root-only trees as their place and transition names, i.e., for all distinct static and/or operator boxes Σ and Θ in $\mathsf{OpBox} \cup \mathsf{ConstBox}$,

$$S_\Sigma \cup T_\Sigma \subseteq \mathsf{P_{root}} \cup \mathsf{T_{root}} \quad \text{and} \quad S_\Sigma \cap S_\Theta = T_\Sigma \cap T_\Theta = \emptyset \ . \tag{16}$$

These assumptions are needed, in particular, to apply the results on solving recursive systems of equations on boxes. Moreover, we require that the operator boxes in OpBox be finite (but we do allow infinite plain boxes in $\mathsf{ConstBox}$).[26]

Syntax We consider an algebra of process expressions over the signature

$$\mathsf{Const} \ \cup \ \{\overline{(.)}, \underline{(.)}\} \ \cup \ \{\mathsf{op}_\Omega \mid \Omega \in \mathsf{OpBox}\}$$

where Const is a fixed non-empty set of *constants* which will be modelled through the boxes in $\mathsf{ConstBox}$, $\overline{(.)}$ and $\underline{(.)}$ are two unary operators, and each op_Ω is a connective of the algebra indexed by an sos-operator box taken from the set OpBox. The set of constants is partitioned into the *static* constants, Const^s, and *dynamic* constants, Const^d; moreover, there are two distinct disjoint subsets of Const^d, denoted by Const^e and Const^x, and respectively called the *entry* and *exit* constants. We will also use a fixed set $\mathcal{X}$ of *process variables*. Although we use the symbols $\overline{(.)}$ and $\underline{(.)}$ to denote both operations on boxes and process algebra connectives, it will always be clear from the context what is the intended interpretation.

There are four classes of process expressions corresponding to previously introduced classes of plain boxes: the *entry*, *dynamic*, *exit* and *static* expressions, denoted respectively

[26] It is possible to allow infinite operator boxes as well, without losing any of the results stated in this section [8].

by Expr^e, Expr^d, Expr^x and Expr^s. Collectively, we will refer to them as the *box expressions*, Expr^{box}. We will also use a counterpart of the notion of the factorisation of a tuple of boxes. For an sos-operator box Ω and an Ω-tuple of box expressions $\mathbf{D}$, we define the *factorisation* of $\mathbf{D}$ to be the quadruple $\mu = (\mu_e, \mu_d, \mu_x, \mu_s)$ such that $\mu_\delta = \{v \mid D_v \in \mathsf{Expr}^\delta\}$, for $\delta \in \{e, x, s\}$, and $\mu_d = \{v \mid D_v \in \mathsf{Expr}^d \backslash (\mathsf{Expr}^e \cup \mathsf{Expr}^x)\}$. The syntax for the box expressions Expr^{box} is given by:

$$\begin{array}{lllllll}
\mathsf{Expr}^s & E & ::= & c^s & | & X & | \quad \mathsf{op}_\Omega(\mathbf{E}) \\
\mathsf{Expr}^e & F & ::= & c^e & | & \overline{E} & | \quad \mathsf{op}_\Omega(\mathbf{F}) \\
\mathsf{Expr}^x & G & ::= & c^x & | & \underline{E} & | \quad \mathsf{op}_\Omega(\mathbf{G}) \\
\mathsf{Expr}^d & H & ::= & c^d & | & F & | \quad G \quad | \quad \mathsf{op}_\Omega(\mathbf{H})
\end{array} \tag{17}$$

where $c^\delta \in \mathsf{Const}^\delta$, for $\delta \in \{e, x, s\}$, and $c^d \in \mathsf{Const}^d \backslash (\mathsf{Const}^e \cup \mathsf{Const}^x)$ are constants; $X \in \mathcal{X}$ is a process variable; $\Omega \in \mathsf{OpBox}$ is an sos-operator box; and $\mathbf{E}$, $\mathbf{F}$, $\mathbf{G}$ and $\mathbf{H}$ are Ω–tuples of box expressions. These tuples have to satisfy some conditions determined by the domain of application of the net operator induced by Ω. More precisely, the factorisations of $\mathbf{E}$, $\mathbf{F}$ and $\mathbf{G}$ are respectively factorisations of the complex empty, entry and exit marking of Ω, and the factorisation of $\mathbf{H}$ is a factorisation of a marking reachable from the entry or exit marking of Ω different from ${}^\circ\Omega$ and Ω°.

In what follows, expressions in $\mathsf{Const} \cup \overline{\mathsf{Const}^s} \cup \underline{\mathsf{Const}^s}$, will be referred to as *flat*.

The above syntax is liberal in the sense that it only reflects properly the first part of the definition of the domain of an operator box, without reflecting the conditions (Dom1) and (Dom2). We could, of course, incorporate these in a purely syntactic way, by taking advantage of the results characterising compositionally and recursively defined ex-exclusive and x-directed boxes (see, for example, theorem 28).[27] However, we found it simpler to describe syntactic restrictions implied by (Dom1) and (Dom2) explicitly, after giving a translation from the constant expressions to boxes.

For every process variable $X \in \mathcal{X}$, there is a unique *defining equation*

$X \stackrel{\text{df}}{=} \mathsf{op}_\Omega(\mathbf{L})$ where $\Omega \in \mathsf{OpBox}$ is an sos-operator box and $\mathbf{L}$ is an Ω–tuple of process variables and static constants.

As in the domain of boxes, it is convenient to have a notation for turning an expression into a corresponding static one. We again use $\lfloor . \rfloor$ to denote such an operation. To achieve the desired effect, we assume that for each dynamic constant c there is a unique static constant denoted by $\lfloor c \rfloor$ and, if C is an expression, then $\lfloor C \rfloor$ is the static expression obtained by removing all occurrences of $\overline{(.)}$ and $\underline{(.)}$, and replacing every occurrence of each dynamic constant c by the corresponding static constant $\lfloor c \rfloor$.

Example (continued) We will use the running example based on the boxes depicted in figure 24 in order to construct a simple algebra of process expressions. More precisely, we will consider the *Do It Yourself (DIY) algebra* based on the following two sets of boxes:

$$\mathsf{OpBox} = \{\Omega_0\} \quad \text{and} \quad \mathsf{ConstBox} = \{\Phi_1, \Phi_{11}, \Phi_{12}\} \cup \{\Phi_2, \Phi_{21}, \Phi_{22}, \Phi_{23}\} \cup \{\Phi_3\}$$

[27] We will provide a concrete instance of such a syntax in the next section.

where $\lfloor \Phi_{ij} \rfloor = \Phi_i = \lfloor \Upsilon_{v_i} \rfloor$, for all i and j, and the following hold:

$$M_{\Phi_{11}} = \{q_{11}, q_{14}\} \quad \text{and} \quad M_{\Phi_{12}} = \{q_{13}, q_{12}\} \quad \text{and} \quad M_{\Phi_{2k}} = \{q_{2k}\} \quad (\text{for } k = 1, 2, 3) .$$

The constants of the DIY algebra correspond to the boxes in ConstBox:

$$\mathsf{Const}^e = \{c_{21}\} \qquad \mathsf{Const}^s = \{c_1, c_2, c_3\}$$
$$\mathsf{Const}^x = \{c_{23}\} \qquad \mathsf{Const}^d = \{c_{11}, c_{12}, c_{21}, c_{22}, c_{23}\} .$$

Moreover, $\lfloor c_{ij} \rfloor = c_i$, for every dynamic constant c_{ij}. The syntax of the DIY algebra is obtained by instantiating (17) with concrete constants and operator introduced above, and taking into account the factorisations in fact_{Ω_0} (see table 1). For example, the syntax for the static and entry DIY expressions is:

$$\mathsf{Expr}^s \quad E ::= \mathsf{op}_{\Omega_0}(E, E, E) \mid c_1 \mid c_2 \mid c_3 \mid X$$
$$\mathsf{Expr}^e \quad F ::= \mathsf{op}_{\Omega_0}(F, F, E) \mid c_{21} \mid \overline{E}$$

An example of valid dynamic expression is $\mathsf{op}_{\Omega_0}(\overline{c_1}, c_{22}, c_3)$ which, as we shall later see, corresponds to the net refinement $\Omega_0(\mathbf{\Upsilon})$ of figure 24.

Syntactic restrictions resulting from (Dom1) and (Dom2) In the semantics of variables, we intend to use theorem 28, and therefore need to define syntactic counterparts of well formed, ex-exclusive, e-directed and x-directed variables. To this end, we will now also have to consider (Dom1) and (Dom2).

Let E^{wf}, E^{xcl}, E^{edir} and E^{xdir} be the largest sets of expressions in Expr^{box} – called respectively the *well formed*, *ex-exclusive*, *e-directed* and *x-directed* box expressions – such that the following hold:

- All ex-exclusive, e-directed and x-directed expressions are well formed. (Expr1)
- If $\mathsf{op}_\Omega(\mathbf{D}) \in E^{wf}$ then, for every $v \in T_\Omega$:
  - $D_v \in E^{wf}$
  - if ${}^\bullet v \cap v^\bullet \neq \emptyset$ then $D_v \in E^{xcl}$
  - if v is not reversible then $D_v \in E^{xdir}$. (Expr2)
- If $\mathsf{op}_\Omega(\mathbf{D}) \in E^{xcl}$ then Ω is ex-exclusive and, for every $v \in T_\Omega$ satisfying

$$ {}^\bullet v \cap {}^\circ\Omega \neq \emptyset \neq v^\bullet \cap \Omega^\circ \quad \text{or} \quad v^\bullet \cap {}^\circ\Omega \neq \emptyset \neq {}^\bullet v \cap \Omega^\circ ,$$

  it is the case that $D_v \in E^{xcl}$. (Expr3)
- If $\mathsf{op}_\Omega(\mathbf{D}) \in E^{edir}$ then Ω is e-directed and, for every $v \in ({}^\circ\Omega)^\bullet$, $D_v \in E^{edir}$. (Expr4)
- If $\mathsf{op}_\Omega(\mathbf{D}) \in E^{xdir}$ then Ω is x-directed and, for every $v \in {}^\bullet(\Omega^\circ)$, $D_v \in E^{xdir}$. (Expr5)
- If $\overline{E} \in \mathsf{Expr}^\delta$, or $\underline{E} \in \mathsf{Expr}^\delta$, or $X \in \mathsf{Expr}^\delta$ and $X \stackrel{\text{df}}{=} E$, then $E \in \mathsf{Expr}^\delta$, for $\delta \in \{wf, xcl, edir, xdir\}$. (Expr6)
- $\mathsf{Const} \subseteq E^{wf}$ and $\mathsf{Const}^\delta = \mathsf{Const} \cap \mathsf{Expr}^\delta$, for $\delta \in \{xcl, edir, xdir\}$. (Expr7)

Notice that (Expr1)–(Expr5) correspond to (Var1)–(Var5).

8.1 Denotational semantics

When dealing with the denotational semantics of the box algebra we will first define the semantics of constants, then of process variables and, finally, of compound box expressions.

Constants Constant expressions are mapped onto constant boxes of corresponding types, i.e., for every constant c and $\delta \in \{e, d, x, s\}$,

$$c \in \mathsf{Const}^{\delta} \iff \mathsf{box}(c) \in \mathsf{Box}^{\delta} \cap \mathsf{ConstBox} \ . \tag{18}$$

It assumed that, for every dynamic constant, the underlying box is the same as that for the corresponding static constant, and that it is reachable from the entry or exit marking of the latter; i.e., for every c in Const^{d},

$$\lfloor \mathsf{box}(c) \rfloor = \mathsf{box}(\lfloor c \rfloor) \quad \text{and} \quad \mathsf{box}(c) \in (\ [\overline{\mathsf{box}(\lfloor c \rfloor)}\rangle \ \cup \ [\underline{\mathsf{box}(\lfloor c \rfloor)}\rangle \ ) \ .$$

It is also assumed that, for every non-entry and non-exit dynamic box reachable from an initially or terminally marked constant box, there is a corresponding dynamic constant, i.e., for every c in Const^{s},

$$(\ [\overline{\mathsf{box}(c)}\rangle \ \cup \ [\underline{\mathsf{box}(c)}\rangle \ ) \setminus \{\overline{\mathsf{box}(c)}, \underline{\mathsf{box}(c)}\} \subseteq \mathsf{box}(\mathsf{Const}^{d}) \ .$$

Variables With each defining equation $X \stackrel{\text{df}}{=} \mathsf{op}_{\Omega}(\mathbf{L})$, we associate an equation on boxes, $X \stackrel{\text{df}}{=} \Omega(\mathbf{\Psi})$, where $\Psi_v = L_v$ if L_v is a process variable (treated here as a box variable), and $\Psi_v = \mathsf{box}(L_v)$ if L_v is a static constant. This creates a system of net equations of the type considered in section 5. Such a system can then be treated by theorem 28 and so it follows that it has at least one solution in the domain of static boxes. We then fix *any* such solution $\mathsf{sol} : \mathcal{X} \to \mathsf{Box}^{s}$ (for instance, the maximal or a minimal one, following observations made in sections 5 and 6) and define, for every process variable X, $\mathsf{box}(X) = \mathsf{sol}(X)$. As a result, for every variable X defined by the equation $X \stackrel{\text{df}}{=} \mathsf{op}_{\Omega}(\mathbf{L})$,

$$\mathsf{box}(X) \ = \ \Omega(\mathsf{box}(\mathbf{L})) \ \in \ \mathsf{Box}^{s} \ .$$

Moreover, it follows from theorem 28 that if X is an ex-exclusive or x-directed variable then $\mathsf{box}(X)$ is ex-exclusive or x-directed, respectively.

Compound expressions The definition of the semantical mapping box is completed by considering all the remaining static and dynamic expressions, following the syntax (17). The box mapping is a homomorphism; hence, for every static or dynamic expression $\mathsf{op}_{\Omega}(\mathbf{D})$ and every static expression E,

$$\mathsf{box}(\mathsf{op}_{\Omega}(\mathbf{D})) = \Omega(\mathsf{box}(\mathbf{D})) \quad \text{and} \quad \mathsf{box}(\overline{E}) = \overline{\mathsf{box}(E)} \quad \text{and} \quad \mathsf{box}(\underline{E}) = \underline{\mathsf{box}(E)} \ .$$

The semantical mapping always returns a box, and the assumed type consistency between constants and their denotations, (18), carries over to the remaining box expressions.

Theorem 29. *For every box expression D, $\mathsf{box}(D)$ is a static or dynamic box. Moreover, for every $\delta \in \{e, d, x, s\}$, $D \in \mathsf{Expr}^{\delta}$ if and only if $\mathsf{box}(D) \in \mathsf{Box}^{\delta}$.* □

Example (continued) In the case of the DIY algebra, for every static constant c_i, $\mathsf{box}(c_i) = \Phi_i$, and for every dynamic constant c_{ij}, $\mathsf{box}(c_{ij}) = \Phi_{ij}$. Thus, for example, the box in figure 24 can be derived in the following way:

$$\begin{aligned}\mathsf{box}(\mathsf{op}_{\Omega_0}(\overline{c_1}, c_{22}, c_3)) &= \Omega_0(\mathsf{box}(\overline{c_1}), \mathsf{box}(c_{22}), \mathsf{box}(c_3)) \;=\; \Omega_0(\overline{\mathsf{box}(c_1)}, \Phi_{22}, \Phi_3) \\ &= \Omega_0(\overline{\Phi_1}, \Phi_{22}, \Phi_3) \;=\; \Omega_0(\Upsilon_{v_1}, \Upsilon_{v_2}, \Upsilon_{v_3}) \;=\; \Omega_0(\boldsymbol{\Upsilon}) \;.\end{aligned}$$

8.2 Structured operational semantics

Similarity relation on expressions Next, we define a structural similarity relation on box expressions, $\equiv$, which provides partial identification of box expressions with the same denotational semantics. It is defined as the least equivalence relation on box expressions such that the following hold.

- For all flat expressions D and H satisfying $\mathsf{box}(D) = \mathsf{box}(H)$, and all defining equations $X \stackrel{\text{df}}{=} \mathsf{op}_\Omega(\mathbf{L})$,

$$\boxed{D \equiv H \qquad X \equiv \mathsf{op}_\Omega(\mathbf{L}) \;.} \tag{19}$$

– For every sos-operator box Ω and every factorisation μ of ${}^\circ\Omega$ or Ω°, we have respectively:

$$\boxed{\mathsf{op}_\Omega(\mathbf{D}) \equiv \overline{\mathsf{op}_\Omega(\mathbf{H})} \qquad \mathsf{op}_\Omega(\mathbf{D}) \equiv \underline{\mathsf{op}_\Omega(\mathbf{H})} \;.} \tag{20}$$

where $\mathbf{D}$ and $\mathbf{H}$ are Ω-tuples of expressions such that, for every $v \in T_\Omega$: $D_v = \overline{H_v}$ if $v \in \mu_e$; $D_v = \underline{H_v}$ if $v \in \kappa_x$; and $D_v = H_v$ otherwise.

- For every sos-operator box Ω, for every complex marking reachable from the entry marking of Ω, different from ${}^\circ\Omega$ and Ω°, and for every pair of different factorisations μ and κ of that marking,

$$\boxed{\mathsf{op}_\Omega(\mathbf{D}) \equiv \mathsf{op}_\Omega(\mathbf{H})} \tag{21}$$

where $\mathbf{D}$ and $\mathbf{H}$ are Ω-tuples of expressions for which there is an Ω-tuple of expressions $\mathbf{F}$ such that, for every $v \in T_\Omega$: $D_v = \overline{F_v}$ if $v \in \mu_e$; $D_v = \underline{F_v}$ if $v \in \mu_x$; and $D_v = F_v$ otherwise. Similarly, for every $v \in T_\Omega$: $H_v = \overline{F_v}$ if $v \in \kappa_e$; $H_v = \underline{F_v}$ if $v \in \kappa_x$; and $H_v = F_v$ otherwise.[28]

- For all static expressions E and F, for every sos-operator box Ω, and for all Ω-tuples of expressions $\mathbf{D}$ and $\mathbf{H}$ with factorisations in fact_Ω:

$$\boxed{\frac{E \equiv F}{\overline{E} \equiv \overline{F}} \qquad \frac{E \equiv F}{\underline{E} \equiv \underline{F}} \qquad \frac{\mathbf{D} \equiv \mathbf{H}}{\mathsf{op}_\Omega(\mathbf{D}) \equiv \mathsf{op}_\Omega(\mathbf{H})}} \tag{22}$$

The structural similarity relation is closed in the domain of box expressions[29] and preserves the denotational semantics.

[28] Intuitively, $\mathbf{F}$ corresponds to the common (and thus unchanging) part of $\mathbf{D}$ and $\mathbf{H}$.

[29] That is, whenever a box expression can match one side of a rule, then it is guaranteed that the other side is a box expression too. Thus the rules can be thought of as well formed term rewriting rules. Similar comment applies to the rules of the structural operational semantics.

Theorem 30. *If D is a box expression and $D \equiv H$ then H is also a box expression and* $\mathsf{box}(D) = \mathsf{box}(H)$ □

Thus $\equiv$ is a sound equivalence notion in the domain of box expressions. It is also complete in the sense that $\mathsf{box}(D) = \mathsf{box}(H)$ implies $D \equiv H$ provided that $\lfloor D \rfloor \equiv \lfloor H \rfloor$.

Example (continued) The DIY algebra gives rise to five specific rules for the structural equivalence relation (we omit here their symmetric, hence redundant, counterparts). The first two are derived from (19): $\overline{\underline{c_2}} \equiv c_{21}$ and $\underline{\underline{c_2}} \equiv c_{23}$. The third and fourth are derived from (20): $\mathsf{op}_{\Omega_0}(\overline{E}, \overline{F}, G) \equiv \overline{\mathsf{op}_{\Omega_0}(E, F, G)}$ and $\mathsf{op}_{\Omega_0}(E, F, \underline{G}) \equiv \underline{\mathsf{op}_{\Omega_0}(E, F, G)}$. Finally, there is a single instance of (21): $\mathsf{op}_{\Omega_0}(\underline{E}, \underline{F}, G) \equiv \mathsf{op}_{\Omega_0}(E, F, \overline{G})$. An application of these rules is illustrated below:

$$\mathsf{op}_{\Omega_0}(\underline{c_1}, c_{23}, c_3) \equiv \mathsf{op}_{\Omega_0}(c_1, c_2, \overline{c_3})$$

$$\overbrace{\mathsf{op}_{\Omega_0}(\underline{c_1}, c_{23}, c_3) \equiv \mathsf{op}_{\Omega_0}(\underline{c_1}, \underline{c_2}, c_3) \qquad \mathsf{op}(\underline{c_1}, \underline{c_2}, c_3) \equiv \mathsf{op}_{\Omega_0}(c_1, c_2, \overline{c_3})}$$

$$\overbrace{\underline{c_1} \equiv \underline{c_1} \qquad c_{23} \equiv \underline{c_2} \qquad c_3 \equiv c_3}$$

Label based operational semantics The first derivation system has moves of the form $D \xrightarrow{\Gamma} H$, where D and H are expressions and Γ is a finite multiset of labels. Formally, we define a ternary relation $\longrightarrow$ which is the least relation comprising all triples $D \xrightarrow{\Gamma} H$, where D and H are box expressions and $\Gamma \in \mathsf{mult}(\mathsf{Lab} \cup \{\mathsf{skip}, \mathsf{redo}\})$, and the following hold.

- For every static expression E:

$$\boxed{\overline{E} \xrightarrow{\{\mathsf{skip}\}} \underline{E} \qquad \underline{E} \xrightarrow{\{\mathsf{redo}\}} \overline{E}} \tag{23}$$

- For all box expressions D, J and H:

$$\boxed{\frac{D \equiv H}{D \xrightarrow{\emptyset} H} \qquad \frac{D \xrightarrow{\emptyset} J \xrightarrow{\Gamma} H}{D \xrightarrow{\Gamma} H} \qquad \frac{D \xrightarrow{\Gamma} J \xrightarrow{\emptyset} H}{D \xrightarrow{\Gamma} H}} \tag{24}$$

- For every flat expression D and a non-empty step of transitions U enabled at $M_{\mathsf{box}(D)}$, there is an expression H such that $\mathsf{box}(D)\ [U\rangle\ \mathsf{box}(H)$, $\lambda_{\mathsf{box}(D)}(U) = \Gamma$ and:

$$\boxed{D \xrightarrow{\Gamma} H}$$

- For every operator box Ω, and all Ω-tuples $\mathbf{D}$ and $\mathbf{H}$ of expressions:

$$\boxed{\frac{\forall v \in T_\Omega\colon D_v \xrightarrow{\Gamma_v^1 + \cdots + \Gamma_v^{k_v}} H_v}{\mathsf{op}_\Omega(\mathbf{D}) \xrightarrow{\sum_{v \in T_\Omega} \{\alpha_v^1, \ldots, \alpha_v^{k_v}\}} \mathsf{op}_\Omega(\mathbf{H})} \quad (\Gamma_v^i, \alpha_v^i) \in \lambda_\Omega(v)} \tag{25}$$

The only way to generate a skip or redo is through applying the rules (23), and possibly the rules (24) afterwards; thus, for example, if $D \xrightarrow{\Gamma} H$ and $\mathsf{skip} \in \Gamma$ then $\Gamma = \{\mathsf{skip}\}$. Moreover, $D \xrightarrow{\emptyset} H$ if and only if $D \equiv H$.

A fundamental property of the operational semantics is that it transforms box expressions into box expressions, and the moves generated are all valid steps for the corresponding boxes.

Theorem 31. *Let D be a box expression. If $D \xrightarrow{\Gamma} H$ then H is a box expression such that $\mathsf{box}(D)_{\mathsf{sr}}\ [\Gamma\rangle_{\mathsf{lab}}\ \mathsf{box}(H)_{\mathsf{sr}}$. Conversely, if $\mathsf{box}(D)_{\mathsf{sr}}\ [\Gamma\rangle\ \Sigma_{\mathsf{sr}}$ then there is a box expression H such that $\mathsf{box}(H) = \Sigma$ and $D \xrightarrow{\Gamma} H$.* □

Example (continued) In the DIY algebra, the operational semantics of flat expressions is given by:

$$\overline{c_1} \xrightarrow{\{a\}} c_{12} \qquad \overline{c_2} \xrightarrow{\{c\}} c_{22} \qquad \overline{c_1} \xrightarrow{\{b\}} c_{11} \qquad c_{21} \xrightarrow{\{c\}} c_{22} \qquad \overline{c_1} \xrightarrow{\{a,b\}} \underline{c_1}$$
$$c_{22} \xrightarrow{\{d\}} \underline{c_2} \qquad c_{12} \xrightarrow{\{b\}} \underline{c_1} \qquad c_{22} \xrightarrow{\{d\}} c_{23} \qquad c_{11} \xrightarrow{\{a\}} \underline{c_1} \qquad \overline{c_3} \xrightarrow{\{e\}} \underline{c_3}$$

and the inference rule for the only operator box can be formulated in the following way:

$$\frac{D_1 \xrightarrow{k\cdot\{a,b\}+\Gamma_1} H_1 \quad , \quad D_2 \xrightarrow{\Gamma_2} H_2 \quad , \quad D_3 \xrightarrow{\Gamma_3} H_3}{\mathsf{op}_{\Omega_0}(D_1, D_2, D_3) \xrightarrow{k\cdot\{f\}+\Gamma_1+\Gamma_2+\Gamma_3} \mathsf{op}_{\Omega_0}(H_1, H_2, H_3)} \quad a, b \notin \Gamma_1$$

An application of these rules is shown below:

$$\frac{\overline{\mathsf{op}_{\Omega_0}(c_1, c_2, c_3)} \xrightarrow{\emptyset} \mathsf{op}_{\Omega_0}(\overline{c_1}, \overline{c_2}, c_3) \qquad \mathsf{op}_{\Omega_0}(\overline{c_1}, \overline{c_2}, c_3) \xrightarrow{\{c,f\}} \mathsf{op}_{\Omega_0}(\underline{c_1}, c_{22}, c_3)}{\overline{\mathsf{op}_{\Omega_0}(c_1, c_2, c_3)} \xrightarrow{\{c,f\}} \mathsf{op}_{\Omega_0}(\underline{c_1}, c_{22}, c_3)}$$

$$\overline{c_1} \xrightarrow{\{a,b\}} \underline{c_1} \qquad \overline{c_2} \xrightarrow{\{c\}} c_{22} \qquad c_3 \xrightarrow{\emptyset} c_3$$

The upper step is an application of (24), while the lower step is an application of the preceding inference rule (and hence of (25)).

Transition based operational semantics Consider the set $\mathsf{T}^{alg}_{\mathsf{tree}}$ of all transition trees in the boxes derived through the box mapping. It is easy to see that each $t \in \mathsf{T}^{alg}_{\mathsf{tree}}$ has a unique label, $\mathsf{lab}(t)$, in all the boxes associated with box expressions in which it occurs.

The operational semantics we will now define has moves of the form $D \xrightarrow{U} H$ such that D and H are box expression and U is a finite subset of $\mathsf{T}^{alg}_{\mathsf{tree}} \cup \{\mathsf{skip}, \mathsf{redo}\}$. The idea here is that U is a valid step for the boxes associated with D and H, after augmenting them with the skip and redo transitions. We will denote, for every such set,

$$\mathsf{lab}(U) \ = \ \sum_{t \in U} \{\mathsf{lab}(t)\}$$

assuming that $\mathsf{lab}(\mathsf{skip}) = \mathsf{skip}$ and $\mathsf{lab}(\mathsf{redo}) = \mathsf{redo}$.

In the process of defining a transition based operational semantics, we adopt without any change the structural similarity relation on expressions, $\equiv$. The rules of the operational semantics are also carried forward after changing Γ to U consistently, with the exception of the last rule (25) which is redefined thus:

$$\frac{\forall\, v \in T_\Omega \;:\; D_v \xrightarrow{U_v^1 \bullet \cdots \bullet U_v^{k_v}} H_v \qquad (\mathsf{lab}(U_v^i), \alpha_v^i) \in \lambda_\Omega(v)}{\mathsf{op}_\Omega(\mathbf{D}) \xrightarrow{U} \mathsf{op}_\Omega(\mathbf{H})}$$

$$\text{where} \quad U \;=\; \bigcup_{v \in T_\Omega} \{(v, \alpha_v^1) \lhd U_v^1 \ldots, (v, \alpha_v^{k_v}) \lhd U_v^{k_v}\}$$

Example (continued) In the DIY algebra, the transition based operational semantics of flat expressions is given by the axioms:

$$\overline{c_1} \xrightarrow{\{t_{11}\}} c_{12} \qquad \overline{c_2} \xrightarrow{\{t_{21}\}} c_{22} \qquad \overline{c_1} \xrightarrow{\{t_{12}\}} c_{11} \qquad c_{21} \xrightarrow{\{t_{21}\}} c_{22} \qquad \overline{c_1} \xrightarrow{\{t_{11}, t_{12}\}} \underline{c_1}$$

$$c_{22} \xrightarrow{\{t_{22}\}} \underline{c_2} \qquad c_{12} \xrightarrow{\{t_{12}\}} \underline{c_1} \qquad c_{22} \xrightarrow{\{t_{22}\}} c_{23} \qquad c_{11} \xrightarrow{\{t_{11}\}} \underline{c_1} \qquad \overline{c_3} \xrightarrow{\{t_{31}\}} \underline{c_3}$$

and the inference rule for the only operator box is:

$$\frac{D \xrightarrow{\{t_1, u_1, \ldots, t_k, u_k, x_1, \ldots, x_l\}} D' \quad , \quad H \xrightarrow{\{y_1, \ldots, y_m\}} H' \quad , \quad G \xrightarrow{\{z_1, \ldots, z_n\}} G'}{\mathsf{op}_{\Omega_0}(D, G, H) \xrightarrow{U} \mathsf{op}_{\Omega_0}(D', G', H')}$$

where $k, l, m, n \geq 0$; $\{\mathsf{lab}(t_i), \mathsf{lab}(u_i)\} = \{a, b\}$, for every $i \leq k$; $\mathsf{lab}(x_i) \notin \{a, b\}$, for every $i \leq l$; and the step U is given by:

$$\begin{aligned} U = \{\ & (v_1, f) \lhd \{t_1, u_1\} \ , \ \ldots \ , \ (v_1, f) \lhd \{t_k, u_k\} \ \} \cup \\ \{\ & (v_1, \mathsf{lab}(x_1)) \lhd x_1 \ , \ \ldots \ , \ (v_1, \mathsf{lab}(x_l)) \lhd x_l \ \} \cup \\ \{\ & (v_2, \mathsf{lab}(y_1)) \lhd y_1 \ , \ \ldots \ , (v_2, \mathsf{lab}(y_m)) \lhd y_m \ \} \cup \\ \{\ & (v_3, \mathsf{lab}(z_1)) \lhd z_1 \ , \ \ldots \ , \ (v_3, \mathsf{lab}(z_n)) \lhd z_n \ \} \end{aligned}$$

For example, the following is a valid sequence of three moves:

$$\overline{\mathsf{op}_{\Omega_0}(c_1, c_2, c_3)} \xrightarrow{\{w_1, w_2\}} \mathsf{op}_{\Omega_0}(\underline{c_1}, c_{22}, c_3) \xrightarrow{\{w_3\}} \mathsf{op}_{\Omega_0}(c_1, c_2, \overline{c_3}) \xrightarrow{\{w_4\}} \underline{\mathsf{op}_{\Omega_0}(c_1, c_2, c_3)} \ .$$

A derivation tree for the first move is shown below:

$$\cfrac{\overbrace{\overline{\mathsf{op}_{\Omega_0}(c_1, c_2, c_3)} \xrightarrow{\{w_1, w_2\}} \mathsf{op}_{\Omega_0}(\underline{c_1}, c_{22}, c_3)}}{\overline{\mathsf{op}_{\Omega_0}(c_1, c_2, c_3)} \xrightarrow{\emptyset} \mathsf{op}_{\Omega_0}(\overline{c_1}, \overline{c_2}, c_3)}$$

$$\mathsf{op}_{\Omega_0}(\overline{c_1}, \overline{c_2}, c_3) \xrightarrow{\{\ (v_1, f) \lhd \{t_{11}, t_{12}\} \ , \ (v_2, c) \lhd t_{21}\ \}} \mathsf{op}_{\Omega_0}(\underline{c_1}, c_{22}, c_3)$$

$$\overline{c_1} \xrightarrow{\{t_{11}, t_{12}\}} \underline{c_1} \qquad \overline{c_2} \xrightarrow{\{t_{21}\}} c_{22} \qquad c_3 \xrightarrow{\emptyset} c_3$$

8.3 Partial order semantics of box expressions

The structure of name trees in $\mathsf{T}^{alg}_{\mathsf{tree}}$ turns out to be a rich source of information about the connectivity between transitions in nets generated from box expressions. What is more, this information is independent of the specific net to which two transitions happen to belong.

Connectedness properties of transition trees It is possible to capture the property of 'being connected to an entry or exit place'. A transition tree $t \in \mathsf{T}^{alg}_{\mathsf{tree}}$ belongs to the set Con^e or Con^x, if one of the following holds: either (i) there is a plain box $\Sigma \in \mathsf{ConstBox} \cap \mathsf{Box}^s$ such that $t \in T_\Sigma$ and we have, respectively, $({}^\bullet t \cup t^\bullet) \cap {}^\circ\Sigma \neq \emptyset$ or $({}^\bullet t \cup t^\bullet) \cap \Sigma^\circ \neq \emptyset$; or (ii) there is an operator box $\Omega \in \mathsf{OpBox}$ such that $t = (v, \alpha) \lhd Q$ and $v \in T_\Omega$, and we have[30], respectively:

$$ {}^\bullet v \cap {}^\circ\Omega \neq \emptyset \neq Q \cap Con^e \ \vee \ v^\bullet \cap {}^\circ\Omega \neq \emptyset \neq Q \cap Con^x \quad \text{or} $$
$$ {}^\bullet v \cap \Omega^\circ \neq \emptyset \neq Q \cap Con^e \ \vee \ v^\bullet \cap \Omega^\circ \neq \emptyset \neq Q \cap Con^x \ . $$

One can then show that if D is a box expression then ${}^\bullet({}^\circ\mathsf{box}(D)) \cup ({}^\circ\mathsf{box}(D))^\bullet = T_{\mathsf{box}(D)} \cap Con^e$ and ${}^\bullet(\mathsf{box}(D)^\circ) \cup (\mathsf{box}(D)^\circ)^\bullet = T_{\mathsf{box}(D)} \cap Con^x$.

Global independence relation The trees in $\mathsf{T}^{alg}_{\mathsf{tree}}$ also contain enough information to reconstruct the independence relation on transitions in nets generated by box expression.

A pair of transition trees $(t, u) \in \mathsf{T}^{alg}_{\mathsf{tree}} \times \mathsf{T}^{alg}_{\mathsf{tree}}$ belongs to the set ind^{alg} if one of the following holds: either (i) there is a plain box $\Sigma \in \mathsf{ConstBox} \cap \mathsf{Box}^s$ such that $(t, u) \in \mathsf{ind}_\Sigma$; or (ii) there is an operator box $\Omega \in \mathsf{OpBox}$ such that $t = (v, \alpha) \lhd Q$ and $u = (w, \beta) \lhd R$ and $v, w \in T_\Omega$, and we have:

$$ {}^\bullet v \cap w^\bullet \neq \emptyset \Longrightarrow Q \cap Con^e = \emptyset \vee R \cap Con^x = \emptyset $$
$$ v^\bullet \cap {}^\bullet w \neq \emptyset \Longrightarrow Q \cap Con^x = \emptyset \vee R \cap Con^e = \emptyset $$
$$ v \neq w \wedge {}^\bullet v \cap {}^\bullet w \neq \emptyset \Longrightarrow Q \cap Con^e = \emptyset \vee R \cap Con^e = \emptyset $$
$$ v \neq w \wedge v^\bullet \cap w^\bullet \neq \emptyset \Longrightarrow Q \cap Con^x = \emptyset \vee R \cap Con^x = \emptyset \ , $$

and, moreover, if $v = w$ then $Q \times R \subseteq \mathsf{ind}^{alg}$.

Intuitively, ind^{alg} can be thought of as a *global independence relation* since, for every box expression D, $\mathsf{ind}_{\mathsf{box}(D)} = (T_{\mathsf{box}(D)} \times T_{\mathsf{box}(D)}) \cap \mathsf{ind}^{alg}$. That this property holds is due to the assumption made in (16) that the operator and static constant boxes have disjoint transition sets. As a result, one can identify, for every transition t in $\mathsf{OpBox} \cup (\mathsf{ConstBox} \cap \mathsf{Box}^s)$, a unique net (operator or static constant box) to which t belongs.

Partial orders generated by box expressions Having defined the global independence relation, we can introduce partial order behaviours modelled by Mazurkiewicz traces directly into the box algebra.

[30] Hence, the definition of the sets Con^e and Con^e (as well as that of ind^{alg} below) is recursive, but since transition trees are finite this is not a problem.

Let D and H be two box expressions, and $U_1, \ldots, U_k$ $(k \geq 0)$ be subsets of $\mathsf{T}^{alg}_{\mathsf{tree}}$ such that there are box expressions $D_0, D_1, \ldots, D_k$ satisfying $D = D_0$, $H = D_k$ and, for every $i < k$,

$$D_i \stackrel{U_{i+1}}{\longrightarrow} D_{i+1} \ .$$

Then $\pi = \mathsf{poset}_{\mathsf{ind}^{alg}}(U_1 \ldots U_k)$ is a *partial order execution* for D, denoted $D \stackrel{\pi}{\longrightarrow} H$.

8.4 Consistency results

The consistency between the denotational and operational step sequence semantics of box expressions is expressed through the transition systems they generate.

Let G be a dynamic expression. We will use $[G\rangle_{\mathsf{sr}}$ to denote the least set of expressions containing G such that if $H \in [G\rangle_{\mathsf{sr}}$ and $H \stackrel{U}{\longrightarrow} C$, for some finite subset U of $\mathsf{T}^{alg}_{\mathsf{tree}} \cup \{\mathsf{skip}, \mathsf{redo}\}$, then $C \in [G\rangle_{\mathsf{sr}}$. The *full transition system* of G is the transition system generated by G up to structural similarity. Formally, it is defined as $\mathsf{fts}_G = (V, L, A, v_0)$ where: $V = \{[H]_{\equiv} \mid H \in [G\rangle_{\mathsf{sr}}\}$ is the set of states; $v_0 = [G]_{\equiv}$ is the initial state; L – the set of all finite subsets of $\mathsf{T}^{alg}_{\mathsf{tree}} \cup \{\mathsf{skip}, \mathsf{redo}\}$ – is the set of labels; and

$$A = \{([H]_{\equiv}, U, [C]_{\equiv}) \in V \times L \times V \mid H \stackrel{U}{\longrightarrow} C\}$$

is the set of arcs. For a static expression E, $\mathsf{fts}_E = \mathsf{fts}_{\overline{E}}$.

The label based operational semantics of a box expression D is captured by the *transition system* of D, denoted by ts_D and defined in exactly the same way as fts_D, with each transition based move $H \stackrel{U}{\longrightarrow} C$ replaced by the corresponding label based move $H \stackrel{\Gamma}{\longrightarrow} C$.

We now state two fundamental results which effectively state that the operational and denotational semantics of a box expression capture exactly the same behaviour.

Theorem 32. *For every box expression D, the relation*

$$\mathsf{iso} \ = \ \{([H]_{\equiv}, \mathsf{box}(H)) \mid [H]_{\equiv} \textit{ is a node of } \mathsf{fts}_D\}$$

is an isomorphism between fts_D *and* $\mathsf{fts}_{\mathsf{box}(D)}$ *as well as between* ts_D *and* $\mathsf{ts}_{\mathsf{box}(D)}$.

Theorem 33. *Let D be a box expression, and π be a partial order.*

(1) If $D \stackrel{\pi}{\longrightarrow} H$ *then* $\mathsf{box}(D)\ [\pi\rangle_{\mathsf{po}}\ \mathsf{box}(H)$.
(2) If $\mathsf{box}(D)\ [\pi\rangle_{\mathsf{po}}\ \Sigma$ *then there is* H *such that* $\mathsf{box}(H) = \Sigma$ *and* $D \stackrel{\pi}{\longrightarrow} H$. □

Example (continued) In the DIY algebra, theorem 32 can be illustrated by taking the expression $D = \mathsf{op}_{\Omega_0}(\overline{c_1}, c_{22}, c_3)$ and the corresponding box shown in figure 24. Figure 26

depicts fts_D and $\mathsf{fts}_{\mathsf{box}(D)}$, where

$$\begin{aligned}
\xi_0 &= \{\mathsf{op}_{\Omega_0}(\overline{c_1}, c_{22}, c_3)\} & M_{\Sigma_0} &= \{p_1, p_2, p_6\}\\
\xi_1 &= \{\overline{\mathsf{op}_{\Omega_0}(c_1, c_2, c_3)}, \mathsf{op}_{\Omega_0}(\overline{c_1}, \overline{c_2}, c_3), \mathsf{op}_{\Omega_0}(\overline{c_1}, c_{21}, c_3)\} & M_{\Sigma_1} &= \{p_1, p_2, p_5\}\\
\xi_2 &= \{\mathsf{op}_{\Omega_0}(\underline{c_1}, \overline{c_2}, c_3), \mathsf{op}_{\Omega_0}(\underline{c_1}, c_{21}, c_3)\} & M_{\Sigma_2} &= \{p_3, p_4, p_5\}\\
\xi_3 &= \{\mathsf{op}_{\Omega_0}(\underline{c_1}, c_{22}, c_3)\} & M_{\Sigma_3} &= \{p_3, p_4, p_6\}\\
\xi_4 &= \{\mathsf{op}_{\Omega_0}(\overline{c_1}, c_{23}, c_3), \mathsf{op}_{\Omega_0}(\overline{c_1}, \underline{c_2}, c_3)\} & M_{\Sigma_4} &= \{p_1, p_2, p_7\}\\
\xi_5 &= \{\mathsf{op}_{\Omega_0}(\underline{c_1}, \underline{c_2}, c_3), \mathsf{op}_{\Omega_0}(\underline{c_1}, c_{23}, c_3), \mathsf{op}_{\Omega_0}(c_1, c_2, \overline{c_3})\} & M_{\Sigma_5} &= \{p_3, p_4, p_7\}\\
\xi_6 &= \{\underline{\mathsf{op}_{\Omega_0}(c_1, c_2, c_3)}, \mathsf{op}_{\Omega_0}(c_1, c_2, \underline{c_3})\} & M_{\Sigma_6} &= \{p_8\}\ .
\end{aligned}$$

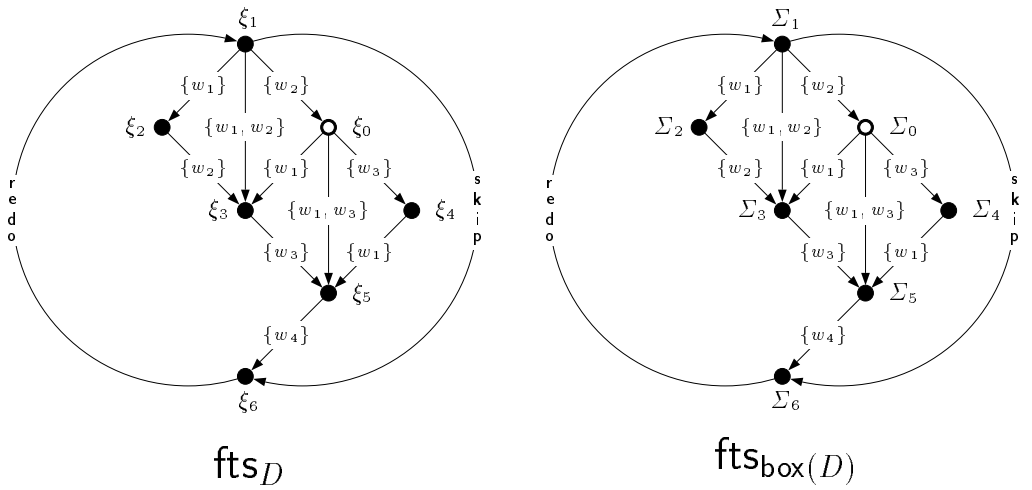


Fig. 26. Isomorphic box transition systems.

Factorisability revisited We now briefly return to the relevance of the factorisability property. Consider the non-factorisable operator box N shown on the right in figure 23. To see that this operator does not have a complete operational semantics in the above sense, try defining a quaternary operator $\mathsf{op}_N(E_1, E_2, E_3, E_4)$ from it, with four arguments (where E_1 – E_4 are supposed to describe the same flow of control as transitions v_1 – v_4 of the operator

box) and the following set of equations, obtained from the generic rules in section 8.2:

$$\overline{\mathsf{op}_N(E_1,E_2,E_3,E_4)} \equiv \mathsf{op}_N(\overline{E_1},\overline{E_2},E_3,E_4)$$
$$\mathsf{op}_N(\underline{E_1},\overline{E_2},E_3,E_4) \equiv \mathsf{op}_N(E_1,\overline{E_2},\overline{E_3},E_4)$$
$$\mathsf{op}_N(\underline{E_1},\underline{E_2},E_3,E_4) \equiv \mathsf{op}_N(E_1,E_2,\overline{E_3},\overline{E_4})$$
$$\mathsf{op}_N(E_1,\underline{E_2},\underline{E_3},E_4) \equiv \mathsf{op}_N(E_1,E_2,\underline{E_3},\overline{E_4})$$
$$\mathsf{op}_N(E_1,E_2,\underline{E_3},\underline{E_4}) \equiv \underline{\mathsf{op}_N(E_1,E_2,E_3,E_4)}\ .$$

At first sight, the equations seem to describe the control flow within the operator box. However, if we consider $E = \mathsf{op}_N(a,(b;b'),(c;c'),d)$ then the box corresponding to the expression $\overline{E}$ can execute the sequence of labels, $abcb'd$, but this cannot be derived from the generic labelled evolution rules using the above set of equations (essentially, the third equation can never be applied in this case).

8.5 Equivalence notions

In section 3.4, we defined two behavioural equivalences between static and dynamic boxes based on the transition systems they generate. These can be lifted in the usual way to the level of static and dynamic box expressions, and then one can show that they are both *congruences* for the box algebra.

9 Examples of Box Algebras

In this section, we formalise PBC using the generic framework of the box algebra, and then argue that other process algebra models can be viewed as instances of the general scheme too.

9.1 PBC

The version of PBC we are going to present is based on eight operator boxes and two kinds of constants, depicted in figure 27. The operator boxes are: $\Omega_{[f]}$ (basic relabelling), $\Omega_{\mathsf{sy}\,A}$ (synchronisation), $\Omega_{\mathsf{rs}\,A}$ (restriction), $\Omega_{[A:]}$ (scoping), $\Omega_{;}$ (sequence), $\Omega_{\square}$ (choice), $\Omega_{\|}$ (parallel composition) and Ω_{*} (iteration), where $f : \mathsf{A}_{\mathrm{PBC}} \to \mathsf{A}_{\mathrm{PBC}}$ is a function and $A \subseteq \mathsf{A}_{\mathrm{PBC}}$ is a set of action particles.[31] All eight operator boxes are sos-operator boxes which are x-directed and have all transitions reversible. Moreover, all except iteration are also pure and e-directed. The specific relabellings used are as follows: (i) $\varrho_{\mathsf{sy}\,A}$ is the smallest relabelling which contains ϱ_{id} and such that, if $(\Gamma,\alpha+\{\{a\}\}) \in \varrho_{\mathsf{sy}\,A}$ and $(\Delta,\beta+\{\{\widehat{a}\}\}) \in \varrho_{\mathsf{sy}\,A}$ where $a \in A \cup \widehat{A}$, then $(\Gamma+\Delta,\alpha+\beta) \in \varrho_{\mathsf{sy}\,A}$, as well; (ii) $\varrho_{[A:]}$ comprises all the pairs $(\Gamma,\alpha) \in \varrho_{\mathsf{sy}\,A}$ such that $\alpha \cap (A \cup \widehat{A}) = \emptyset$; (iii) $\varrho_{\mathsf{rs}\,A}$ comprises all the pairs $(\{\alpha\},\alpha) \in \varrho_{id}$ such that $\alpha \cap (A \cup \widehat{A}) = \emptyset$; and (iv) $\varrho_{[f]}$ comprises all the pairs $(\{\alpha\},f(\alpha))$ where f is applied to α element-wise.

The static PBC constants (all ex-directed and ex-exclusive) are: N_{α} for a multiaction $\alpha \in \mathsf{Lab}_{\mathrm{PBC}}$, and $\mathsf{N}_{[x_{()}]}$ for a data variable x (not a recursion variable). The latter are more complicated and deserve an additional explanation.

[31] See section 2 for the definition of $\mathsf{A}_{\mathrm{PBC}}$ and other related notions.

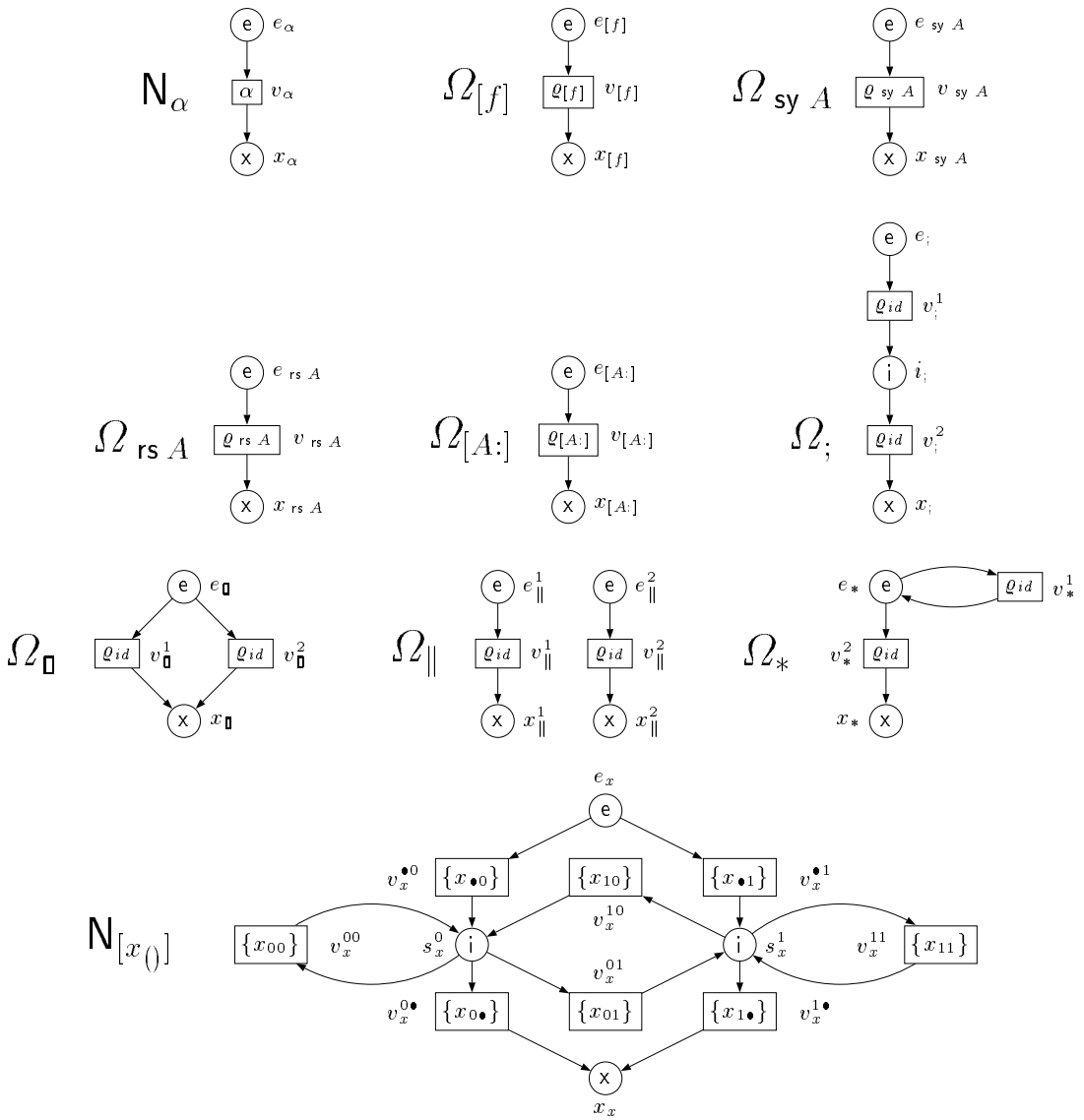


Fig. 27. PBC operator boxes and constants.

Data variables Let x be a data variable of type D, and $\bullet$ be a special symbol not in D, interpreted as an 'undefined value'. For such a variable, the set of action particles $\mathsf{A}_{\mathrm{PBC}}$ contains the symbols x_{kl} and $\widehat{x_{kl}}$ for all $k, l \in D \cup \{\bullet\}$; moreover, x_{kl} and $\widehat{x_{kl}}$ are conjugate action particles.

The behaviour of x will be represented by a plain box $\mathsf{N}_{[x_{()}]}$. Intuitively, it is composed of an initialisation part, a core looping part and a termination part. Such a box can first execute $x_{\bullet u}$, for any value u in D, then a sequence of zero or more executions of x_{uv}, where $u, v \in D$, and possibly terminate by executing $x_{u\bullet}$. Each such execution is carried out under a restriction that, if x_{ku} and x_{vl} are two consecutively performed actions, then $u = v$.

We use x_{uv} to represent the change of the value of x from u to v, $x_{\bullet u}$ to represent an initialisation of x to the value u, and $x_{u\bullet}$ to represent the 'destruction' of x when its current value is u. The net model of a data variable is exemplified in figure 27 for a binary x, i.e., $D = \{0, 1\}$. In general, we define $\mathsf{N}_{[x_{()}]} = (S_x, T_x, W_x, \lambda_x, \emptyset)$ where:

$$S_x \;=\; \{e_x, x_x\} \cup \{s_x^u \mid u \in D\} \quad \text{and} \quad T_x \;=\; \{v_x^{\bullet u}, v_x^{u\bullet} \mid u \in D\} \cup \{v_x^{uv} \mid u, v \in D\}\ ,$$

the weight function W_x returns 1, for all the pairs in the set

$$\{\ (e_x, v_x^{\bullet u})\ ,\ (v_x^{\bullet u}, s_x^u)\ ,\ (s_x^u, v_x^{uv})\ ,\ (v_x^{uv}, s_x^v)\ ,\ (s_x^u, v_x^{u\bullet})\ ,\ (v_x^{u\bullet}, x_x)\ \mid\ u, v \in D\}$$

and 0 otherwise; and the label function is defined, for all the values $u, v \in D$, by:

$$\begin{array}{lll} \lambda_x(e_x) = \mathsf{e} & \lambda_x(x_x) = \mathsf{x} & \lambda_x(s_x^u) = \mathsf{i} \\ \lambda_x(v_x^{\bullet u}) = \{x_{\bullet u}\} & \lambda_x(v_x^{u\bullet}) = \{x_{u\bullet}\} & \lambda_x(v_x^{uv}) = \{x_{uv}\}\ . \end{array}$$

There is also a set of dynamic constant boxes $\mathsf{N}_{[\overline{x_{(u)}}]}$, for every $u \in D$, defined in the same way as $\mathsf{N}_{[x_{()}]}$, but with a single token in the place s_x^u.

PBC syntax I The following is the syntax (17) for static box expressions generated by the operator boxes and constants in figure 27.

$$\begin{array}{rcl} E & ::= & X \mid \alpha \mid [x_{()}] \mid E \| E \mid E \,\square\, E \mid E; E \mid \langle E * E] \mid \\ & & E[f] \mid E\ \mathsf{sy}\ A \mid E\ \mathsf{rs}\ A \mid [A : E]\ . \end{array} \tag{26}$$

Notice that we have replaced the functional style of denoting box expressions by more common infix and postfix notations. The symbols used should make obvious the association between the boxes in figure 9.1 and the PBC term operators.

The *dynamic PBC expressions* are defined below, where E denotes a static PBC expression given by (26):

$$\begin{array}{rcl} G & ::= & \overline{E} \mid \underline{E} \mid [\overline{x_{(u)}}] \mid G \| G \mid G \,\square\, E \mid E \,\square\, G \mid G; E \mid \\ & & E; G \mid \langle G * E] \mid \langle E * G] \mid G[f] \mid G\ \mathsf{sy}\ A \mid G\ \mathsf{rs}\ a \mid [A : G]\ . \end{array} \tag{27}$$

The syntaxes (26) and (27) do not yet incorporate the condition (Dom1); the other condition, (Dom2), is trivially satisfied since the operator boxes in figure 27 have only reversible transitions.

PBC syntax II The refinement of the syntax (26) – shown below – incorporates (Dom1). It is assumed that the variables $\mathcal{X}^{wf}$ and $\mathcal{X}^{xcl}$ are defined by $\mathcal{X}^{wf} \stackrel{\text{df}}{=} E^{wf}$ and $\mathcal{X}^{xcl} \stackrel{\text{df}}{=} E^{xcl}$, respectively.

$$\begin{array}{lllllllllll} E^{wf} & ::= & \mathcal{X}^{wf} & | & E^{wf}[f] & | & E^{wf} \,\square\, E^{wf} & | & \langle E^{xcl} * E^{wf} \rangle & | & E^{wf} \| E^{wf} & | & E^{xcl} \\ E^{xcl} & ::= & \mathcal{X}^{xcl} & | & E^{xcl}[f] & | & E^{xcl} \,\square\, E^{xcl} & | & \langle E^{xcl} * E^{xcl} \rangle & | & E^{wf}; E^{wf} & | & \alpha\ . \end{array} \tag{28}$$

Expressions defined by E^{wf} and E^{xcl} are the well formed and ex-exclusive PBC expressions, respectively. Moreover, since non-trivial aspects of non-ex-exclusiveness and non-ex-directedness are associated with control flow rather than communication interface operators, we may treat the latter as equivalent; hence we shortened the syntax by arbitrarily selecting only one communication interface operator, viz. basic relabelling. The same comment applies to the constant boxes; since in PBC all these are both ex-exclusive and ex-directed, we included just the basic action. Similar, rather immediate, modifications can be incorporated into the syntax of dynamic PBC expressions.

The refined PBC syntax (28) is an instance of the general box algebra and, as a consequence, *all results formulated in the sections 7 and 8 hold for it.*

PBC syntax III The syntax (28) still admits expressions for which the corresponding boxes may not satisfy the conditions (Beh1-5). We therefore further refine it by introducing a third syntax which identifies a subset of expressions generated by (28) where these conditions have been incorporated. The syntax is based on four syntactic classes E^{β}, and it is assumed that each variable X^{β} is defined by $X^{\beta} \stackrel{\text{df}}{=} E^{\beta}$.

$$\begin{array}{lllllllllllll} E^{wf} & ::= & \mathcal{X}^{wf} & | & E^{wf}[f] & | & E^{wf} \| E^{wf} & | & \langle E^{edir:xcl} * E^{edir} \rangle & | & E^{xcl} & | & E^{edir} \\ E^{edir} & ::= & \mathcal{X}^{edir} & | & E^{edir}[f] & | & E^{edir} \| E^{edir} & | & E^{edir} \,\square\, E^{edir} & | & E^{edir:xcl} & & \\ E^{xcl} & ::= & \mathcal{X}^{xcl} & | & E^{xcl}[f] & | & E^{wf}; E^{wf} & | & \langle E^{edir:xcl} * E^{edir:xcl} \rangle & | & E^{edir:xcl} & & \\ E^{edir:xcl} & ::= & \mathcal{X}^{edir:xcl} & | & E^{edir:xcl}[f] & | & E^{edir}; E^{wf} & | & E^{edir:xcl} \,\square\, E^{edir:xcl} & | & \alpha\ . & & \end{array}$$

The above syntax conforms to the general box algebra scheme,[32] and all expressions it specifies satisfy the five conditions, (Beh1-5). Referring to the notation used in section 8.1, the annotations *wf*, *e* and *ecl* indicate that the respective syntactic entities define expressions which are respectively well formed, e-directed and ex-exclusive. Notice that all expressions are also x-directed since so are the PBC constants and operator boxes in figure 27.

Other PBC operators and constants The version of the PBC presented here is not the only one possible. Indeed, the original definition of PBC (as well as that used in several subsequent papers) did not include the binary iteration operator $\langle E * F \rangle$; instead, a ternary iteration $[E' * E * F]$ was used, with E' representing the initialisation, E the main looping part, and F the exit from the loop. Clearly, such a construct is a derived operator in the current setting, $[E' * E * F]$ being equivalent to $E'; \langle E * F \rangle$. The main benefit for the formal

[32] Again, we omitted a straightforward adaptation of the syntax for the dynamic PBC expressions.

treatment was that the operator box modelling the ternary iteration was ex-directed and pure, which made the conditions (Dom1-2) and (Beh1-5) automatically fulfilled.

Other PBC constructs not included in the present syntax are infinite choice, infinite parallel composition and infinite sequence. As to the constant (plain) boxes, more complex programming primitives, such as communication channels, were also defined and used to give a net semantics of concurrent programming languages.

9.2 The two semantics of PBC

The denotational semantics of PBC follows the general pattern, so we omit the details. As to the structural equivalence equations, we provide those specific to the PBC in table 2. The rules of the label based and transition based operational semantics, except those which hold for any box algebra, are shown in tables 3 and 4.

In the tables 3 and 4, the notations $(v,f)\oplus U$ and $v\oplus U$, where v is a transition name, $f:\mathsf{A}_{\mathrm{PBC}}\to\mathsf{A}_{\mathrm{PBC}}$ is a mapping from labels to labels and $U=\{t_1,\ldots,t_k\}$ is a finite set of transition trees, respectively denote the following sets of transition trees:

$$\{(v,f(\mathsf{lab}(t_1)))\lhd t_1,\ldots,(v,f(\mathsf{lab}(t_k)))\lhd t_k\}\quad\text{ and }\quad\{(v,\mathsf{lab}(t_1))\lhd t_1,\ldots,(v,\mathsf{lab}(t_k))\lhd t_k\}\ .$$

Let us verify, for instance, that the rule (25), when instantiated for the sequential composition operator, is equivalent to the two rules SL and SR of table 3. What we need to do is look closer at the syntax of legal PBC expressions and, moreover, bear in mind that static expressions allow only empty moves. Then (25) can be seen as giving rise to two different rules, where G and H are dynamic, while E and F static expressions:

$$\mathrm{RS}_1\qquad\frac{G\xrightarrow{\Gamma}H\ ,\ E\xrightarrow{\emptyset}F}{G;E\xrightarrow{\Gamma}H;F}$$

$$\mathrm{RS}_2\qquad\frac{E\xrightarrow{\emptyset}F\ ,\ G\xrightarrow{\Gamma}H}{E;G\xrightarrow{\Gamma}F;H}$$

And, if we take $F=E$ (which is always possible since $E\xrightarrow{\emptyset}E$), we obtain SR and SL. Conversely, SR and SL together with the rules (22) and (24) as well as the fact that $E\xrightarrow{\emptyset}F$ implies $E\equiv F$, yield RS_1 and RS_2.

9.3 Equivalence notions

As yet we have not discussed any specific algebraic properties of the various box algebra operators with respect to behavioural equivalences. Although we could, in principle, consider these equivalences separately, within the net framework and the process expression framework, in view of the consistency results formulated in section 8.4, we would do the same thing twice, as every result developed for one would hold (almost) *verbatim* for the other. We therefore present a set of properties – or laws, in CSP terminology – for the static

IPAR1	$\overline{E\|\|F} \equiv \overline{E}\|\|\overline{F}$	IPAR2	$\underline{E}\|\|\underline{F} \equiv \underline{E\|\|F}$
IC1L	$\overline{E \,[]\, F} \equiv \overline{E} \,[]\, F$	IC2L	$\underline{E} \,[]\, F \equiv \underline{E \,[]\, F}$
IC1R	$\overline{E \,[]\, F} \equiv E \,[]\, \overline{F}$	IC2R	$E \,[]\, \underline{F} \equiv \underline{E \,[]\, F}$
IS1	$\overline{E;F} \equiv \overline{E};F$	IS2	$\underline{E};F \equiv E;\overline{F}$
IS3	$E;\underline{F} \equiv \underline{E;F}$		
IIT1	$\overline{\langle E * F\rangle} \equiv \langle\overline{E} * F\rangle$	IIT2	$\langle\underline{E} * F\rangle \equiv \langle\overline{E} * F\rangle$
IIT3	$\langle\underline{E} * F\rangle \equiv \langle E * \overline{F}\rangle$	IIT4	$\langle E * \underline{F}\rangle \equiv \underline{\langle E * F\rangle}$
IR1	$\overline{E[f]} \equiv \overline{E}[f]$	IR2	$\underline{E}[f] \equiv \underline{E[f]}$
IRS1	$\overline{E \text{ rs } A} \equiv \overline{E} \text{ rs } A$	IRS2	$\underline{E \text{ rs } A} \equiv \underline{E} \text{ rs } A$
ISY1	$\overline{E \text{ sy } A} \equiv \overline{E} \text{ sy } A$	ISY2	$\underline{E \text{ sy } A} \equiv \underline{E} \text{ sy } A$
ISC1	$\overline{[A:E]} \equiv [A:\overline{E}]$	ISC2	$\underline{[A:E]} \equiv [A:\underline{E}]$

Table 2. PBC structural equivalence rules.

PBC expressions (their extension to the dynamic expressions being immediate). To be more precise, we will say that two static PBC expressions, E and F are *ts-isomorphic*, $E \cong F$, if ts_E and ts_F are isomorphic, and *strongly equivalent*, $E \approx F$, if so are ts_E and ts_F. We now list several fundamental laws for PBC.

- Parallel and choice compositions are associative and commutative, while sequential composition is only associative:

$$\begin{array}{lcllcl} E\|(F\|F') & \cong & (E\|F)\|F' & E\|F & \cong & F\|E \\ E \,[]\, (F \,[]\, F') & \cong & (E \,[]\, F) \,[]\, F' & E \,[]\, F & \cong & F \,[]\, E \\ E;(F;F') & \cong & (E;F);F' & & & \end{array}$$

  Choice is also idempotent, in the following sense:

$$E \,[]\, E \;\approx\; E \qquad \alpha \,[]\, \alpha \;\cong\; \alpha$$

- Synchronisation and restriction are both commutative, idempotent and insensitive to conjugation; they also distribute over choice, sequence, iteration and, in the case of restriction, parallel composition:

$$\begin{array}{lcl} E \text{ sy } A \text{ sy } B & \cong & E \text{ sy } B \text{ sy } A \\ E \text{ sy } A \text{ sy } A & \cong & E \text{ sy } A \\ E \text{ sy } \widehat{A} & \cong & E \text{ sy } A \\ (E \,[]\, F) \text{ sy } A & \cong & (E \text{ sy } A) \,[]\, (F \text{ sy } A) \\ (E;F) \text{ sy } A & \cong & (E \text{ sy } A);(F \text{ sy } A) \\ \langle E * F\rangle \text{ sy } A & \cong & \langle (E \text{ sy } A) * (F \text{ sy } A)\rangle \end{array} \qquad \begin{array}{lcl} E \text{ rs } A \text{ rs } B & \cong & E \text{ rs } B \text{ rs } A \\ E \text{ rs } A \text{ rs } A & \cong & E \text{ rs } A \\ E \text{ rs } \widehat{A} & \cong & E \text{ rs } A \\ (E \,[]\, F) \text{ rs } A & \cong & (E \text{ rs } A) \,[]\, (F \text{ rs } A) \\ (E;F) \text{ rs } A & \cong & (E \text{ rs } A);(F \text{ rs } A) \\ \langle E * F\rangle \text{ rs } A & \cong & \langle (E \text{ rs } A) * (F \text{ rs } A)\rangle \\ (E\|F) \text{ rs } A & \cong & (E \text{ rs } A)\|(F \text{ rs } A) \end{array}$$

AR	$\overline{\alpha} \xrightarrow{\{\alpha\}} \underline{\alpha}$	AR'	$\overline{\alpha} \xrightarrow{\{v_\alpha\}} \underline{\alpha}$
DAT1	$\overline{[x_{()}]} \xrightarrow{\{\{x_{\bullet u}\}\}} [\overline{x_{(u)}}]$	DAT1'	$\overline{[x_{()}]} \xrightarrow{\{v_x^{\bullet u}\}} [\overline{x_{(u)}}]$
DAT2	$[\overline{x_{(u)}}] \xrightarrow{\{\{x_{u v}\}\}} [\overline{x_{(v)}}]$	DAT2'	$[\overline{x_{(u)}}] \xrightarrow{\{v_x^{u v}\}} [\overline{x_{(v)}}]$
DAT3	$[\overline{x_{(u)}}] \xrightarrow{\{\{x_{u\bullet}\}\}} \underline{[x_{()}]}$	DAT3'	$[\overline{x_{(u)}}] \xrightarrow{\{v_x^{u\bullet}\}} \underline{[x_{()}]}$
PAR	$\dfrac{G \xrightarrow{\Gamma} G' \ , \ H \xrightarrow{\Delta} H'}{G \| H \xrightarrow{\Gamma + \Delta} G' \| H'}$	PAR'	$\dfrac{G \xrightarrow{U} G' \ , \ H \xrightarrow{W} H'}{G \| H \xrightarrow{v_\|^1 \oplus U \ \cup \ v_\|^2 \oplus W} G' \| H'}$
CL	$\dfrac{G \xrightarrow{\Gamma} H}{G \,\square\, E \xrightarrow{\Gamma} H \,\square\, E}$	CL'	$\dfrac{G \xrightarrow{U} H}{G \,\square\, E \xrightarrow{v_\square^1 \oplus U} H \,\square\, E}$
CR	$\dfrac{G \xrightarrow{\Gamma} H}{E \,\square\, G \xrightarrow{\Gamma} E \,\square\, H}$	CR'	$\dfrac{G \xrightarrow{U} H}{E \,\square\, G \xrightarrow{v_\square^2 \oplus U} E \,\square\, H}$
SL	$\dfrac{G \xrightarrow{\Gamma} H}{G; E \xrightarrow{\Gamma} H; E}$	SL'	$\dfrac{G \xrightarrow{U} H}{G; E \xrightarrow{v_;^1 \oplus U} H; E}$
SR	$\dfrac{G \xrightarrow{\Gamma} H}{E; G \xrightarrow{\Gamma} E; H}$	SR'	$\dfrac{G \xrightarrow{U} H}{E; G \xrightarrow{v_;^2 \oplus U} E; H}$
IT1	$\dfrac{G \xrightarrow{\Gamma} H}{\langle G * E] \xrightarrow{\Gamma} \langle H * E]}$	IT1'	$\dfrac{G \xrightarrow{U} H}{\langle G * E] \xrightarrow{v_*^1 \oplus U} \langle H * E]}$
IT2	$\dfrac{G \xrightarrow{\Gamma} H}{\langle E * G] \xrightarrow{\Gamma} \langle E * H]}$	IT2'	$\dfrac{G \xrightarrow{U} H}{\langle E * G] \xrightarrow{v_*^2 \oplus U} \langle E * H]}$
RR	$\dfrac{G \xrightarrow{\Gamma} H}{G[f] \xrightarrow{f(\Gamma)} H[f]}$	RR'	$\dfrac{G \xrightarrow{U} H}{G[f] \xrightarrow{(v_{[f]}, f) \oplus U} H[f]}$

Table 3. PBC operational semantics rules I.

RS	$\dfrac{G \xrightarrow{\Gamma} H}{G \,\mathsf{rs}\, A \xrightarrow{\Gamma} H \,\mathsf{rs}\, A}$	$\alpha \in \Gamma \Rightarrow (A \cup \widehat{A}) \cap \alpha = \emptyset$
RS'	$\dfrac{G \xrightarrow{U} H}{G \,\mathsf{rs}\, A \xrightarrow{v_{\mathsf{rs}\,A} \oplus U} H \,\mathsf{rs}\, A}$	$\alpha \in \mathsf{lab}(U) \Rightarrow (A \cup \widehat{A}) \cap \alpha = \emptyset$
SY	$\dfrac{G \xrightarrow{\Gamma_1 + \ \ldots\ + \Gamma_k} H}{G \,\mathsf{sy}\, A \xrightarrow{\{\alpha_1,\ldots,\alpha_k\}} H \,\mathsf{sy}\, A}$	$(\Gamma_i, \alpha_i) \in \varrho_{\mathsf{sy}\,A}$
SY'	$\dfrac{G \xrightarrow{U_1 \ \mathsf{sy} \ \ldots \ \mathsf{sy}\ U_k} H}{G \,\mathsf{sy}\, A \xrightarrow{\{(v_{\mathsf{sy}\,a},\alpha_1) \lhd U_1,\ldots,(v_{\mathsf{sy}\,a},\alpha_k) \lhd U_k\}} H \,\mathsf{sy}\, A}$	$(\mathsf{lab}(U_i), \alpha_i) \in \varrho_{\mathsf{sy}\,A}$
SC	$\dfrac{G \xrightarrow{\Gamma_1 + \ \ldots\ + \Gamma_k} H}{[A:G] \xrightarrow{\{\alpha_1,\ldots,\alpha_k\}} [A:H]}$	$(\Gamma_i, \alpha_i) \in \varrho_{[A:]}$
SC'	$\dfrac{G \xrightarrow{U_1 \ \mathsf{sy} \ \ldots \ \mathsf{sy}\ U_k} H}{[A:G] \xrightarrow{\{(v_{[A:]},\alpha_1) \lhd U_1,\ldots,(v_{[A:]},\alpha_k) \lhd U_k\}} [A:H]}$	$(\mathsf{lab}(U_i), \alpha_i) \in \varrho_{[A:]}$

Table 4. PBC operational semantics rules II.

Moreover, synchronisation propagates through parallel composition:

$$(E \| F) \,\mathsf{sy}\, A \;\cong\; ((E \,\mathsf{sy}\, A) \| (F \,\mathsf{sy}\, A)) \,\mathsf{sy}\, A$$

- Scoping is idempotent and insensitive to conjugation; it also distributes over choice, sequence and iteration compositions:

$$\begin{array}{lll} [A:[A:E]] \;\cong\; [A:E] & & [\widehat{A}:E] \;\cong\; [A:E] \\ [A:(E \,\square\, F)] \;\cong\; [A:E] \,\square\, [A:F] & & [A:(E;F)] \;\cong\; [A:E];[A:F] \\ [A:\langle E * F]] \;\cong\; \langle [A:E] * [A:F]] & & \end{array}$$

- For non-interfering sets of action particles A and B (i.e., $A \cap (B \cup \widehat{B}) = \emptyset$) synchronisation and scoping commute with restriction:

$$(E \,\mathsf{sy}\, A) \,\mathsf{rs}\, B \;\cong\; (E \,\mathsf{rs}\, B) \,\mathsf{sy}\, A \qquad [A:E] \,\mathsf{rs}\, B \;\cong\; [A:(E \,\mathsf{rs}\, B)]$$

- Relabelling distributes over the control flow operators:

$$\begin{array}{lll} (E;F)[f] \;\cong\; (E[f]);(F[f]) & & (E \,\square\, F)[f] \;\cong\; (E[f]) \,\square\, (F[f]) \\ (E \| F)[f] \;\cong\; (E[f]) \| (F[f]) & & \langle E * F][f] \;\cong\; \langle E[f] * F[f]] \end{array}$$

- For a bijective relabelling f which preserves conjugates (i.e., $f(\widehat{a}) = \widehat{f(a)}$) we obtain properties akin to commutativity and distributivity:

$$\begin{array}{lll} (E \,\mathsf{sy}\, A)[f] \;\cong\; (E[f]) \,\mathsf{sy}\, f(A) & & (E[f]) \,\mathsf{sy}\, A \;\cong\; (E \,\mathsf{sy}\, f^{-1}(A))[f] \\ (E \,\mathsf{rs}\, A)[f] \;\cong\; (E[f]) \,\mathsf{rs}\, f(A) & & (E[f]) \,\mathsf{rs}\, A \;\cong\; (E \,\mathsf{rs}\, f^{-1}(A))[f] \\ [A:E][f] \;\cong\; [f(A):(E[f])] & & [A:(E[f])] \;\cong\; [f^{-1}(A):E][f] \end{array}$$

- Successive relabellings can be collapsed, and $[id]$ (where id is the identity relabelling) is a neutral operator:

$$E[f][g] \;\cong\; E[g \circ f] \qquad E[id] \;\cong\; E$$

- Iteration can be expressed through recursion (see also example 3 below):

$$\langle E * F] \;\approx\; X \quad \text{where} \quad X \stackrel{\text{df}}{=} (E; X) \,\square\, F$$

It is possible to consider other equivalences in the domain of PBC expressions. For example, two static expressions are *duplication equivalent* if so are the corresponding static boxes. The resulting equivalence was studied in [37] where its sound and complete axiomatisation for a variant of the PBC syntax was given.

The applicability of the laws presented in this section can be extended, since they also hold if the behavioural equivalence is expressed using the partial order semantics based on Mazurkiewicz traces.[33]

9.4 Examples

Below we show derivations based on the inference rules and equations for the PBC expressions, indicating which specific rules in tables 2, 3 and 4 have been applied; other rules introduced in section 8.2 are applied implicitly. To simplify notation, in the example expressions, a singleton multiset $\{a\} \in \mathsf{Lab}_{\mathrm{PBC}}$ will be abbreviated by a, and similarly $\{b\}$ by b etc. Moreover, we use $\mathsf{sy}\, a$ to denote $\mathsf{sy}\, \{a\}$.

Example 1 We first deduce that $\overline{(a;b) \,\square\, c} \xrightarrow{\{\{a\}\}\{\{b\}\}} \underline{(a;b) \,\square\, c}$, i.e., that the expression $\overline{(a;b) \,\square\, c}$ can do an a-move, and thereafter a terminating b-move:

$$\begin{array}{lllllll}
\overline{(a;b) \,\square\, c} & \equiv & \overline{(a;b)} \,\square\, c & \text{IC1L} & \equiv & (\overline{a};b) \,\square\, c & \text{IS1} \\
& \xrightarrow{\{\{a\}\}} & (\underline{a};b) \,\square\, c & \text{AR SL CL} & \equiv & (a;\overline{b}) \,\square\, c & \text{IS2} \\
& \xrightarrow{\{\{b\}\}} & (a;\underline{b}) \,\square\, c & \text{AR SR CL} & \equiv & \underline{(a;b)} \,\square\, c & \text{IS3} \\
& \equiv & \underline{(a;b) \,\square\, c} & \text{IC2L} & & &
\end{array}$$

Example 2 As an application of the rule for synchronisation, we show that $\overline{E} \xrightarrow{\{\{b,c\}\}} \underline{E}$ where $E = ((\{a,b\} \| \{\widehat{a},\widehat{a}\}) \| \{a,c\})\ \mathsf{sy}\ a$:

$$\begin{array}{lll}
\overline{E} \;\equiv\; (\overline{(\{a,b\} \| \{\widehat{a},\widehat{a}\}) \| \{a,c\}})\ \mathsf{sy}\ a & & \text{ISY1} \\
\;\equiv\; (\overline{\{a,b\} \| \{\widehat{a},\widehat{a}\}} \| \overline{\{a,c\}})\ \mathsf{sy}\ a & & \text{IPAR1} \\
\;\equiv\; (\overline{\{a,b\}} \| \overline{\{\widehat{a},\widehat{a}\}} \| \overline{\{a,c\}})\ \mathsf{sy}\ a & & \text{IPAR1} \\
\xrightarrow{\{\{b,c\}\}} (\underline{\{a,b\}} \| \underline{\{\widehat{a},\widehat{a}\}} \| \underline{\{a,c\}})\ \mathsf{sy}\ a & & \text{AR PAR SY} \\
\;\equiv\; \underline{E} & & \text{IPAR2 ISY2}
\end{array}$$

[33] For the majority of PBC laws, this follows from the fact that boxes derived for the consituent expressions are isomorphic.

Example 3 The next example shows that $\overline{K} \xrightarrow{\{\{a\}\}\{\{b\}\}} \underline{K}$ where K is a process variable with the defining equation $K \stackrel{\mathrm{df}}{=} (a;K) \,[]\, b$. To conform with the general scheme, we first replace the defining equation by a set of two equations, $K \stackrel{\mathrm{df}}{=} M \,[]\, b$ and $M \stackrel{\mathrm{df}}{=} a;K$. Then the detailed derivation can be carried out as follows:

$\overline{K}$	$\equiv$	$\overline{M \,[]\, b}$		$\equiv$	$\overline{M} \,[]\, b$	IC1L
	$\equiv$	$\overline{a;K} \,[]\, b$		$\equiv$	$(\overline{a};K) \,[]\, b$	IS1
	$\xrightarrow{\{\{a\}\}}$	$(\underline{a};K) \,[]\, b$	AR SL CL	$\equiv$	$(a;\overline{K}) \,[]\, b$	IS2
	$\equiv$	$(a;\overline{(M \,[]\, b)}) \,[]\, b$		$\equiv$	$(a;(M \,[]\, \overline{b})) \,[]\, b$	IC1R
	$\xrightarrow{\{\{b\}\}}$	$(a;(M \,[]\, \underline{b})) \,[]\, b$	AR CR SR CL	$\equiv$	$(a;\underline{(M \,[]\, b)}) \,[]\, b$	IC2R
	$\equiv$	$(a;\underline{K}) \,[]\, b$		$\equiv$	$\underline{(a;K)} \,[]\, b$	IS3
	$\equiv$	$\underline{M} \,[]\, b$		$\equiv$	$\underline{M \,[]\, b}$	IC2L
	$\equiv$	$\underline{K}$				

Note that $\overline{K}$ can make an infinite sequence of a-moves since $\overline{K} \xrightarrow{\{\{a\}\}} (a;\overline{K}) \,[]\, b$.

Example 4 The fourth example is similar to that discussed in [1] and involves a non-tail-end recursive expression not directly expressible in CCS. Below we show that the dynamic expression $\overline{L}$, for which the defining equation is $L \stackrel{\mathrm{df}}{=} (L;a) \,[]\, a$, can make two successive a-moves, i.e.,

$$\overline{L} \xrightarrow{\{\{a\}\}\{\{a\}\}} \underline{L} \ .$$

We first transform the equation into the format required by the scheme developed in section 8: $L \stackrel{\mathrm{df}}{=} N \,[]\, a$ and $N \stackrel{\mathrm{df}}{=} L;a$. The derivation is then obtained in the following way:

$$\begin{array}{llllllll}
\overline{L} & \equiv \overline{N \,[]\, a} & \equiv \overline{N} \,[]\, a & \equiv \overline{(L;a)} \,[]\, a & \equiv (\overline{L};a) \,[]\, a \\
& \equiv (\overline{(N \,[]\, a)};a) \,[]\, a & \equiv ((N \,[]\, \overline{a});a) \,[]\, a & \xrightarrow{\{\{a\}\}} ((N \,[]\, \underline{a});a) \,[]\, a & \equiv \underline{(N \,[]\, a)};a) \,[]\, a \\
& \equiv (\underline{L};a) \,[]\, a & \equiv (L;\overline{a}) \,[]\, a & \xrightarrow{\{\{a\}\}} (L;\underline{a}) \,[]\, a & \equiv \underline{(L;a)} \,[]\, a \\
& \equiv \underline{N} \,[]\, a & \equiv \underline{N \,[]\, a} & \equiv \underline{L}
\end{array}$$

By induction, it can be shown that $\overline{L}$ can do n successive a-moves, for every n. But the argument cannot be used to show that it can do an infinite sequence of a-moves; in fact it cannot. This may be contrasted with the previous example.

Example 5 We now prove by induction that the dynamic expression $\overline{C}$, where $C \stackrel{\mathrm{df}}{=} a \| C$, can do a Γ_n-move, for every $\Gamma_n = n \cdot \{\{a\}\}$, $n \geq 1$. That is, for every $n \geq 1$, there is C_n such that

$$\overline{C} \xrightarrow{\Gamma_n} C_n \ .$$

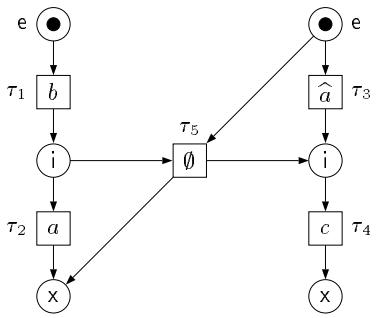
In the base step, $n = 1$, the above follows from the following derivation:

$$\overline{C} \;\equiv\; \overline{a \| C} \;\equiv\; \overline{a} \| \overline{C} \;\stackrel{\{\{a\}\}}{\longrightarrow}\; \underline{a} \| \overline{C} \;=\; C_1$$

In the induction step, by the induction hypothesis, there is C_n such that $\overline{C} \stackrel{\Gamma_n}{\longrightarrow} C_n$. Then

$$\overline{C} \;\equiv\; \overline{a \| C} \;\equiv\; \overline{a} \| \overline{C} \;\stackrel{\{\{a\}\}+\Gamma_n}{\longrightarrow}\; \underline{a} \| C_n \;=\; C_{n+1}$$

which completes the proof.



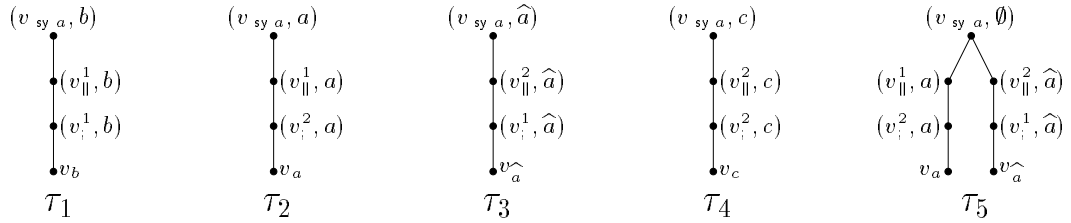


Fig. 28. The box of $\overline{E} = \overline{((b;a) \| (\widehat{a};c))\ \mathsf{sy}\ a}$.

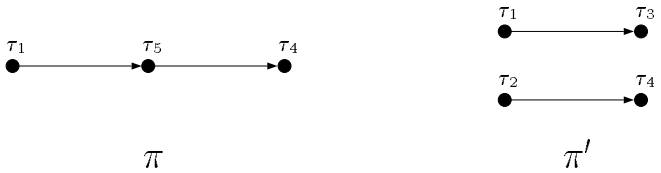
Example 6 We finally turn to the partial order semantics based on transition trees and the derived partial order semantics. Let E be an expression defined as $((b;a) \| (\widehat{a};c))\ \mathsf{sy}\ a$. Figure 28 shows $\mathsf{box}(\overline{E})$ together with the transition trees which serve as the actual names of transitions. Note that there are four pairs of independent transition trees in the sense of the relation ind^{alg}, namely (τ_1, τ_3), (τ_1, τ_4), (τ_2, τ_3) and (τ_2, τ_4). It is possible to show that $\overline{E}$ can generate the following two step sequences,

$$\overline{E} \stackrel{\{\tau_1\}\{\tau_5\}\{\tau_4\}}{\longrightarrow} \underline{E} \quad \text{and} \quad \overline{E} \stackrel{\{\tau_1\}\{\tau_2,\tau_3\}\{\tau_4\}}{\longrightarrow} \underline{E}\,.$$

Below we give a derivation for the first one:

$$
\begin{array}{lllllll}
\overline{E} & \equiv & \overline{((b;a)\|(\widehat{a};c))}\ \mathsf{sy}\ a & \equiv & (\overline{(b;a)}\|\overline{(\widehat{a};c)})\ \mathsf{sy}\ a & \equiv & ((\overline{b};a)\|(\overline{\widehat{a}};c))\ \mathsf{sy}\ a \\
 & \stackrel{\{\tau_1\}}{\longrightarrow} & ((\underline{b};a)\|(\overline{\widehat{a}};c))\ \mathsf{sy}\ a & \equiv & ((b;\overline{a})\|(\overline{\widehat{a}};c))\ \mathsf{sy}\ a & \stackrel{\{\tau_5\}}{\longrightarrow} & ((b;\underline{a})\|(\underline{\widehat{a}};c))\ \mathsf{sy}\ a \\
 & \equiv & (\underline{(b;a)}\|(\widehat{a};\overline{c}))\ \mathsf{sy}\ a & \stackrel{\{\tau_4\}}{\longrightarrow} & (\underline{(b;a)}\|(\widehat{a};\underline{c}))\ \mathsf{sy}\ a & \equiv & \underline{(b;a)}\|\underline{(\widehat{a};c)})\ \mathsf{sy}\ a \\
 & \equiv & \underline{((b;a)\|(\widehat{a};c))}\ \mathsf{sy}\ a & \equiv & \underline{E} & &
\end{array}
$$

As a result, we obtain that $\overline{E} \stackrel{\pi}{\longrightarrow} \underline{E}$ and $\overline{E} \stackrel{\pi'}{\longrightarrow} \underline{E}$, where π and π' are the following labelled partial orders:



π $\qquad\qquad$ π'

9.5 CCS, TCSP and COSY

In this section we outline how CCS, TCSP and COSY could be treated in the general compositionality framework provided by the box algebra. In what follows, by a *simple operator box* we will mean a two-place one-transition operator box similar to $\Omega_{[f]}$ shown in figure 27. We will denote such an operator box by $\Omega{:}\varrho$, where ϱ is the relabelling of its only transition.

CCS To model CCS processes we assume that $\mathsf{Lab} = Act \cup \{\tau\}$ is the set of CCS labels and $\widehat{\ }{:}\, Act \to Act$ is a bijection on Act satisfying $\widehat{\widehat{a}} = a$ and $\widehat{a} \neq a$, for all $a \in Act$. We then define five simple operator boxes, $\Omega{:}restr(b)$, $\Omega{:}relab(h)$, $\Omega{:}left$, $\Omega{:}right$ and $\Omega{:}syn(CCS)$, where $b \in Act$ and h is a mapping on labels $h : \mathsf{Lab} \to \mathsf{Lab}$ commuting with $\widehat{(.)}$, defined by the following relabellings:

$$
\begin{array}{ll}
restr(b) & = \{(\{a\}, a) \mid a \in \mathsf{Lab}\backslash\{b\}\} \\
relab(h) & = \{(\{a\}, h(a)) \mid a \in \mathsf{Lab}\} \\
left & = \{(\{a\}, a^L) \mid a \in \mathsf{Lab}\} \\
right & = \{(\{a\}, a^R) \mid a \in \mathsf{Lab}\} \\
syn(CCS) & = \{(\{a^L, \widehat{a}^R\}, \tau) \mid a \in Act\} \ \cup\ \{\,(\{a^L\}, a)\ ,\ (\{a^R\}, a) \mid a \in \mathsf{Lab}\,\}\ .
\end{array}
$$

It is assumed that Lab is extended by the labels a^L and a^R, for $a \in \mathsf{Lab}$, but neither a^L nor a^R are allowed in the syntax of CCS expressions; they are auxiliary symbols used to model correctly the semantics of CCS parallel composition. Below, we give a denotational semantics of CCS, through the full translation ϕ_{CCS} for the CCS processes conforming to

the standard syntax [50]:

$$\begin{aligned}
\phi_{CCS}(\mathsf{nil}) &= [\{a\} : a] \quad \text{(for any } a\text{)} \\
\phi_{CCS}(a.E) &= \mathsf{op}_{\Omega.}(\mathsf{N}_a, \phi_{CCS}(E)) \\
\phi_{CCS}(E+F) &= \mathsf{op}_{\Omega\,\square}(\phi_{CCS}(E), \phi_{CCS}(F)) \\
\phi_{CCS}(E|F) &= \mathsf{op}_{\Omega:syn(CCS)}(\mathsf{op}_{\Omega\,\|}(\mathsf{op}_{\Omega:left}(\phi_{CCS}(E)), \mathsf{op}_{\Omega:right}(\phi_{CCS}(F)))) \\
\phi_{CCS}(E[h]) &= \mathsf{op}_{\Omega:relab(h)}(\phi_{CCS}(E)) \\
\phi_{CCS}(E\backslash a) &= \mathsf{op}_{\Omega:restr(a)}(\phi_{CCS}(E)) \ .
\end{aligned}$$

TCSP The other two models employ synchronisation mechanisms different from those used in CCS and PBC. We will not provide a full translation of TCSP and COSY into a box algebra, leaving this task as an exercise for the reader. Rather, we shall briefly explain how concurrent composition – and thus synchronisation – used in these two models could be handled.

The concurrent composition in TCSP [39] is a binary operator, $E\|_A F$, where $A \subseteq \mathsf{Lab}$ is a set of actions on which $\|_A$ enforces synchronisation. The resulting process can execute an action $a \in A$ if it can be executed simultaneously by the two component processes; this simultaneous execution is denoted by a. The actions outside A can be executed autonomously by the two component processes. Such a synchronisation discipline can be modelled similarly as in CCS. This time, however, we do not assume that the set of labels, Lab, has any special properties. The relevant fragment of the transformation ϕ_{TCSP} from TCSP expressions to box algebra expressions is modelled thus:

$$\phi_{TCSP}(E\|_A F) \ = \ \mathsf{op}_{\Omega:TCSP(A)}(\mathsf{op}_{\Omega\,\|}(\mathsf{op}_{\Omega:left}(\phi_{TCSP}(E)) \ , \ \mathsf{op}_{\Omega:right}(\phi_{TCSP}(F))))$$

where $\Omega{:}TCSP(A)$ is a simple operator box with the relabelling defined thus:

$$TCSP(A) \ = \ \{(\{a^L, a^R\}, a) \mid a \in A\} \cup \{\, (\{a^L\}, a) \ , \ (\{a^R\}, a) \mid a \in (\mathsf{Lab}\backslash A)\} \ .$$

As before, we assume that the set of labels is temporarily extended by labels a^L and a^R, for each $a \in A$, which are not in the TCSP syntax.

COSY The concurrent composition in COSY [40] is based on a multi-way synchronisation. Consider a path program

$$prog \ = \ \mathsf{program} \quad path_1 \quad \ldots \quad path_n \quad \mathsf{endprogram}$$

Here, $prog$ can execute a if it is executed simultaneously in all the paths $path_i$ in which a occurs. To model such a synchronisation mechanism, we first extend $\Omega_\|$ to n-ary ($n \geq 1$) parallel composition operator boxes, $\Omega^n_\|$, in the obvious way. Let $A \subseteq \mathsf{Lab}$ be the set of labels occurring in program $prog$ and, for every $a \in A$, let $index(a)$ be the set of all indices i such that a occurs in $path_i$. Let $Index = \{(a, index(a)) \mid a \in A\}$. Define a simple operator $\Omega{:}COSY(Index)$ with the relabelling being given by:

$$COSY(Index) \ = \ \bigcup_{a \in A} \{(\bigcup_{i \in index(a)} \{a^i\}, a)\}$$

and n simple operators, $\Omega{:}index_i$, with the relabellings: $index_i = \{(\{a\}, a^i) \mid a \in \mathsf{Lab}\}$, for $i = 1, \ldots, n$. Then the relevant fragment of the transformation ϕ_{COSY} from COSY programs to box algebra expressions is modelled thus:

$$\phi_{COSY}(prog) = \mathsf{op}_{\Omega:COSY(Index)}(\mathsf{op}_{\Omega^n_{\|}}(\mathsf{op}_{\Omega:index_1}(\phi_{COSY}(path_1)), \ldots, \mathsf{op}_{\Omega:index_n}(\phi_{COSY}(path_n))))) \ .$$

10 Modelling a concurrent programming language

In order to illustrate how it is possible to exploit the framework developed so far let us discuss informally how to give a fully compositional semantics to a concurrent programming language using PBC. Consider, for example, the following fragment of a concurrent program:

$$\begin{array}{l} \ldots \\ \quad \textbf{begin}\ \ \textbf{var}\ x : \{0,1\}; \\ \qquad\quad [\![\, x{:=}x \oplus 1 \,]\!] \,\|\, [\![\, x{:=}y \,]\!] \\ \quad \textbf{end} \\ \ldots \end{array}$$

where y is assumed to be declared by **var** $y : \{0,1\}$ in some outer block, $\oplus$ denotes the addition modulo 2, $\|$ denotes 'shared variable parallelism' (as variable x occurs on both sides of $\|$), and $[\![\ldots]\!]$ delineates an atomic action. Let us discuss how to construct an appropriate, and as small as possible, Petri net modelling this block. Moreover, such a net should be constructed compositionally from nets derived for its three constituents: the declaration **var** $x : \{0,1\}$, and the two atomic assignments, $[\![\, x{:=}x \oplus 1 \,]\!]$ and $[\![\, x{:=}y \,]\!]$, and itself be composable, i.e., usable in further compositions with similar nets, e.g., derived for the outer blocks.

As in section 9.1, we use special action particles, in the form of indexed terms x_{vw} and $\widehat{x_{vw}}$ where $v, w \in \{0,1\}$. Each such term denotes a change of the value of the program (or data) variable x from v to w, or a test of the value of x, if $v = w$. Then the following will be translations of the two assignments into PBC:

$$\begin{array}{ll} [\![\, x{:=}x{\oplus}1 \,]\!] & \rightsquigarrow \{\widehat{x_{01}}\} \,\square\, \{\widehat{x_{10}}\} \\ [\![\, x{:=}y \,]\!] & \rightsquigarrow \{\widehat{x_{00}}, \widehat{y_{00}}\} \,\square\, \{\widehat{x_{10}}, \widehat{y_{00}}\} \,\square\, \{\widehat{x_{01}}, \widehat{y_{11}}\} \,\square\, \{\widehat{x_{11}}, \widehat{y_{11}}\} \ . \end{array} \tag{29}$$

The first expression means that 'either x could be 0, and then it is changed into 1, or x could be 1, and then it is changed into 0', which is a sound semantics of $[\![\, x{:=}x{\oplus}1 \,]\!]$ for a binary variable. Unlike the multiactions in the first line, the two-element multiactions in the second PBC expression are not singletons because $[\![\, x{:=}y \,]\!]$ involves both variables, x and y. Each two-element multiaction should be interpreted as denoting two simultaneously executable accesses to these variables, one checking the value of y, the other updating the value of x. For instance, $\{\widehat{x_{10}}, \widehat{y_{00}}\}$ denotes the value of y being checked to be 0 and, simultaneously, the value of x being changed from 1 to 0. The expressions for $[\![\, x{:=}x \oplus 1 \,]\!]$ and $[\![\, x{:=}y \,]\!]$ given in (29) have corresponding Petri nets with respectively two and four transitions labelled by the corresponding multiactions, as shown in figure 29.

The Petri net corresponding to the declaration of x involves action particles x_{vw}, where x_{vw} is the conjugate of $\widehat{x_{vw}}$, which ensures that the net of a variable can be put in parallel

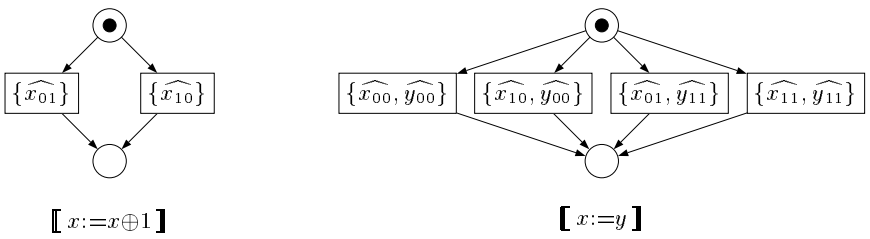


Fig. 29. Petri nets representing $[\![x{:=}x{\oplus}1]\!]$ and $[\![x{:=}y]\!]$.

with the net of an atomic action, and then both nets can be synchronised in order to describe a block. The net describing **var** $x : \{0, 1\}$ is shown in figure 27 and – in abbreviated form – in figure 30. In the latter, it is arbitrarily assumed that the current value of x is 1, and thus the net contains a token on the corresponding place.

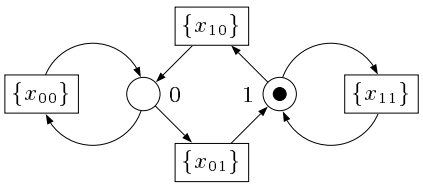


Fig. 30. Part of the Petri net representing a binary variable x, $\mathsf{N}_{[x_{()}]}$.

What was just described is the principal use of conjugation in the PBC semantics of programming languages; we always assume that the command part of a block employs 'hatted' action symbols, while the declaration part uses their 'unhatted' (conjugated) versions in order to describe accesses to variables. A block will be described by putting command part and declaration part in parallel and then synchronising and restricting (hence scoping) over all variables that are local to it. This leaves only non-local accesses to be still visible in the communication interface, i.e., in the transitions.

In order to describe the block structure of the program fragment (with x being a local and y global variable of the block), the net of **var** $x : \{0, 1\}$ needs to be synchronised with the nets of $[\![x{:=}x \oplus 1]\!]$ and $[\![x{:=}y]\!]$. For example, the transition labelled $\{x_{10}\}$ coming from the declaration of x is synchronised with the transition labelled $\{\widehat{x_{10}}, \widehat{y_{00}}\}$ coming from the action $[\![x{:=}y]\!]$ using and consuming the conjugate pair $(\widehat{x_{10}}, x_{10})$ but retaining $\widehat{y_{00}}$ (as we have already seen, this mechanism is built into the definition of synchronisation). The resulting transition has the label $\{\widehat{y_{00}}\}$. After that, all the transitions with labels containing $\widehat{x_{uv}}$ and x_{uv} action particles are removed by applying the PBC restriction operator, since the variable x is not known outside its declaring block. The net of the block, obtained through synchronisation and restriction (i.e., scoping), is shown in figure 31. This net describes the

effect of actions on local variable x directly by its connectivity, and on global variable y indirectly through its labelling. The transitions labelled $\{\widehat{y_{vv}}\}$ can be synchronised again with transitions labelled $\{y_{vv}\}$ coming from the declaration of y in an outer block, using and consuming conjugate pairs $(y_{vv}, \widehat{y_{vv}})$ and yielding transitions labelled by $\emptyset$, after what is a 3-way synchronisation. During the translation process we have thus seen an effective application of the multiway synchronisation mechanism, in the modelling of an assignment involving two program variables which are declared in different blocks but accessed in the same atomic assignment.

The details and formal treatment are in [8].

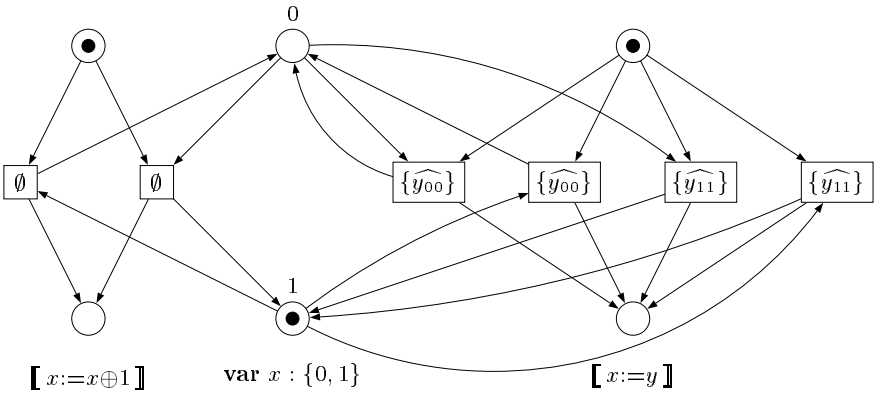


Fig. 31. Petri net representation of a small program fragment.

Acknowledgements

The work described in this chapter has greatly benefited from cooperation with other people, in particular Javier Esparza, Jon Hall and Richard Hopkins. It has been supported by the Esprit Basic Research Project 3148 DEMON (Design Methods Based on Nets) and Working Group 6067 CALIBAN (Causal Calculi Based on Nets), as well as the Anglo-German Foundation ARC Project 1032 BAT (Box Algebra with Time).

References

1. J. Baeten and W. P. Weijland: *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press (1990).
2. J.. C.. M.. Baeten and C.. Verhoef: Concrete Process Algebra. In: *Handbook of Logic in Computer Science*, S.. Abramsky, D.. M.. Gabbay and T.. S.. E.. Maibaum. Oxford University Press (1995) 149-268.
3. E. Best: *Semantics of Sequential and Parallel Programs.* Prentice Hall (1996).

4. E. Best, R. Devillers and M. Koutny: Petri Nets, Process Algebras and Concurrent Programming Languages. In: *Advances in Petri Nets. Lectures on Petri Nets II: Applications*, W. Reisig and G. Rozenberg (Eds.). Springer-Verlag, Lecture Notes in Computer Science 1492 (1998) 1-84.
5. E. Best: Partial Order Verification with PEP. Proc. of *POMIV'96: Partial Order Methods in Verification*, G. Holzmann, D. Peled and V. Pratt (Eds.). American Mathematical Society (1997) 305-328.
6. E. Best and R. Devillers: Sequential and Concurrent Behaviour in Petri Net Theory. *Theoretical Computer Science* 55 (1988) 87-136.
7. E. Best, R. Devillers and J. Esparza: General Refinement and Recursion Operators for the Petri Box Calculus. Proc. of *STACS 93, 10th Annual Symposium on Theoretical Aspects of Computer Science*, P. Enjalbert, A. Finkel and K. W. Wagner (Eds.). Springer-Verlag, Lecture Notes in Computer Science 665 (1993) 130-140.
8. E. Best, R. Devillers and M. Koutny: *Petri Net Algebra*. Book Manuscript (1999).
9. E. Best, R. Devillers and M. Koutny: The Box Algebra - a Model of Nets and Process Expressions. Proc. of *ICATPN'99*, S. Donatelli and J. Kleijn. Springer-Verlag, Lecture Notes in Computer Science 1639 (1999) 344-363.
10. E. Best, R. Devillers and J. Hall: The Petri Box Calculus: a New Causal Algebra with Multilabel Communication. In: *Advances in Petri Nets1992*, G. Rozenberg (Ed.). Springer-Verlag, Lecture Notes in Computer Science 609 (1992) 21-69.
11. E. Best, H. Fleischhack, W. Frączak, R. P. Hopkins, H. Klaudel and E. Pelz: A Class of Composable High Level Petri Nets. Proc. of *ICATPN'95*, G. De Michelis and M. Diaz (Eds.). Springer-Verlag, Lecture Notes in Computer Science 935 (1995) 103-120.
12. E. Best, W. Frączak, R. P. Hopkins, H. Klaudel and E. Pelz: M-nets: an Algebra of High-level Petri Nets, with an Application to the Semantics of Concurrent Programming Languages. *Acta Informatica* 53 (1998) 813-857.
13. E. Best and B. Grahlmann: PEP - more than a Petri Net Tool. Proc. of *TACAS'96: Tools and Algorithms for the Construction and Analysis of Systems*, T. Margaria and B. Steffen (Eds.). Springer-Verlag, Lecture Notes in Computer Science 1055 (1996) 397-401.
14. E. Best and R. P. Hopkins: $B(PN)^2$ - a Basic Petri Net Programming Notation. Proc. of *PARLE'93: Parallel Architectures and Languages Europe*, A. Bode, M. Reeve and G. Wolf (Eds.). Springer-Verlag, Lecture Notes in Computer Science 694 (1993) 379-390.
15. E. Best and H.-G. Linde-Göers: Compositional Process Semantics of Petri Boxes. Proc. of *Mathematical Foundations of Programming Semantics (MFPS'93)*, S. Brookes, M. Main, A. Melton, M. Mislove, D. Schmidt (Eds.). Springer-Verlag, Lecture Notes in Computer Science 802 (1994) 250-270.
16. G. Boudol and I. Castellani: Flow Models of Distributed Computations: Event Structures and Nets. Teachnical Report RR-1484, INRIA, Sophia Antipolis (1991).
17. G. W. Brams (nom collectif de Ch. André, G. Berthelot, C. Girault, G. Memmi, G. Roucairol, J. Sifakis, R. Valette, G. Vidal-Naquet): *Résaux de Petri: Théorie et Pratique*. Two volumes. Editions Masson (1985).
18. G. Bruns and J. Esparza: Trapping Mutual Exclusion in the Box Calculus. *Theoretical Computer Science* 153 (1995) 95-128.
19. B. A. Davey and H. A. Priestley: *Introduction to Lattices and Order*. Cambridge Mathematical Textbook (1990).
20. P. Degano, R. De Nicola and U. Montanari: A Distributed Operational Semantics for CCS Based on C/E Systems. *Acta Informatica* 26 (1988) 59-91.
21. J. Desel and J. Esparza: *Free Choice Petri Nets*. Cambridge Tracts in Theoretical Computer Science 40, Cambridge University Press (1995).
22. R. Devillers: Maximality Preservation and the ST-idea for Action Refinements. In: *Advances in Petri Nets1992*, G. Rozenberg (Ed.). Springer-Verlag, Lecture Notes in Computer Science 609 (1992) 108-151.

23. R. Devillers: Construction of S-invariants an S-components for Refined Petri Boxes. Proc. of *ICATPN'93*, M. Ajmone Marsan (Ed.). Springer-Verlag, Lecture Notes in Computer Science 691 (1993) 242-261.
24. R. Devillers: S-invariant Analysis of General Recursive Petri Boxes. *Acta Informatica* 32 (1995) 313-345.
25. R. Devillers and M. Koutny: Recursive Nets in the Box Algebra. Proc. of *CSD'98: International Conference on Application of Concurrency to System Design*, IEEE Press (1998) 239-249.
26. H. Fleischhack and B. Grahlmann: A Petri Net Semantics for BPN^2 with Procedures which Allows Verification. Teachnical Report 21/96, Hildesheimer Informatikbericht (1996).
27. W.. J.. Fokkink: *Introduction to Process Algebra*. Springer-Verlag. Texts in Theoretical Computer Science. An EATCS Series (2000).
28. H. J. Genrich, K. Lautenbach and P. S. Thiagarajan: Elements of General Net Theory. In: *Net Theory and Applications, Proc. of the Advanced Course on General Net Theory of Processes and Systems*, W. Brauer (Ed.). Springer-Verlag, Lecture Notes in Computer Science 84 (1980) 21-163.
29. R. van Glabbeek: The refinement theorem for ST-bisimulation semantics. Proc. of *IFIP Working Conference on Programming Concepts and Methods*, M. Broy and C. Jones (Eds.). IFIP, Elsevier Science Publishers B.V. (North-Holland) (1990) 27-52.
30. U. Goltz: On Representing CCS Programs by Finite Petri Nets. Proc. of *MFCS'88*, M. P. Chytil, L. Janiga and V. Koubek (Eds.). Springer-Verlag, Lecture Notes in Computer Science 324 (1988) 339-350.
31. U. Goltz and R. van Glabbeek: Refinement of Actions in Causality Based Models. Proc. of *Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness, Proceedings REX Workshop*, J. W. de Bakker, W.-P. de Roever and G. Rozenberg (Eds.). Springer-Verlag, Lecture Notes in Computer Science 430 (1990) 267-300.
32. U. Goltz and R. Loogen: A Non-interleaving Semantic Model for Nondeterministic Concurrent Processes. *Fundamentae Informaticae* 14 (1991) 39-73.
33. U. Goltz and W. Reisig: The Non-sequential Behaviour of Petri Nets. *Information and Control* 57 (1983) 125-147.
34. J. Grabowski: On Partial Languages. *Fundamentae Informaticae* 4 (1981) 427-498.
35. J. G. Hall: *An Algebra of High-level Petri Nets*. PhD Thesis, University of Newcastle upon Tyne (1996).
36. M. B. Hennessy: *Algebraic Theory of Processes*. The MIT Press (1988).
37. M. Hesketh and M. Koutny: An Axiomatisation of Duplication Equivalence in the Petri Box Calculus. Proc. of *ICATPN'98*, J. Desel and M. Silva (Eds.). Springer-Verlag, Lecture Notes in Computer Science 1420 (1998) 165-184.
38. C. A. R. Hoare: Communicating Sequential Processes. *Communications of the ACM* 21 (1978) 666-677.
39. C. A. R. Hoare: *Communicating Sequential Processes*. Prentice Hall (1985).
40. R. Janicki and P. E. Lauer: *Specification and Analysis of Concurrent Systems - the COSY Approach*. EATCS Monographs on Theoretical Computer Science, Springer-Verlag (1992).
41. K. Jensen: *Coloured Petri Nets. Basic Concepts*. EATCS Monographs on Theoretical Computer Science, Springer-Verlag (1992).
42. H. Klaudel and R.-Ch. Riemann: Consistent Equivalence Notion for a Class of High Level Petri Nets. Proc. of *ISCIS, 11th International Symposium on Computer and Information Science*, V. Atalay, U. Halici, K. Inan, N. Yalabik and A. Yazici (Eds.). Middle East Technical University, Antalya (1996) 7-16.
43. M. Koutny and E. Best: Fundamental Study: Operational and Denotational Semantics for the Box Algebra. *Theoretical Computer Science* 211 (1999) 1-83.
44. M. Koutny: Partial Order Semantics of Box Expressions. Proc. of *ICATPN'94*, R. Valette (Ed.). Springer-Verlag, Lecture Notes in Computer Science 815 (1994) 318-337.

45. P. E. Lauer: Path Expressions as Petri Nets, or Petri Nets with Fewer Tears. Teachnical Report MRM/70, Computing Laboratory, University of Newcastle upon Tyne (1974).
46. K. Lautenbach: Liveness in Petri Nets. Teachnical Report ISF/75-02-1, GMD, Bonn (1975).
47. J. Lilius and E. Pelz: An M-net Semantics for BPN^2 with Procedures. Proc. of *ISCIS, 11th International Symposium on Computer and Information Science*, V. Atalay, U. Halici, K. Inan, N. Yalabik and A. Yazici (Eds.). Middle East Technical University, Antalya (1996) 365-374.
48. A. Mazurkiewicz: Trace Theory. In: *Advances in Petri Nets1986, Petri Nets: Applications and Relationships to Other Models of Concurrency, Part II*, W. Brauer, W. Reisig and G. Rozenberg (Eds.). Springer-Verlag, Lecture Notes in Computer Science 255 (1987) 279-324.
49. R. Milner: *A Calculus of Communicating Systems.* Springer-Verlag, Lecture Notes in Computer Science 92 (1980).
50. R. Milner: *Communication and Concurrency.* Prentice Hall (1989).
51. T. Murata: Petri Nets: Properties, Analysis and Applications. *Proc. of IEEE* 77 (1989) 541-580.
52. E. R. Olderog: *Nets, Terms and Formulas.* Cambridge Tracts in Theoretical Computer Science 23, Cambridge University Press (1991).
53. G. D. Plotkin: A Structural Approach to Operational Semantics. Teachnical Report FN-19, Computer Science Department, University of {a}rhus (1981).
54. W. Reisig: *Petri Nets. An Introduction.* EATCS Monographs on Theoretical Computer Science, Springer-Verlag (1985).
55. W. Reisig: Modelling and Verification of Distributed Algorithms. Proc. of *CONCUR 96, Concurrency Theory, 7th International Conference*, U. Montanari and V. Sassone (Eds.). Springer-Verlag, Lecture Notes in Computer Science 1119 (1996) 579-595.
56. W. Reisig: *Elements of Distributed Algorithms.* Springer-Verlag (1998).
57. P. H. Starke: Processes in Petri Nets. *Elektronische Informationsverarbeitung und Kybernetik* 17 (1981) 389-416.
58. D. Taubner: *Finite Representation of CCS and TCSP Programs by Automata and Petri Nets.* Springer-Verlag, Lecture Notes in Computer Science 369 (1989).
59. W. Vogler: Partial Words versus Processes: a Short Comparison. In: *Advances in Petri Nets1992*, G. Rozenberg (Ed.). Springer-Verlag, Lecture Notes in Computer Science 609 (1992) 292-303.
60. W. Vogler: Bisimulation and Action Refinement. *Theoretical Computer Science* 114 (1993) 173-200.

Index

Ω-tuple, 25

ACP, 3, 8
action, 12
– redo, 20
– skip, 20
action particles, 10, 11, 73, 84
approximations, 30
auto-concurrency, 15

B-cut, 17
Beh1 – Beh5, 59
bisimulation
– strong, 22
bounded net, 15
boundedness, 15
box, 11, 23
– seed, 32
– dynamic, 24
– e-directed, 14, 58
– entry, 24
– ex-directed, 14
– ex-exclusive, 25, 58
– exit, 24
– factorisable, 52
– operator, 11
– plain, 11, 23
– rec-, 30
– seed, 36
– semi-dynamic, 46
– semi-static, 46
– sos operator, 53
– static, 24
– x-directed, 14, 58
box algebra, 60
– denotational semantics, 63
– independence relation ind^{alg}, 68
– semantics
–– label based, 69
–– partial order based, 68
–– transition based, 66
– signature, 60
– structural equivalence, 64
– syntax, 60
– transition trees, 66
box expressions, 61

cardinality of multiset, 7
causality, 20
chain, 30
clean
– marking, 15
– net, 15
cleanness, 15
communication interface operator, 11
complex marking, 16
– imaginary part, 16
– real part, 16
concurrency, 17, 18
conservative S-invariant, 39
constant
– dynamic, 60
– exit, 60
– static, 60
constant boxes, 60
constant relabelling, 13
constants, 60
continuity, 31
control flow operator, 11
COSY, 3, 83
CSP, 3, 83
cuts consistency, 18

defining equation, 61
degree-finite place, 43
denotational semantics, 63
derivability, 15
distribution function, 42
– multiplicative, 42
DIY algebra, 61
– constants, 62
– denotational semantics, 64
– operational semantics, 66, 67
– similarity relation, 65
– syntax, 62
domain of application of sos-operator box, 55
duplication, 23
dynamic
– box, 24
– constant, 60
– expression, 60

e-directed
– box, 14
– expression, 62
– variable, 58
enabled
– step, 14
– transition, 4
entry
– constant, 60
– expression, 60
– marking, 14
– place, 13
entry box, 24
equation
– X-finitary, 31
– defining, 61
– front-guarded, 35
– guarded, 35
– rear-guarded, 35
event occurrence, 19
ex-binary S-invariant, 43
ex-directed
– box, 14
ex-exclusive
– box, 25
– expression, 62
– marking, 15
– variable, 58
ex-restrictedness, 14
exit
– constant, 60
– expression, 60
– marking, 14
– place, 13
exit box, 24
Expr1 – Expr6, 62
expression
– dynamic, 60
– e-directed, 62
– entry, 60
– ex-directed, 62
– ex-exclusive, 62
– exit, 60
– flat, 61
– static, 60, 73
– well formed, 62
– x-directed, 62

factorisable box, 52
factorisation
– of a complex marking, 52
– of a set of places, 52
– of a tuple of boxes, 55
– of a tuple of expressions, 61
final coherence, 17
finite accessibility
– first FFA, 18
– second SFA, 18
– third TFA, 19
fixpoint
– positive, 33
full transition system
– of a net, 21
– of an expression, 69

global independence relation, 68
guardedness, 35
– front, 35
– rear, 35

handshake, 8
hatting, 8

identity relabelling, 13
independence relation, 19
– global, 68
initial marking, 4
initial coherence, 17
internal place, 13
isomorphism of nets, 22

labelled net, 13
labelled step, 15
labelling function, 13
limit, 30
limit-preserved property, 30
linearisation, 17
local coherence, 17

marked net, 21
marking, 4, 13
– clean, 15
– complex, 16
– entry, 14
– ex-exclusive, 15
– exit, 14
– initial, 4
– reachable, 4, 15
– safe, 4, 15
Mazurkiewicz traces, 19

monotonicity, 31
multiactions
– $\mathsf{Lab}_{\mathrm{PBC}}$, 10
multiplicative distribution function, 42
multiset, 7
– cardinality, 7
– difference, 7
– inclusion, 7
– sum, 7

names of places and transitions, 26
net
– behaviour, 4
– bounded, 15
– clean, 15
– derivable, 15
– ex-restricted, 14
– finite, 13
– isomorphism, 22
– labelled, 13
– marked, 21
– pure, 14
– S- or T-covered, 39
– safe, 4, 15
– simple, 14
– T-restriction, 14
– transition system ts_{Σ}, 21
– unmarked, 21
net refinement, 25

occurence net, 16
occurrence
– of transitions, 4
operational semantics for box algebra, 66
operator
– communication interface, 11
– control flow, 11
operator box, 11, 25
– sos, 53
ordering
– on labelled nets, 30
overbarring, 5

partial order, 7
PBC syntax, 73, 74
place, 13
– degree-finite, 43
– entry, 13
– exit, 13
– internal, 13
– labelling, 13
plain box, 11
predecessor, 17
process, 16
pure net, 14

reachability, 15
– graph, 20
reachable marking, 4
rec-box, 30
recursion
– maximal solution, 35
– minimal solution, 35
– unique solution, 35
– variable, 30
redo, 20
relabelling
– constant, 13
– identity, 13
– relation, 12
– restriction, 13
– transformational, 13
reversible transition, 55

S- or T-covered node, 39
S-aggregate, 40
S-invariant, 39
– conservative, 39
– ex-binary, 43
– weight of, 39
safe
– marking, 4, 15
– net, 4, 15
safeness, 15
SCCS, 3
seed box, 32, 36
semi-aggregate, 45
semi-dynamic box, 46
semi-static box, 46
side loop, 14
similarity relation
– for expressions, 64
simple net, 14
skip, 20
solution
– of recursive equation, 31
solution of recursive equation, 31
sort preservation, 17
sos-operator box, 53
– domain of application, 55

sos-operator boxes, 60
static
– box, 24
– constant, 60
– expression, 60, 73
step, 14
– enabled, 14
– engaged, 16
– executed, 15
– labelled, 15
step sequence
– finite, 15
strong
– bisimulation, 22
– equivalence, 22

T-restricted net, 14
TCSP, 83
token in a marking, 4
transformational relabelling, 13
transition, 13
– enabled, 4
– labelling, 13
– occurrence, 4
– reversible, 55
transition system, 69
– full, 21, 69
– of an expression, 69
– of net, 21
tree equations, 35
ts-isomorphism, 22

underbarring, 5
unmarked net, 21

Var1 – Var5, 58
variable
– e-directed, 58
– ex-exclusive, 58
– recursion, 30
– well formed, 58
– x-directed, 58

weight function, 13
weight of S-invariant, 39
well formed
– expression, 62
– variable, 58

x-directed
– box, 14
– expression, 62
– variable, 58
X-finitary equation, 31

Glossary

$\mathsf{A}_{\mathrm{PBC}}$	10
$^\circ\Sigma$	13
Σ°	13
$\ddot{\Sigma}$	13
$\mathbf{N}$	4
$\stackrel{t}{\longrightarrow}$	4
$\overline{E}$	5
$\underline{E}$	8
$\widehat{a}$	8
τ	8
$\mathsf{Lab}_{\mathrm{PBC}}$	10
$E \| F$	10
E sy A	10
$E \,[]\, F$	10
$\langle E * F]$	10
$E[f]$	11
E rs A	11
$\widehat{A}$	11
$[A : E]$	11
$\stackrel{\mathrm{df}}{=}$	11
Lab	12
ϱ	12
ϱ_α	13
$\varrho_{\mathsf{Lab}'}$	13
ϱ_{id}	13
Σ	13
(S, T, W, λ, M)	13
e, i, x	13
$^\bullet x$	14
$x^\bullet$	14
$\lfloor \Sigma \rfloor$	14
$\overline{\Sigma}$	14
$\underline{\Sigma}$	14
$\Sigma[U\rangle$	15
$M[U\rangle$	15
$M\ [U\rangle\ M'$	15
$\Sigma\ [U\rangle\ \Theta$	15
$\Sigma\ [\Gamma\rangle_{\mathsf{lab}}\ \Theta$	15

$[\sigma\rangle$ 15
$[\Sigma\rangle$ 15
$[M_\Sigma\rangle$ 15
ε 15
$\mathcal{M}$ 16
$[U + V^+ + W^-\rangle$ 16
π 16
$\mathsf{poset}(\pi)$ 16
$\mathsf{Min}(\pi)$ 17
$\mathsf{Max}(\pi)$ 17
$\mathsf{poset}_{\mathsf{ind}}(\sigma)$ 19
ind_Σ 19
$\Sigma\ [\ldots\rangle_{\mathsf{po}}\ \Theta$ 20
skip 20
redo 20
Σ_{sr} 21
ts_Σ 21
fts_Σ 21
$\cong$ 22
$\approx$ 22
iso 23
$\mathsf{iso}_{\mathsf{S}}$ 23
$\mathsf{iso}_{\mathsf{T}}$ 23
$\Sigma\ \mathsf{iso}_{\mathsf{ST}}\ \Theta$ 23
Box^s 25
Box^d 25
Box^e 25
Box^x 25
Box 25
ex 25
Ω 25
$\mathbf{\Sigma}$ 25
$\mathbf{\Sigma}_T$ 25
$\mathsf{P}_{\mathsf{root}}$ 26
$\mathsf{T}_{\mathsf{root}}$ 26
$\mathsf{P}_{\mathsf{tree}}$ 26

$\mathsf{T}_{\mathsf{tree}}$	26
$\Omega(\mathbf{\Sigma})$	26
$S_{\Omega(\mathbf{\Sigma})}$	27
$\mathsf{ST}^{v}_{\mathsf{new}}$	27
$\mathsf{SP}^{s}_{\mathsf{new}}$	27
$\mathsf{trees}^{y}(p)$	27
$M_{\Omega(\mathbf{\Sigma})}$	27
$T_{\Omega(\mathbf{\Sigma})}$	27
$\mathsf{T}^{v}_{\mathsf{new}}$	27
$\mathsf{trees}(u)$	27
$\lambda_{\Omega(\mathbf{\Sigma})}$	27
$W_{\Omega(\mathbf{\Sigma})}$	28
$\sqsubseteq$	30
$\bigsqcup \chi$	30
$\mathcal{X}$	30
$X \stackrel{\text{df}}{=} \Omega_X(\mathbf{\Delta}_X)$	30
Box^{rec}	30
$\mathbf{\Delta}_X[\mathsf{sol}]$	31
$X \stackrel{\text{df}}{=} \Omega(\mathbf{\Delta})$	31
$\mathbf{\Delta}[\Sigma]$	31
$\mathbf{F} : \mathsf{Box}^{rec} \to \mathsf{Box}^{rec}$	31
$\widetilde{\sqsubseteq}$	32
$\mathbf{F}^{*}(\Upsilon)$	32
$\mathbf{F}^{n}(\Upsilon)$	32
$\mathbf{f}$	33
$\mathsf{fix}_{\mathsf{max}}$	35
$\mathsf{fix}_{\mathsf{min}}$	35
θ	39
$\theta(M_{\Sigma})$	39
$\mathsf{SP}^{s}_{\mathsf{new}}[vq]$	41
$\mathsf{subtrees}(p)$	42
∂_s	42
θ_{∂}	42

$\partial_s^{\mathsf{mult}}$ 42
$\mu = (\mu_e, \mu_d, \mu_x, \mu_s)$ 52 55 61
fact_Ω 53
dom_Ω 55
$(\Omega : \mathbf{\Sigma}) \xrightarrow{U} (\Omega : \mathbf{\Theta})$ 56
$\mathbf{\Sigma} \equiv_\Omega \mathbf{\Theta}$ 57
$\mathcal{X}^{wf}$ 58
$\mathsf{box}(c)$ 63
E^{wf}, E^{xcl}, E^{edir}, E^{xdir} 62
$\mathsf{box}(X)$ 63
$\mathsf{box}(\mathsf{op}_\Omega(\mathbf{D}))$, $\mathsf{box}(\overline{E})$, $\mathsf{box}(\overline{E})$ 63
$\equiv$ 64
$\xrightarrow{\emptyset}$ 66
$\xrightarrow{\Gamma}$ 65
$\mathsf{T}_{\mathsf{tree}}^{alg}$ 66
$\mathcal{X}^{xcl}$ 58
$\mathcal{X}^{edir}$ 58
$\mathcal{X}^{xdir}$ 58
Box^{xcl} 58
Box^{edir} 58
Box^{xdir} 58
OpBox 60
ConstBox 60
op_Ω 60
Const^s 60
Const^d 60
Const^e 60
Const^x 60
Expr^e, Expr^d, Expr^x, Expr^s 61
Expr^{box} 61
$\mathsf{lab}(U)$ 66
Con^e, Con^x 68
ind^{alg} 68
$\xrightarrow{\pi}$ 69
fts_G 69
ts_D 69