Asynchonous Box Calculus

Raymond Devillers, Hanna Klaudel, Maciej Koutny, Franck Pommereau

Fundamenta Informaticae 54 (2003) 1–50
IOS Press

Asynchronous Box Calculus

Raymond Devillers
Département d'informatique
Université Libre de Bruxelles
B-1050 Bruxelles, Belgium
rdevil@ulb.ac.be

Hanna Klaudel
LACL, Université Paris 12
61, avenue du général de Gaulle
94010 Créteil, France
klaudel@univ-paris12.fr

Maciej Koutny
Department of Computing Science
University of Newcastle upon Tyne
NE1 7RU, United Kingdom
maciej.koutny@newcastle.ac.uk

Franck Pommereau*
LACL, Université Paris 12
61, avenue du général de Gaulle
94010 Créteil, France
pommereau@univ-paris12.fr

Abstract. The starting point of this paper is an algebraic Petri net framework allowing one to express net compositions, such as iteration and parallel composition, as well as transition synchronisation and restriction. We enrich the original model by introducing new constructs supporting asynchronous interprocess communication. Such a communication is made possible thanks to special 'buffer' places where different transitions (processes) may deposit and remove tokens. We also provide an abstraction mechanism, which hides buffer places, effectively making them private to the processes communicating through them. We then provide an algebra of process expressions, whose constants and operators directly correspond to those used in the Petri net framework. Such a correspondence is used to associate nets to process expressions in a compositional way. That the resulting algebra of expressions is consistent with the net algebra is demonstrated by showing that an expression and the corresponding net generate isomorphic transition systems. This results in the *Asynchronous Box Calculus* (or ABC), which is a coherent dual model, based on Petri nets and process expressions, suitable for modelling and analysing distributed systems whose components can interact using both synchronous and asynchronous communication.

Keywords: Petri nets, process algebra, synchronous and asynchronous communication, structured operational semantics.

* Address for correspondence: LACL, Université Paris 12, 61, avenue du général de Gaulle, 94010 Créteil, France

1. Introduction

This paper is concerned with the theme of relating process algebras, such as CCS [21] and CSP [14], and Petri nets [28]. In general, the approaches proposed in the literature aim at providing a Petri net semantics to process algebras whose definition has been given independently of any Petri nets semantics as, *e.g.*, in [7, 9, 11, 12, 13, 16, 22, 23, 29]. Another approach is to translate elements from Petri nets into process algebras such as ACP [1] as done, *e.g.*, in [2].

A specific framework within which the present paper is set, is the *Petri Box Calculus* (PBC [3]). The PBC has been designed with the aim of allowing a compositional Petri net semantics of nondeterministic and concurrent programming languages [6], and was later extended into a more generic *Petri Net Algebra* (PNA [4, 5]). The model is composed of an algebra of process expressions (called *box expressions*) with a fully compositional translation into labelled safe Petri nets (called *boxes*). When compared with previous works conducted in this framework, the contribution made by the present paper (which itself is an extended version of the short conference communication [10]) is twofold.

First, we extend the correspondence between the operational semantics defined structurally for the process expressions and the denotational semantics defined through the translation in terms of labelled Petri nets to the theory obtained after adding asynchronous communication operators. More precisely, in the standard theory, as developed extensively in [5], the algebra is built from two kinds of operators: (i) the control flow operators which specify that some components of a complex system reach their initial or final state when so do some other components, independently of what has been performed in the past (typical examples here are sequence, choice, parallel composition and iteration operators); and (ii) the synchronous communication operators which allow one to synchronise actions performed by parallel components provided that they satisfy some constraints (a typical example would be the handshake communication of a 'send' action with a corresponding 'receive' action). The latter can also be combined with other operators modifying (renaming) or dropping (restricting) some available actions; this may, for instance, be useful in order to synchronise a component handling some data with a component describing how this data may evolve in time. The control and synchronous communication operators allow for great flexibility in the description of complex causal dependencies in the behaviour of concurrent systems, but they are not well suited to specify some important 'counting' features; for instance, one might want to ensure that the number of iterations performed by one loop is related to the number of iterations performed by another, otherwise independent, loop. Within process algebras based on synchronous communication, this kind of relationship is usually modelled using a specialised expression whose behaviour is in fact that of a shared variable. This may result in overly complex processes *w.r.t.* what is actually represented; *e.g.*, within many existing process algebras (including PBC or PNA) a simple non-terminating producer/consumer system would require a complicated recursive expression to model a store for items exchanged between the two basic components. In order to overcome this difficulty (and several others, *e.g.*, those encountered when dealing with timing features, see [19]) one may introduce asynchronous communication, allowing a component to generate messages (or tokens) which can much later on be received by other components. And, thanks to the explicit asynchronous communication and in contrast to the complicated recursive solution, the producer/consumer system given in the section 2 can be very simple. More generally, a faithful modelling of any data shared by two (or more) components of a concurrent system is more easily modelled by asynchronous communication. In particular, it was shown in [17, 26] that supporting asynchronous communication allows one to give a Petri nets semantics to an expressive parallel programming language in terms of process expressions

instead of ad-hoc nets used previously. Both papers were set in the PBC framework but they did not investigate in detail the resulting framework. In particular, they did not relate the operational semantics of the new kind of process expressions to the semantics obtained through a translation into labelled Petri nets where pending messages are simply modelled using buffer places.

It can be said that the nature of the control flow operators is also in some sense asynchronous, *e.g.*, when putting two components in sequence the start of the second one is determined by the sending of an information about the completion of the first one (in terms of PNA, this is modelled by one or more intermediate places). Such an *implicit* asynchrony is a common feature of process algebras (note that action prefix is an example of sequential composition), but it concerns only the control flow without necessarily providing facilities for asynchronous message passing between different components. In contrast, the feature we focus in this paper is an *explicit* asynchronous communication which can involve components regardless of any control flow structure used to define a system. It will turn out that such a communication is quite simple to implement in terms of Petri nets, which are inherently asynchronous, but requires more effort when treating process expressions and their operational semantics.

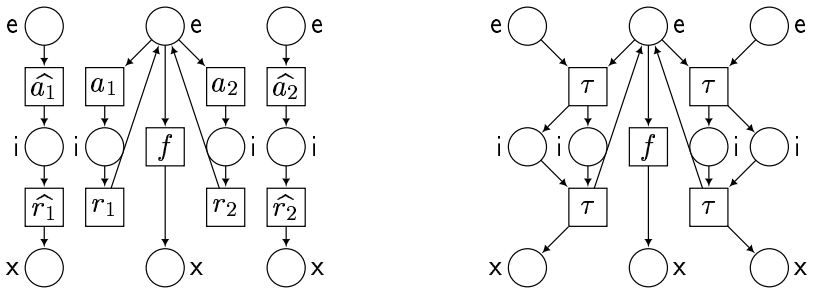
The second contribution of this paper is more technical, in that we relax certain restrictions placed on the application domains of some of the control flow operators, introduced in the standard PNA to ensure that the nets associated with process expressions are always 1-safe. Such restrictions are rather complicated and may be considered artificial since 1-safeness is not a feature of the operational semantics of process expressions. Moreover, since asynchronous communication is based on places which are not 1-safe, retaining those restrictions can hardly be justified in the new framework.

There are two main problems which must be addressed when adding asynchronous communication to a process algebra like PNA: (i) the components are no longer memory-less, *e.g.*, it is possible to remember how many times an event occurred in the previous runs of a component involved in some iterative construct; and (ii) it is necessary to express the persistency and availability of shared (communicated) resources. In the Petri net framework, (ii) can be solved in a simple way by using communication based on shared places, but it is more difficult to find an equally simple solution at the level of process expressions. The approach adopted in this paper is based on an explicit syntactic representation of the available resources (using symbolic markers) together with suitable (and intuitive) operational rules allowing one to move such markers within an expression. Such a device turns out also to solve (i) since resources produced in the past may be kept in a dormant component of a process expression and used later when it is re-activated.

We will now introduce informally the main features of the proposed framework, before proceeding with the more formal aspects of the theory.

1.1. An algebra of nets and process expressions

The variant of the PNA model relevant to this paper considers the following operators: *sequence* $E_1; E_2$ (the execution of E_1 is followed by that of E_2); *choice* $E_1 \Box E_2$ (either E_1 or E_2 can be executed); *parallel composition* $E_1 \| E_2$ (E_1 and E_2 can be executed concurrently); *iteration* $E_1 \circledast E_2$ (E_1 can be executed an arbitrary number of times, and then can be followed by E_2); and *scoping* $E \,\mathsf{sc}\, a$ (all handshake synchronisations involving pairs of a- and $\widehat{a}$-labelled transitions are enforced, and after that the synchronising transitions may no longer be executed). We illustrate these constructs using an example



the box of PREMUTEX the box of MUTEX

Figure 1. Boxes modelling a critical section and two users.

based on three process expressions:

$$\begin{array}{lcl} \text{CRITSECT} & \stackrel{\text{df}}{=} & ((a_1; r_1) \,\Box\, (a_2; r_2)) \circledast f \\ \text{USER}_1 & \stackrel{\text{df}}{=} & \widehat{a_1}; \widehat{r_1} \\ \text{USER}_2 & \stackrel{\text{df}}{=} & \widehat{a_2}; \widehat{r_2} \ , \end{array}$$

modelling a critical section and two user processes. The atomic actions a_1 and a_2 (together with the matching actions $\widehat{a_1}$ and $\widehat{a_2}$) model the granting of access to a shared resource, r_1 and r_2 (together with the matching $\widehat{r_1}$ and $\widehat{r_2}$) model its release, and f models a final action. The system where these three processes operate in parallel is

$$\text{PREMUTEX} \stackrel{\text{df}}{=} \text{USER}_1 \parallel \text{CRITSECT} \parallel \text{USER}_2 \ ,$$

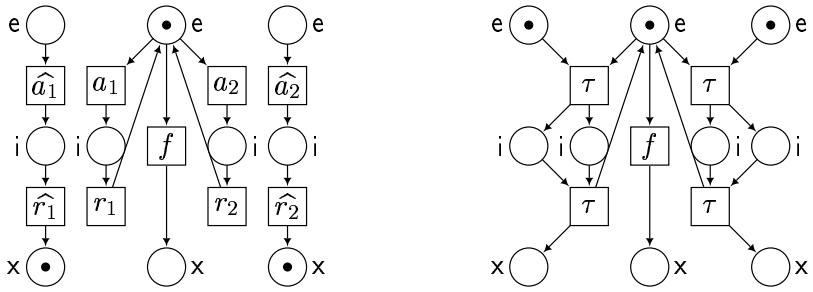
and the net (called a *box*) on the left of figure 1 is the translation of this process expression (itself called a *box expression*). Note that, in a box, places are labelled by their *status* (e for entry, x for exit, and i for internal) while transitions are labelled by CCS-like synchronous communication actions, such as a_1, $\widehat{a_1}$, and τ (similarly as in CCS, τ represents an internal action).

The box expression PREMUTEX and the corresponding box correctly specify the three constituent processes, but it does not allow for interprocess communication. This is, however, easily achieved by applying the scoping *w.r.t.* the synchronisation actions a_i and r_i, which results in the box expression

$$\text{MUTEX} \stackrel{\text{df}}{=} \text{PREMUTEX} \ \mathsf{sc}\, a_1 \ \mathsf{sc}\, a_2 \ \mathsf{sc}\, r_1 \ \mathsf{sc}\, r_2 \ ,$$

and the corresponding box is shown on the right of figure 1.

The operational semantics of box expressions is given through SOS rules in Plotkin's style [25]. However, instead of expressing the evolutions using rules modifying the structure of the expressions, like $a.E \stackrel{a}{\longrightarrow} E$ in CCS, the idea here is to represent the current state of an evolution using overbars and underbars, corresponding respectively to the initial and final states of (sub)expressions. This is illustrated in figure 2, where the net on the left represents PREMUTEX after the two user processes have terminated, and the critical section is still in its initial state, while the net on the right represents the initial state of the system specified by MUTEX, *i.e.*, that corresponding to the box expression $\overline{\text{MUTEX}}$.



the box of $\underline{\text{USER}_1} \| \overline{\text{CRITSECT}} \| \underline{\text{USER}_2}$ the box of $\overline{\text{MUTEX}}$

Figure 2. Boxes corresponding to box expressions with underbars and overbars.

There are two kinds of SOS rules: equivalence rules specifying when two distinct expressions denote the very same state, *e.g.*, one can derive that

$$\overline{\text{USER}_1 \| \text{CRITSECT} \| \text{USER}_2} \equiv \overline{\text{USER}_1} \| \overline{\text{CRITSECT} \| \text{USER}_2} \equiv \overline{\text{USER}_1} \| \overline{\text{CRITSECT}} \| \overline{\text{USER}_2} \ ,$$

and evolution rules specifying when we may have a state change, *e.g.*, one can derive that

$$((a_1; r_1) \,\square\, (a_2; r_2)) \circledast \overline{f} \xrightarrow{\{f\}} ((a_1; r_1) \,\square\, (a_2; r_2)) \circledast \underline{f} \ .$$

As it was shown in [5, 20], the two algebras constituting PNA are fully compatible, in the sense that a box expression and the corresponding box generate isomorphic transition systems.

1.2. Asynchronous communication

Recently, [18] (and then[1] [10]) introduced a novel feature into the box algebra, aimed at the modelling of asynchronous interprocess communication (such an extension was used, in particular, to model time-dependent concurrent systems). The basic devices facilitating this are new kinds of basic actions, for sending, receiving and testing for the presence of tokens in buffer places.[2] We introduce all three using the following very simple process expressions, modelling a producer, consumer and tester processes (each process can perform exactly *one* action, after which it terminates):

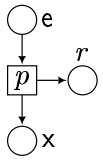
$$\begin{aligned} \text{PRODONE} &\stackrel{\text{df}}{=} pr^+ \\ \text{CONSONE} &\stackrel{\text{df}}{=} cr^- \\ \text{TESTONE} &\stackrel{\text{df}}{=} tr^{\pm} \ . \end{aligned}$$

In the above, pr^+ is an atomic action whose role is to 'produce' a token (resource) and deposit it in a buffer place identified by r; in doing so, it generates the visible label p. Similarly, cr^- is an atomic action

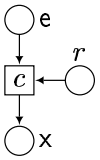
[1] Note that, with respect to [18], the notations and technical details are slightly different in [10], but the principles remain the same. In this paper, we adopt the notations from [10].

[2] Buffer places are a well-known technique for modelling asynchronous communication in Petri nets [27]; here we introduce them to the domain of Petri boxes.

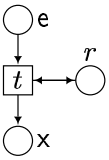
which 'consumes' a resource from buffer r, generating the visible label c. And $tr^{\pm}$ combines the effect of the two previous atomic actions, effectively 'testing' for the presence of a token in the buffer while generating a visible label t. The Petri nets rendering of the above three processes are shown in figure 3, where the buffer places are identified by the r labels.



the box of PRODONE



the box of CONSONE



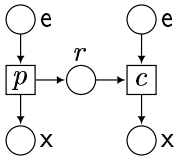
the box of TESTONE

Figure 3. Basic asynchronous communication actions (note that the double-headed arrow represents self-loop).

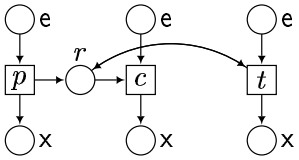
In general, some transitions can insert tokens into a buffer place (perhaps several in a single step), while others can (later) remove them, thus effecting a form of asynchronous communication. We can, for example, compose in parallel the simple producer and consumer processes, forming a new box expression SYST $\stackrel{\text{df}}{=}$ PRODONE$\|$CONSONE. The corresponding box is shown on the left of figure 4, where the original parallel composition of PNA has been modified so that the two r-labelled buffer places are merged into a single one, also r-labelled. As a consequence, this place can later be merged with other similar buffer places, *e.g.*, we might form TESTSYST $\stackrel{\text{df}}{=}$ SYST$\|$TESTONE with the corresponding box shown in the middle of figure 4. The reader may easily verify that the asynchronous communication outlined above is now indeed feasible, after placing one token in each of the e-labelled places of the box of SYST (or TESTSYST).

The scheme described thus far still needs an abstraction mechanism for the asynchronous communication part, which comes in the form of the *buffer restriction* operator, denoted by tie. The tie operator *w.r.t.* a buffer r changes the status of the r-labelled buffer place into a b-labelled one, indicating that the place can no longer be merged with other buffer places (it should be stressed that b is a place *status* — like e, x and i). In effect, the b-labelled places may be viewed as internal places. This is illustrated on the right of figure 4, for the box expression HIDESYST $\stackrel{\text{df}}{=}$ SYST tie r.

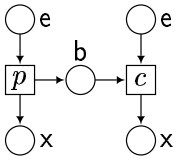
The resulting model, called the *Asynchronous Box Calculus* (or ABC), is no longer based on safe Petri nets since, in particular, one cannot expect the buffer places to be safe; indeed, they may even be unbounded. However, the extension is such that no auto-concurrency is ever generated (*i.e.*, no transition can be executed concurrently with itself), as further discussed below.



the box of SYST



the box of TESTSYST



the box of HIDESYST

Figure 4. Asynchronous communication and buffer restriction.

1.3. About this paper

Asynchronous communication was introduced into the PBC framework in [18], where it was used to model time-dependent systems, and then in [17, 26], where it was used to give an algebraic semantics to a parallel programming language. However, its full impact on the theory of boxes and box expressions was not considered, and a first step in this direction was made in the conference paper [10]. The main aim of this paper is to develop a comprehensive presentation and analysis of the ABC model and, apart from introducing devices supporting asynchronous interprocess communication, we will remove the restriction of the standard PNA theory to consider only 1-safe boxes (such a property was maintained in order to support a straightforward concurrency semantics based on causal partial orders). The 1-safeness will be replaced by auto-concurrency freeness which is a property guaranteeing powerful concurrency semantics in terms of event structures, as shown in [15], and thus ensuring that the resulting model is still highly relevant from a theoretical point of view. As a result, and unlike in [10], we will allow parallel composition to be present directly under iteration.

The model will be based on two algebras: a (syntactical) algebra of box expressions and a (semantical) algebra of boxes. The latter is in fact a Petri net interpretation of the former, thus providing a denotational semantics for it. On the technical level, we will introduce specific syntactic constructs manipulating the tokens in buffer places, and the buffer restriction operator. The original operational semantics rules will be augmented, yielding a system of SOS rules providing the operational semantics of ABC expressions. The two algebras forming ABC will be related through a mapping which, for any box expression, will return a corresponding box with an isomorphic transition system.

This paper will not address recursion which allows one to use expressions such as $X \stackrel{\text{df}}{=} \cdots$ where X can appear recursively in its definition (directly or indirectly). Such an extension is left for future research. However, this is not an important limitation since recursion is a very special feature which often leads to infinite nets and thus does not fall into the scope of the present framework. Moreover, in many cases infinite behaviours needing recursion in classical process algebras may be realised with iteration provided here.

The structure of the paper is as follows. The next section presents three examples of systems employing asynchronous communication. Section 3 describes the class of labelled nets on which ABC is based. In section 4, we introduce the net algebra part of ABC; in particular, we define a number of operators, either directly, or by using an auxiliary net substitution meta-operator. Section 5 investigates the relationship between the structure and behaviour of composite nets. In section 6, we introduce an algebra of process expressions which forms the second part of the ABC model. In particular, we define a translation from expressions to nets, and an operational semantics of process expressions both in terms of steps of transitions of the corresponding boxes and their labels. We also present there our main result that a box expression and the corresponding box generate isomorphic transition systems.

To streamline the presentation, the appendix contains auxiliary results and proofs as well as generalisations of the results presented in the main body of the paper.

2. Three examples

In order to provide more intuition about the way the ABC algebra is defined and used, this section presents further examples built upon three iteration-based processes, respectively modelling a producer,

consumer and tester of some resource:

$$\begin{array}{lcll} \textsc{Prod} & \stackrel{\text{df}}{=} & \textsc{ProdOne} \circledast f & = & pr^{+} \circledast f \\ \textsc{Cons} & \stackrel{\text{df}}{=} & \textsc{ConsOne} \circledast f & = & cr^{-} \circledast f \\ \textsc{Test} & \stackrel{\text{df}}{=} & \textsc{TestOne} \circledast f & = & tr^{\pm} \circledast f \ , \end{array}$$

where PRODONE, CONSONE and TESTONE are as in the introduction. For example, since PRODONE constitutes the iterative part of PROD, the latter can repeatedly send a token to a r-labelled buffer place. Tokens produced in such a way can then be removed, also repeatedly, by the CONSONE part of the CONS process when, *e.g.*, PROD and CONS operate in parallel. The f is in each case a finishing action. All three examples have been chosen for purely illustrative reasons: more interesting, yet also more complex, examples showing the interplay of the asynchronous operators, the control flow operators and the synchronous operators can be found, for instance, in [18, 19].

Example I. The example, given on the left of figure 5, models a concurrent system composed of two producers and one consumer operating in parallel:

$$\textsc{Syst}_{\text{I}} \stackrel{\text{df}}{=} s; ((\textsc{Prod} \parallel \textsc{Prod} \parallel \textsc{Cons}) \,\mathsf{tie}\, r) \ .$$

The two-producers/one-consumer system is encapsulated by an application of the buffer restriction operator. This makes the buffer place b-labelled, and so no longer available for merging with other buffer places. The whole system is preceded by a 'startup' action s.

Example II. The encapsulating feature of buffer restriction is further illustrated by the second example:

$$\textsc{Syst}_{\text{II}} \stackrel{\text{df}}{=} (\textsc{ProdOne}; ((\textsc{Prod} \| \textsc{Cons}) \,\mathsf{tie}\, r); \textsc{ConsOne}) \,\mathsf{tie}\, r \ ,$$

shown in the middle of figure 5. This example will be used to illustrate the result of a nested application of buffer restriction.

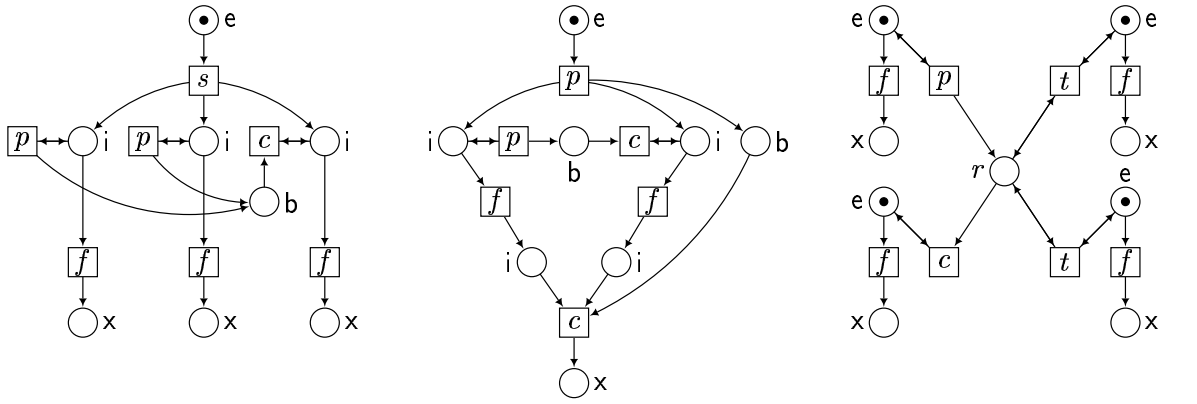
Example III. The example on the right of figure 5 shows a system with one producer, one consumer and two tester processes whose role is to check for the presence of tokens in the buffer:

$$\textsc{Syst}_{\text{III}} \stackrel{\text{df}}{=} \textsc{Prod} \| \textsc{Cons} \| \textsc{Test} \| \textsc{Test} \ .$$

Such an example allows one to illustrate a concurrent access to the buffer place. Notice that this system is not closed *w.r.t.* asynchronous communications; it would be necessary to apply buffer restriction on r in order to make private (*i.e.*, b-labelled) the r-labelled buffer place.

2.1. Three system evolutions

We will now describe three evolutions, or execution scenarios, for the systems given above. In each case, we will use step sequences as a formal device for capturing concurrent behaviours, and assume that the system starts from its implicit initial state, *e.g.*, we consider $\overline{\textsc{Syst}_{\text{I}}}$ rather than $\textsc{Syst}_{\text{I}}$. There are two particular aspects we intend to illustrate: the naming mechanism for buffer places which means that buffers with the same label are shared and not renamed as a result of various compositions, and the fact that the capability to access a buffer in any way from the outside disappears after it has been hidden.



Σ_I — the box of $\overline{\text{SYST}_\text{I}}$ Σ_{II} — the box of $\overline{\text{SYST}_\text{II}}$ Σ_{III} — the box of $\overline{\text{SYST}_\text{III}}$

Figure 5. Boxes for the three execution scenarios.

Scenario I. Consider the net Σ_I in figure 5, also shown in figure 6 with its transition names, and the following evolution:

- the system is started up by executing the s-labelled transition;
- the two producers send a token each to the b-labelled buffer place;
- the consumer takes one of the two tokens from the buffer place and, at the same time, the first producer sends there another token;
- the two producers and the consumer finish their operation by simultaneously executing the three f-labelled transitions.

Such a scenario corresponds to the following step sequence:

$$\Sigma_I \, [\{t_1\}\{t_2, t_3\}\{t_2, t_4\}\{t_5, t_6, t_7\}\rangle \, \Sigma_I' \ ,$$

where Σ_I' is Σ_I with two tokens in the buffer place, one token in each of the x-labelled places, and no token elsewhere. In terms of labelled step sequences, the scenario is represented by

$$\Sigma_I \, [\{s\}\{p, p\}\{p, c\}\{f, f, f\}\rangle \, \Sigma_I' \ .$$

Since each x-labelled place has exactly one token, we will say that Σ_I' is in a final marking (or state). It may be observed that in any possible evolution, and at any instant, the number of c's executed is less or equal to the number of p's previously performed. This is an example of a counting feature facilitated by asynchronous communication.

In the next two examples, we only consider labelled steps as these are usually sufficient to capture the relevant behavioural information. However, in order to obtain the desired properties of boxes and box expressions, the formal reasoning will have to be carried out first using the more detailed view provided by transition steps.

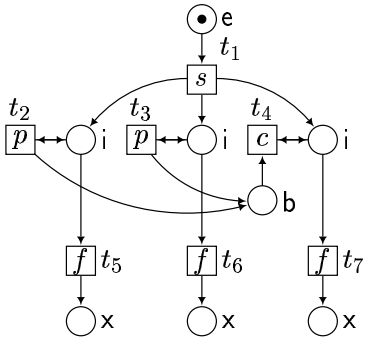


Figure 6. The box of $\overline{\text{SYST}_\text{I}}$ with transition names for the first execution scenario.

Scenario II. Consider the net Σ_{II} in figure 5, and the following evolution:

- the system begins by executing the topmost p-labelled transition, and puts a token in the outer buffer place;
- the producer sends twice in a row one token to the inner buffer place;
- the consumer takes one of the two tokens from the inner place;
- the producer and the consumer finish their operation by simultaneously executing the f-labelled transitions;
- the system finishes by firing the bottom c-labelled transition, which removes the token from the outer buffer place.

Such a scenario corresponds to the following labelled step sequence:

$$\Sigma_{II}\,[\{p\}\{p\}\{p\}\{c\}\{f,f\}\{c\}\rangle\,\Sigma'_{II}\ ,$$

where Σ'_{II} is Σ_{II} with one token in the internal buffer place and the x-labelled place, and no token elsewhere, so that Σ'_{II} is in a final state.

The buffer hiding feature was crucial in [17] to give the semantics of a programming language in which variables local to a declaring block may be re-declared in a nested block, leading to the hiding of the former declarations. In the proposed solution, the value held by a variable is stored in a buffer place local to the net which gives the semantics of the declaring block. The relevant fragment of the resulting box is then very similar to Σ_{II}.

Scenario III. Consider the net Σ_{III} in figure 5, and the following evolution:

- the producer sends in a row two tokens to the buffer place;
- in a single step, the consumer removes one token, the first tester checks for the presence of the other one, and the producer sends another token to the buffer place;
- both testers check for the presence of tokens in parallel.

Such a scenario corresponds to the following labelled step sequence:

$$\Sigma_{III}\ [\{p\}\{p\}\{c,t,p\}\{t,t\}\rangle\ \Sigma'_{III}\ ,$$

where Σ'_{III} is Σ_{III} with two additional tokens in the buffer place. Notice that Σ'_{III} is *not* in a final state.

3. Preliminaries

In this section, we present a number of definitions used throughout the paper.

3.1. Multisets

A *multiset* over a set X is a function $\mu : X \to \mathbb{N}$, where $\mathbb{N} = \{0, 1, 2, \ldots\}$. We denote by $\mathsf{mult}(X)$ the set of all finite multisets μ over X, *i.e.*, those satisfying $\sum_{x\in X} \mu(x) < \infty$. We will write $\mu \leq \mu'$ if the domain X of μ is included in that of the multiset μ', and $\mu(x) \leq \mu'(x)$, for all $x \in X$. An element $x \in X$ belongs to μ, denoted $x \in \mu$, if $\mu(x) > 0$. The sum and difference of multisets, and the multiplication by a non-negative integer are respectively denoted by $+$, $-$ and $\cdot$ (the difference will only be applied when the second argument is smaller or equal to the first one). A subset of X may be treated as a multiset over X, by identifying it with its characteristic function, and a singleton set can be identified with its sole element. A finite multiset μ over X may be written as

$$\sum_{x\in X} \mu(x) \cdot x \quad \text{or} \quad \sum_{x\in X} \mu(x) \cdot \{x\}\ ,$$

as well as in extended set notation, *e.g.*, $\{a, a, \tau\}$ denotes a multiset μ such that $\mu(a) = 2$, $\mu(\tau) = 1$ and $\mu(x) = 0$ for all $x \in X \setminus \{a, \tau\}$.

3.2. Actions, buffers and synchronisation

We assume that there is a set $\mathbb{A}$ of (atomic) *actions* representing synchronous interface activities used, in particular, to model handshake communication. We will also assume that, similarly as in CCS, $\mathbb{A}_\tau \stackrel{\text{df}}{=} \mathbb{A} \uplus \{\tau\}$ and for every $a \in \mathbb{A}$, $\widehat{a}$ is an action in $\mathbb{A}$ such that $\widehat{\widehat{a}} = a$. In addition to the set of atomic actions, there is a finite set $\mathbb{B}$ of *buffer symbols* (or buffers) for asynchronous interprocess communications.[3]

We need some mechanism to express action synchronisation and the result of it. For this purpose, we will use (communication) interface functions, that are partial functions $\varphi : \mathsf{mult}(\mathbb{A}_\tau)\backslash\{\varnothing\} \to \mathbb{A}_\tau$. In particular, to introduce forced CCS-like synchronisation for each $a \in \mathbb{A}$ we will have $\varphi_{\mathsf{sc}\,a}$ (called scoping interface function) whose domain is $\{\{c\} \mid c \in \mathbb{A}_\tau \setminus \{a, \widehat{a}\}\} \cup \{\{a, \widehat{a}\}\}$ and such that $\varphi_{\mathsf{sc}\,a}(\{c\}) \stackrel{\text{df}}{=} c$ for all $c \in \mathbb{A}_\tau \setminus \{a, \widehat{a}\}$ and $\varphi_{\mathsf{sc}\,a}(\{a, \widehat{a}\}) \stackrel{\text{df}}{=} \tau$. Another commonly used interface function is the identity φ_{id} with the domain $\{\{a\} \mid a \in \mathbb{A}_\tau\}$ and such that $\varphi_{id}(\{a\}) \stackrel{\text{df}}{=} a$, for all $a \in \mathbb{A}_\tau$.

[3] The finiteness of $\mathbb{B}$ is not essential, but it will allow us to consider only finite nets, as for technical reasons a box will contain a buffer place for each element of $\mathbb{B}$. We shall assume that $\mathsf{e}, \mathsf{x}, \mathsf{i}, \mathsf{b} \notin \mathbb{B}$.

3.3. Labelled nets

A *(marked) labelled net* is, in the present framework, a tuple $\Sigma \stackrel{\text{df}}{=} (S, T, W, \lambda, M)$ 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 $\mathbb{N}$;
- λ is a *labelling* for places and transitions such that $\lambda(s)$ is a symbol in $\{\mathsf{e}, \mathsf{i}, \mathsf{x}\} \uplus (\mathbb{B} \uplus \{\mathsf{b}\})$, for every place $s \in S$, and $\lambda(t)$ is an interface function φ or an action in $\mathbb{A}_\tau$, for every transition $t \in T$;
- M is a *marking*, *i.e.*, a multiset over S (in other words, a mapping from the set of places S to $\mathbb{N}$).

Moreover, Σ is *finite* if both S and T are finite sets, and it is *simple* if W always returns 0 or 1.

If the labelling of a place s is e, i or x, then s is an *entry*, *internal* or *exit* place, respectively. If the labelling is b then s is a *closed buffer* place, and if it is a buffer symbol $b \in \mathbb{B}$, then s is an *open buffer* place. Collectively, the e-, i- and x–labelled places are called *control (flow)* places. Moreover, the set of all entry (resp. exit) places will be denoted by ${}^\circ\Sigma$ (resp. Σ°). We shall also use M^{ctr} and M^{opb} to denote M restricted to the control places and to the open buffer places, respectively. If $M^{ctr} = {}^\circ\Sigma$, we will say that Σ is in an *initial* state (or marking), and if $M^{ctr} = \Sigma^\circ$, we will say that Σ is in a *final* state (or marking). Finally, $\langle\Sigma\rangle$ is the labelled net obtained from Σ by removing all the tokens from open and closed buffer places and all the arcs adjacent to them. Note that $\langle\langle\Sigma\rangle\rangle = \langle\Sigma\rangle$. The $\langle\cdot\rangle$ notation extends component-wise to tuples of labelled nets.

We adopt the standard rules about representing nets as directed graphs; moreover, double-headed arrows will represent self-loops (where a *self-loop* is a pair of arcs between two nodes pointing in the opposite directions). To avoid ambiguity, we will sometimes decorate the various components of Σ with the index Σ; *e.g.*, T_Σ will denote the set of transitions of Σ. In order to simplify diagrams, we will omit isolated unmarked buffer places.

For every place (transition) x, we use ${}^\bullet x$ to denote its pre-set, *i.e.*, the set of all transitions (places) y such that there is an arc from y to x, *i.e.*, $W(y, x) > 0$. The post-set $x^\bullet$ is defined in a similar way. The pre- and post-set notation extends in the usual way to sets X of places and transitions, *e.g.*, ${}^\bullet X \stackrel{\text{df}}{=} \bigcup_{x \in X} {}^\bullet x$.

3.4. Step sequences

A finite step sequence semantics of a labelled net Σ captures 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 by $\mathsf{enabled}(\Sigma)$ the set of all steps enabled by Σ; notice that we always have $\varnothing \in \mathsf{enabled}(\Sigma)$. A step $U \in \mathsf{enabled}(\Sigma)$ can be *executed*, leading to a marking M' given, for every place $s \in S$, by

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

We will denote this by $\Sigma[U\rangle\Sigma'$, where Σ' is Σ with the marking changed to M'. Transition labelling may be extended to steps, through the formula

$$\lambda(U) \stackrel{\text{df}}{=} \sum_{t\in U} U(t)\cdot\lambda(t)\ .$$

In particular, we will denote $\Sigma\,[\Gamma\rangle_\lambda\,\Sigma'$ whenever there is a multiset of transitions U such that $\Sigma\,[U\rangle\,\Sigma'$ and $\Gamma = \lambda(U)$. This allows one to translate various behavioural notions defined in terms of multisets of transitions into notions based on multisets of transition labels (or *labelled steps*). Although we will use the same term 'step' to refer both to a finite multiset of transitions and to a finite multiset of labels, it will always be clear from the context which one is actually meant. It may happen that two different transition steps correspond to the same labelled step, when different transitions have the same label (see, for instance, the step $\{p, c\}$ in scenario I for which p might have equally well come from the first or from the second producer).

A *step sequence* of Σ is a (possibly empty) sequence of steps, $\omega = U_1 \ldots U_k$, such that there are nets $\Sigma_1, \ldots, \Sigma_k$ satisfying $\Sigma\,[U_1\rangle\,\Sigma_1\,[U_2\rangle\,\Sigma_2 \cdots [U_k\rangle\,\Sigma_k$. We will denote this by $\Sigma\,[\omega\rangle\,\Sigma_k$ or $\Sigma_k \in [\Sigma\rangle$, and call Σ_k *derivable* from Σ and its marking M_{Σ_k} *reachable* from M_Σ (with the convention that $\Sigma\,[\omega\rangle\,\Sigma$ if $k = 0$, *i.e.*, if ω is the empty step sequence). The definition of a *labelled step sequence* of Σ is similar.

3.5. Clean, ac-free and quasi-safe markings

A marking M of a labelled net Σ is:

- *clean* if $M^{ctr} \geq {}^\circ\Sigma \Rightarrow M^{ctr} = {}^\circ\Sigma$ and $M^{ctr} \geq \Sigma^\circ \Rightarrow M^{ctr} = \Sigma^\circ$.
- *ac-free* if, for every transition t, there is a control place $s \in {}^\bullet t$ such that $M(s) < 2\cdot W(s,t)$, which means that the marking of the control places does not allow *auto-concurrency*.
- *quasi-safe* if, for every transition t, there is a control place $s \in {}^\bullet t$ such that $M(s) \leq 1$; note that this implies ac-freeness.

4. An asynchronous algebra of boxes

To model concurrent systems, we will use a class of labelled nets called *asynchronous boxes*. To model operations on such nets, we will use another class of labelled nets, called *operator boxes*, and the net substitution meta-operator (called also refinement [4]), which allows one to substitute transitions in an operator box by possibly complex asynchronous boxes.[4]

4.1. Asynchronous boxes and marking operators

An *asynchronous box* (or, shortly, a box) is a labelled net Σ such that each transition is labelled by an action in $\mathbb{A}_\tau$, and Σ itself is:

- *ex-restricted*: there is at least one entry place and at least one exit place;

[4]Note that one could introduce the two abstraction operators (*i.e.*, scoping and tie) without using operator boxes as these are particularly simple (see fi gure 8); however, treating them in the same way as other operators leads to more uniform presentation.

- $\mathbb{B}$-*restricted*: for every $b \in \mathbb{B}$, there is exactly one b-labelled place;
- *control-restricted*: for every transition t there is at least one control place in ${}^\bullet t$, and at least one control place in $t^\bullet$.

Let Σ be a box and $\Theta = \langle\Sigma\rangle$. Then Σ is *static* (resp. *dynamic*) if $M_\Theta = \varnothing$ (resp. $M_\Theta \neq \varnothing$) and all the markings reachable from ${}^\circ\Theta$ or Θ° (resp. from M_Θ or ${}^\circ\Theta$ or Θ°) in Θ are both clean and ac-free. Moreover, if all such markings are quasi-safe then Σ is itself called *quasi-safe*. The asynchronous boxes, static boxes and dynamic boxes will respectively be denoted by abox, abox^{stc} and abox^{dyn}. In what follows, we will only consider finite asynchronous boxes and operator boxes.

Proposition 4.1. Let Σ be a labelled net and $\Theta = \langle\Sigma\rangle$.

1. Σ is a box *iff* Θ is a box.
2. Σ is a static (dynamic) box *iff* Θ is a static (dynamic) box.
3. Σ is quasi-safe *iff* Θ is quasi-safe, whenever Σ is a static or dynamic box.
4. If $\Sigma[U\rangle\Sigma'$, then $\Theta[U\rangle\Theta'$, where $\Theta' = \langle\Sigma'\rangle$.
5. If $\Theta[U\rangle\Theta'$, then $\Sigma'[U\rangle\Sigma''$, for some Σ' and Σ'' such that $\langle\Sigma'\rangle = \Theta$ and $\langle\Sigma''\rangle = \Theta'$.
6. If Σ is a dynamic box, then all the markings reachable from M_Σ are both clean and ac-free. Moreover, if Σ is quasi-safe, then all such markings are quasi-safe.

Proof:
(1) follows directly from the definition of a box (note that control-restrictedness is not affected by removing the arcs adjacent to the buffer places); (2) and (3) follow from $\langle\Theta\rangle = \langle\langle\Sigma\rangle\rangle = \langle\Sigma\rangle$; (4) and (5) follow directly from the definitions; and (6) follows from (4). □

Proposition 4.1(6) captures the central behavioural property of the nets treated within the ABC model. However, taking the first part of proposition 4.1(6) as the definition of a 'dynamic' box would not have worked as the property is not preserved through various net compositions introduced later on. With the stronger (though more complicated) definition we adopted, it will be possible to show that being a static or dynamic box is preserved in the compositional net model contained in ABC.

Our next result is that being a static or dynamic box is invariant over any possible evolution.

Proposition 4.2. Let Σ be a labelled net and $\Sigma[U\rangle\Sigma'$.

1. If Σ is a static box, then $U = \varnothing$ and $\Sigma = \Sigma'$.
2. If Σ is a dynamic (and quasi-safe) box, then U is a set of transitions[5] and Σ' is a dynamic (and quasi-safe) box.

[5]But if $\Sigma\ [\Gamma\rangle_\lambda\ \Sigma'$, then the labelled step Γ may be a true multiset of actions; see, *e.g.*, scenario I in section 2.

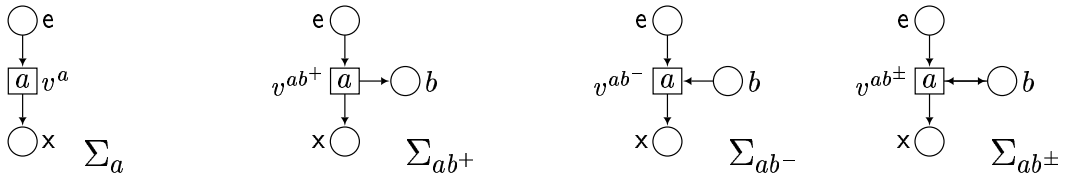


Figure 7. Basic asynchronous boxes used in ABC, where $a \in \mathbb{A}_\tau$ and $b \in \mathbb{B}$.

Proof:
(1) That $U = \varnothing$ follows from $\Sigma^{ctr} = \langle\Sigma\rangle^{ctr} = \varnothing$ and the control-restrictedness of any box. Moreover, $\Sigma = \Sigma'$ follows from $U = \varnothing$.

(2) That U is a set of transitions follows from the ac-freeness of M_Σ in proposition 4.1(6) and the control-restrictedness of any box. We then observe that, by proposition 4.1(4), we have $\Theta[U\rangle\Theta'$, where $\Theta = \langle\Sigma\rangle$ and $\Theta' = \langle\Sigma'\rangle$. The rest of the result then follows from the definitions as well as ${}^\circ\Theta' = {}^\circ\Theta$ and $(\Theta')^\circ = \Theta^\circ$ and the fact that all the markings reachable from $M_{\Theta'}$ are also reachable from M_Θ. □

Although arbitrarily complex asynchronous boxes could be defined directly, we are interested in their modular construction by applying operator boxes to suitable operand boxes. The basic building blocks we shall use in this paper for such a construction are the boxes Σ_α, shown in figure 7, where $\alpha \in \mathcal{A} \stackrel{\text{df}}{=} \mathbb{A}_\tau \uplus \{\, ab^+ , ab^- , ab^\pm \mid a \in \mathbb{A}_\tau \wedge b \in \mathbb{B}\}$; it may easily be checked that all these boxes are static and quasi-safe. All the remaining static and dynamic boxes of the Asynchronous Box Calculus will be constructed from the Σ_α's through various operators.

The first kind of net operators we introduce are three *marking operators*, which modify the marking of a box Σ, where $b \in \mathbb{B}$ and $B \in \mathsf{mult}(\mathbb{B})$:

- $\Sigma.B$ adds $B(b)$ tokens to the b-labelled open buffer place of Σ, for each $b \in \mathbb{B}$; in particular, $\Sigma.b \stackrel{\text{df}}{=} \Sigma.\{b\}$ adds one token to the b-labelled buffer place of Σ. This operation will be called *buffer stuffing*.
- $\overline{\Sigma}$ (resp. $\underline{\Sigma}$) is Σ with one additional token in each entry (resp. exit) place, *i.e.*, $M_{\overline{\Sigma}} \stackrel{\text{df}}{=} M_\Sigma + {}^\circ\Sigma$ (resp. $M_{\underline{\Sigma}} \stackrel{\text{df}}{=} M_\Sigma + \Sigma^\circ$).
- $\lfloor\Sigma\rfloor$ is Σ with all the tokens in its control places removed, and $\lceil\Sigma\rceil$ is Σ with the empty marking. Both notations extend component-wise to tuples of boxes.

Proposition 4.3. Let Σ be a box and $B, B' \in \mathsf{mult}(\mathbb{B})$.

1. $\Sigma.\varnothing = \Sigma$, $\Sigma.B.B' = \Sigma.(B + B')$, $\overline{\Sigma}.B = \overline{\Sigma.B}$ and $\underline{\Sigma}.B = \underline{\Sigma.B}$.
2. $\lceil\lceil\Sigma\rceil\rceil = \lceil\lfloor\Sigma\rfloor\rceil = \lfloor\lceil\Sigma\rceil\rfloor = \lceil\Sigma\rceil$.
3. $\lfloor\Sigma\rfloor.B = \lfloor\Sigma.B\rfloor$, $\lfloor\overline{\Sigma}\rfloor = \lfloor\underline{\Sigma}\rfloor = \lfloor\lfloor\Sigma\rfloor\rfloor = \lfloor\Sigma\rfloor$ and $\lceil\Sigma.B\rceil = \lceil\overline{\Sigma}\rceil = \lceil\underline{\Sigma}\rceil = \lceil\Sigma\rceil$.
4. $\langle\Sigma.B\rangle = \langle\Sigma\rangle$, $\langle\overline{\Sigma}\rangle = \overline{\langle\Sigma\rangle}$, $\langle\underline{\Sigma}\rangle = \underline{\langle\Sigma\rangle}$, $\langle\lfloor\Sigma\rfloor\rangle = \lfloor\langle\Sigma\rangle\rfloor$ and $\langle\lceil\Sigma\rceil\rangle = \lceil\langle\Sigma\rangle\rceil$.
5. Σ is static (and quasi-safe) *iff* $\Sigma.B$ is static (and quasi-safe).

6. Σ is dynamic (and quasi-safe) *iff* $\Sigma.B$ is dynamic (and quasi-safe).
7. $\overline{\Sigma}$ is dynamic (and quasi-safe) *iff* Σ is static (and quasi-safe) *iff* $\underline{\Sigma}$ is dynamic (and quasi-safe).
8. If Σ is static or dynamic (and quasi-safe), then $\lfloor\Sigma\rfloor$ and $\lceil\Sigma\rceil$ are static (and quasi-safe) boxes.

Proof:
(1,2,3,4) follow directly from the definitions, and (5,6) follow directly from the definitions and from the first equality in part (4).

(7) We will prove that $\overline{\Sigma}$ is dynamic *iff* Σ is static (the other parts of the result can be shown in a similar way). Suppose that $\overline{\Sigma}$ is a dynamic box. Then $M_{\langle\overline{\Sigma}\rangle}$ is a clean marking. Since, by part (4), we have $M_{\langle\overline{\Sigma}\rangle} = M_{\overline{\langle\Sigma\rangle}}$, it follows that $M_{\langle\Sigma\rangle} = \varnothing$. This, together with the fact that $\overline{\Sigma}$ is dynamic and ${}^\circ\langle\Sigma\rangle = {}^\circ\langle\overline{\Sigma}\rangle$ and $\langle\Sigma\rangle^\circ = \langle\overline{\Sigma}\rangle^\circ$, means that Σ is a static box. Suppose now that Σ is a static box. Then $M_{\langle\Sigma\rangle} = \varnothing$ and so $M_{\overline{\langle\Sigma\rangle}} = {}^\circ\langle\Sigma\rangle$. Thus, by part (4) and the ex-restrictedness of any box, $M_{\langle\overline{\Sigma}\rangle} = {}^\circ\langle\Sigma\rangle \neq \varnothing$. This, together with the fact that Σ is static and $M_{\langle\overline{\Sigma}\rangle} = {}^\circ\langle\overline{\Sigma}\rangle = {}^\circ\langle\Sigma\rangle$ and $\langle\overline{\Sigma}\rangle^\circ = \langle\Sigma\rangle^\circ$, means that $\overline{\Sigma}$ is a dynamic box.

(8) Follows from the definitions and the fact that $M_{\langle\lfloor\Sigma\rfloor\rangle} = M_{\langle\lceil\Sigma\rceil\rangle} = \varnothing$. □

4.2. Transition systems

The complete behaviour of a static or dynamic box can be represented by a transition system. And, since we have two kinds of possible steps, we introduce two kinds of transition systems.

The *full transition system* of a dynamic box Σ is $\mathsf{fts}_\Sigma \stackrel{\text{df}}{=} (V, L, A, init)$ where $V \stackrel{\text{df}}{=} [\Sigma\rangle$ is the set of states; $L \stackrel{\text{df}}{=} 2^{T_\Sigma}$ is the set of arc labels; $A \stackrel{\text{df}}{=} \left\{(\Sigma', U, \Sigma'') \in V \times L \times V \mid \Sigma'\,[U\rangle\,\Sigma''\right\}$ is the set of arcs; and $init \stackrel{\text{df}}{=} \Sigma$ is the initial state. For a static box Σ, $\mathsf{fts}_\Sigma \stackrel{\text{df}}{=} \mathsf{fts}_{\overline{\Sigma}}$.

The *labelled transition system* of a static or dynamic box Σ, denoted by lts_Σ, is derived from fts_Σ by changing each arc label U to $\lambda_\Sigma(U)$. Note that several different arcs between two states in fts_Σ may lead to a single arc in lts_Σ.

As usual, if $(V, L, A, init)$ and $(V', L', A', init')$ are two transition systems, then an *isomorphism* between them is a bijection $\mathsf{iso} : V \to V'$ such that $\mathsf{iso}(init) = init'$ and, for all states $v, w \in V$ and labels $\ell \in L \cup L'$, $(v, \ell, w) \in A$ *iff* $(\mathsf{iso}(v), \ell, \mathsf{iso}(w)) \in A'$. In the consistency results presented in section 6, we will construct binary relations defining isomorphisms between transition systems of boxes and the corresponding box expressions.

4.3. Operator boxes

An *operator box* Ω is an unmarked, finite, simple, ex-restricted and control-restricted labelled net with only control places (hence it is not $\mathbb{B}$-restricted) and such that every transition v is labelled by an interface function. For every operator box Ω, we will assume that its transitions $v_1, \ldots, v_n$ are implicitly ordered, and then each n–tuple of nets (or expressions later on) $\boldsymbol{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, will be referred to as an Ω-*tuple* (or, simply, a *tuple*); we will also use Σ_{v_i} to denote Σ_i, for $i \leq n$. The notation $\boldsymbol{\Sigma}$ will be used in the net substitution operation, denoted by $\Omega(\boldsymbol{\Sigma})$, and defined in the next section.

In the main body of this paper, we will consider four groups of operator boxes for ABC, which are either binary ($n = 2$, typically used in infix notation) or unary ($n = 1$, typically used in postfix notation), as described below and shown in figure 8.

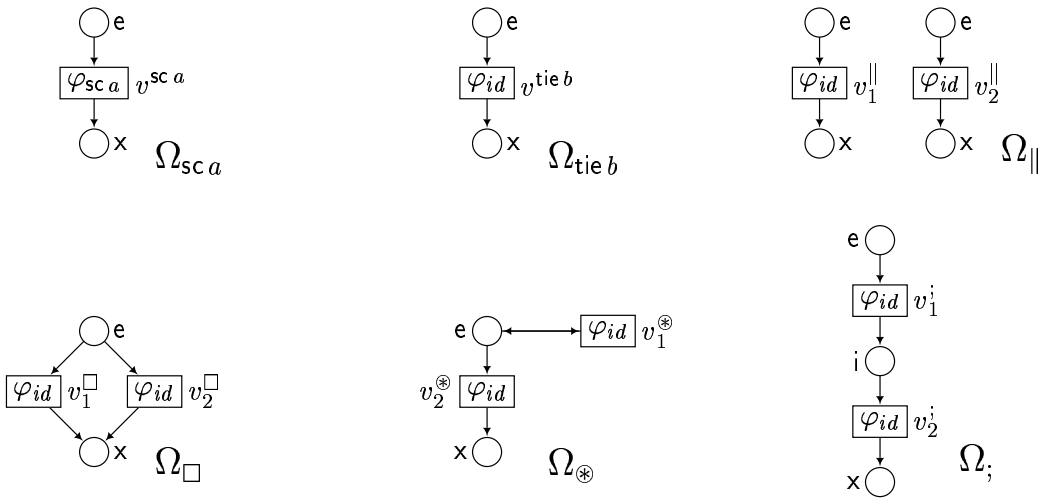


Figure 8. Operator boxes used in ABC, where $a \in \mathbb{A}$ and $b \in \mathbb{B}$.

Choice $\Omega_{\square}$, iteration $\Omega_{\circledast}$ and sequence $\Omega_{;}$. The first three operator boxes specify different ways to sequentially compose behaviours. They are all binary, with the domains of application $dom_{\Omega_{\square}} = dom_{\Omega_{\circledast}} = dom_{\Omega_{;}} \stackrel{\text{df}}{=} (\mathsf{abox}^{stc})^2 \cup (\mathsf{abox}^{dyn} \times \mathsf{abox}^{stc}) \cup (\mathsf{abox}^{stc} \times \mathsf{abox}^{dyn})$, so that at most one of their two operands is ever dynamic. We will denote: $\Sigma_1 \,\square\, \Sigma_2 \stackrel{\text{df}}{=} \Omega_{\square}(\Sigma_1, \Sigma_2)$, $\Sigma_1 \circledast \Sigma_2 \stackrel{\text{df}}{=} \Omega_{\circledast}(\Sigma_1, \Sigma_2)$ and $\Sigma_1 \,;\, \Sigma_2 \stackrel{\text{df}}{=} \Omega_{;}(\Sigma_1, \Sigma_2)$.

Parallel composition $\Omega_{\|}$. This is also a binary operator box, but with the domain of application $dom_{\Omega_{\|}} \stackrel{\text{df}}{=} (\mathsf{abox}^{stc})^2 \cup (\mathsf{abox}^{dyn})^2$, so that in the latter case its two operands may evolve concurrently. We will denote $\Sigma_1 \| \Sigma_2 \stackrel{\text{df}}{=} \Omega_{\|}(\Sigma_1, \Sigma_2)$.

Scoping $\Omega_{\mathsf{sc}\,a}$. Parameterised by a communication action $a \in \mathbb{A}$, this is a unary operator with the domain of application $dom_{\Omega_{\mathsf{sc}\,a}} \stackrel{\text{df}}{=} \mathsf{abox}^{stc} \cup \mathsf{abox}^{dyn}$. The change of the synchronous communication interface it provides is captured by $\varphi_{\mathsf{sc}\,a}$, already defined; essentially, this forces the synchronisation of the pairs of actions a and $\widehat{a}$. We will denote $\Sigma\,\mathsf{sc}\,a \stackrel{\text{df}}{=} \Omega_{\mathsf{sc}\,a}(\Sigma)$.

Buffer restriction $\Omega_{\mathsf{tie}\,b}$. Parameterised by a buffer $b \in \mathbb{B}$, this unary operator has the domain of application $dom_{\Omega_{\mathsf{tie}\,b}} \stackrel{\text{df}}{=} \mathsf{abox}^{stc} \cup \mathsf{abox}^{dyn}$. Buffer restriction will hide the b-labelled open buffer place of the box it is applied to. We will denote $\Sigma\,\mathsf{tie}\,b \stackrel{\text{df}}{=} \Omega_{\mathsf{tie}\,b}(\Sigma)$.

4.4. Net substitution

Throughout the rest of the paper, the identities of transitions in asynchronous boxes will play a key technical role, especially when defining the SOS semantics of process expressions. For such a model, transition identities will come in the form of finite labelled trees retracing the operators used to construct a box. This will allow us to prove a crucial link between the transition-based semantics of process

expressions and nets, formulated as theorem 6.9, which is then used to derive a fundamental result on their label-based semantics formulated as theorem 6.11.

We assume that there is a set η of *basic* transition identities and a corresponding set of basic labelled trees with a single node labelled with an element of η. All the transitions identities in figures 7 and 8 are assumed to be of that kind. To express more complex (unordered) finite trees, or sets of trees, used as transition identities in boxes obtained through net substitution, we will use the following linear notations:

- $v \lhd \mathsf{T}$, where $v \in \eta$ is a basic transition identity and T is a finite multiset of finite labelled trees, denotes a tree where the trees of the multiset T are appended to a root labelled with v.
- $v \lhd \mathtt{t}$ denotes the tree $v \lhd \{\mathtt{t}\}$, and $v \blacktriangleleft \mathsf{T}$ denotes the set of trees $\{v \lhd \mathtt{t} \mid \mathtt{t} \in \mathsf{T}\}$.

A similar, though slightly more complex, naming discipline could be used for the places of the constructed nets, following the scheme used in [5]. However, since place trees were essentially needed for the definition of recursion, which is not considered in this paper, we will not use them here. Instead, we assume that place identities may be changed at will to avoid clashes. In particular, when applying net substitution, we will assume that the place sets of the operands are pairwise disjoint; if this is not the case, we rename them in a consistent way (see also section 4.5). Under such an assumption, in the following, we shall construct new places by grouping existing ones, *e.g.*, if s_1 and s_2 are places of some operand boxes, then (s_1, s_2) may be the identity of a newly constructed place.

The transition naming scheme given above as well as the net substitution meta-operator are illustrated in figure 9, where explicit transition identities are shown for various stages of the construction, from the basic net and transition identities shown in figures 7 and 8.

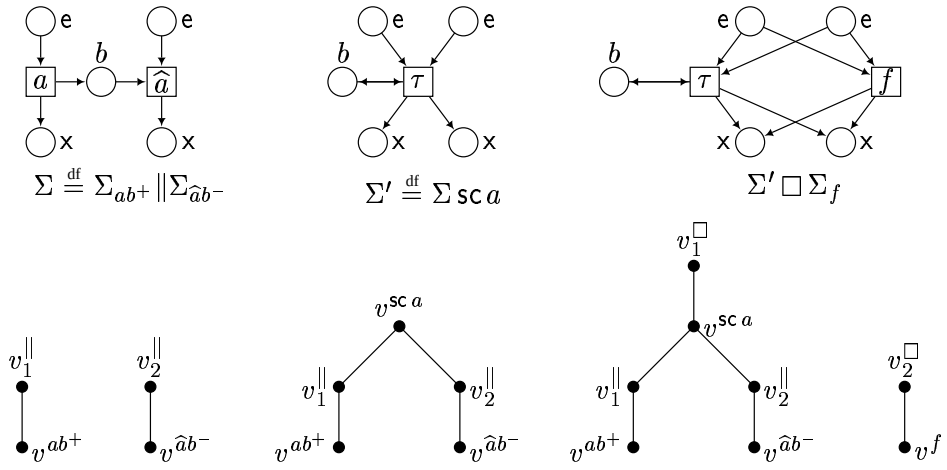


Figure 9. The trees in the bottom row give the identities of the transitions (from left to right) of the boxes shown in the upper row. In the linear notation, the fourth tree is $v_1^{\square} \lhd v^{\mathsf{sc}\,a} \lhd \{v_1^{\|} \lhd v^{ab^+}, v_2^{\|} \lhd v^{\widehat{a}b^-}\}$.

Binary operators. Let $\Omega = \Omega_{\mathsf{bin}}$ (for $\mathsf{bin} \in \{\square, \circledast, ;, \|\}$) be a binary ABC operator box, $v_i = v_i^{\mathsf{bin}}$ (for $i \in \{1, 2\}$) be its transitions, and $\boldsymbol{\Sigma} = (\Sigma_1, \Sigma_2) = (\Sigma_{v_1}, \Sigma_{v_2})$ be a pair of boxes. The result of

a simultaneous substitution of the transitions v_1 and v_2 in Ω by the boxes Σ_1 and Σ_2 is a labelled net $\Omega(\boldsymbol{\Sigma}) = \Phi$ whose components are defined as follows.

The set of transitions of Φ is the set of all trees $v_i \lhd t$ with $t \in T_{\Sigma_i}$ and $i \in \{1, 2\}$. The label of each $v_i \lhd t$ is that of t. Each i-labelled or b-labelled place $p \in S_{\Sigma_i}$ belongs to S_Φ. Its label and marking are unchanged and for every transition $w \lhd t$, the weight function is given by:

$$W_\Phi(p, w \lhd t) \stackrel{\text{df}}{=} \begin{cases} W_{\Sigma_i}(p,t) & \text{if } w = v_i \\ 0 & \text{otherwise ,} \end{cases}$$

and similarly for $W_\Phi(w \lhd t, p)$.

For every place $s \in S_\Omega$ with ${}^\bullet s = \{u_1, \ldots, u_k\}$ and $s^\bullet = \{w_1, \ldots, w_m\}$, we add to S_Φ all places of the form $p \stackrel{\text{df}}{=} (x_1, \ldots, x_k, e_1, \ldots, e_m)$, where each x_i (if any) is an exit place of Σ_{u_i}, and each e_j (if any) is an entry place of Σ_{w_j}. The label of p is that of s, its marking is the sum of the markings of $x_1, \ldots, x_k, e_1, \ldots, e_m$, and for every transition $w \lhd t$, the weight function is given by:

$$W_\Phi(p, w \lhd t) \stackrel{\text{df}}{=} \begin{cases} W_{\Sigma_w}(x_i,t) + W_{\Sigma_w}(e_j,t) & \text{if } w \in {}^\bullet s \cap s^\bullet \text{ and } w = u_i = w_j \\ W_{\Sigma_w}(x_i,t) & \text{if } w \in {}^\bullet s \setminus s^\bullet \text{ and } w = u_i \\ W_{\Sigma_w}(e_j,t) & \text{if } w \in s^\bullet \setminus {}^\bullet s \text{ and } w = w_j \\ 0 & \text{otherwise ,} \end{cases} \tag{1}$$

and similarly for $W_\Phi(w \lhd t, p)$. It may be noticed that for the algebra considered here it is always the case that $k, m \in \{0, 1, 2\}$ and $k + m \in \{1, 2, 3\}$, but that the definition has been formulated in such a way that it works for any values of k and m.

For every $b \in \mathbb{B}$, there is a unique b-labelled place $p^b \stackrel{\text{df}}{=} (p^b_{v_1}, p^b_{v_2}) \in S_\Phi$, where $p^b_{v_1}$ and $p^b_{v_2}$ are the unique b-labelled places of respectively Σ_1 and Σ_2. The marking of p^b is the sum of the markings of $p^b_{v_1}$ and $p^b_{v_2}$, and for each transition $w \lhd t$, the weight function is given by:

$$W_\Phi(p^b, w \lhd t) \stackrel{\text{df}}{=} W_{\Sigma_w}(p^b_w, t) \ ,$$

and similarly for $W_\Phi(w \lhd t, p^b)$.

Scoping. Applying the scoping operator $\Omega_{\mathsf{sc}\, a}$ to a box Σ results in a labelled net which is like Σ with the only difference that the set of transitions comprises all trees $t \stackrel{\text{df}}{=} v^{\mathsf{sc}\, a} \lhd \{t_1, t_2\}$ with $t_1, t_2 \in T_\Sigma$ such that $\{\lambda_\Sigma(t_1), \lambda_\Sigma(t_2)\} = \{a, \widehat{a}\} \in dom(\varphi_{\mathsf{sc}\, a})$, as well as all trees $t' \stackrel{\text{df}}{=} v^{\mathsf{sc}\, a} \lhd t_3$ with $t_3 \in T_\Sigma$ such that $\lambda_\Sigma(t_3) \in dom(\varphi_{\mathsf{sc}\, a}) \setminus \{a, \widehat{a}\}$, *i.e.*, $\lambda_\Sigma(t_3) \notin \{a, \widehat{a}\}$. The label of t is $\tau = \varphi_{\mathsf{sc}\, a}(\{a, \widehat{a}\})$, and that of t' is $\lambda(t_3) = \varphi_{\mathsf{sc}\, a}(\lambda(t_3))$. The weight function, for every place p, is given by:

$$W_{\Omega_{\mathsf{sc}\, a}(\Sigma)}(p,t) \stackrel{\text{df}}{=} W_\Sigma(p,t_1) + W_\Sigma(p,t_2) \qquad \text{and} \qquad W_{\Omega_{\mathsf{sc}\, a}(\Sigma)}(p,t') \stackrel{\text{df}}{=} W_\Sigma(p,t_3) \ ,$$

and similarly for $W_{\Omega_{\mathsf{sc}\, a}(\Sigma)}(t,p)$ and $W_{\Omega_{\mathsf{sc}\, a}(\Sigma)}(t',p)$.

Buffer restriction. Applying the buffer restriction operator $\Omega_{\mathsf{tie}\, b}$ to a box Σ results in a labelled net like Σ with the only difference that the identity of each transition $t \in T_\Sigma$ is changed to $v^{\mathsf{tie}\, b} \lhd t$, the label of the only b-labelled place is changed to b, and a new unmarked isolated b-labelled place is added to $S_{\Omega_{\mathsf{tie}\, b}(\Sigma)}$.

4.5. Places of compositionally defined nets

As already mentioned, in ABC we tried to avoid using an explicit scheme for generating new names of places in compositionally defined nets, which was primarily introduced in PNA to handle recursion. However, since we will often aim at establishing that some composite nets are the same (and not only isomorphic) as other composite nets, we need to assume that some deterministic scheme for generating new places has been implemented. For the binary ABC operators, a suitable scheme has been defined in section 4.3, based on 'consistent renaming' of places aimed at avoiding clashes when composing nets with overlapping place identities. Henceforth we shall assume that this renaming is itself well defined and deterministic, and that it only relies on the identities of the places of the two component nets. As a consequence, for any ABC binary operator bin, if Σ has the same places with the same labels (but possibly different markings) as Σ', and the same holds for Θ and Θ', then the places and their labels (but not necessarily the markings) will be the same for the composite nets $\Sigma\,\mathsf{bin}\,\Theta$ and $\Sigma'\,\mathsf{bin}\,\Theta'$. Below we clarify the situation for the remaining part of our net algebra.

To start with, no operator which only changes markings of control and/or buffer places affects the identities and labels of the existing places (though it may change their markings). Even less intrusive is the scoping operator which only affects transitions, but leaves place, their markings and labels untouched. The last operator to consider is $\mathsf{tie}\,b$ which adds one new buffer place and modifies the label (but neither the identity nor marking) of one existing buffer place. We shall henceforth assume that the identity of the new buffer place is determined by the existing places of the net it is applied to, and so if two nets differ only by their markings, then the identity of newly added buffer places is the same.

We therefore have the following immediate and useful properties of composite nets.

Proposition 4.4. Let Σ and Θ be boxes such that $\lceil\Sigma\rceil = \lceil\Theta\rceil$, $a \in \mathbb{A}$, $b \in \mathbb{B}$ and $B \in \mathsf{mult}(\mathbb{B})$. Then

$$\Sigma\,\mathsf{sc}\,a = \Theta\,\mathsf{sc}\,a \;\; \textit{iff} \;\; \Sigma\,\mathsf{tie}\,b = \Theta\,\mathsf{tie}\,b \;\; \textit{iff} \;\; \Sigma.B = \Theta.B \;\; \textit{iff} \;\; \Sigma = \Theta. \qquad \square$$

Proposition 4.5. Let Σ be a box, $a \in \mathbb{A}$, $b \in \mathbb{B}$ and $B \in \mathsf{mult}(\mathbb{B})$.

1. $\lfloor\Sigma\,\mathsf{tie}\,b\rfloor = \lfloor\Sigma\rfloor\,\mathsf{tie}\,b$, $\lfloor\Sigma\,\mathsf{sc}\,a\rfloor = \lfloor\Sigma\rfloor\,\mathsf{sc}\,a$ and $\lfloor\Sigma.B\rfloor = \lfloor\Sigma\rfloor.B$.
2. $\lceil\Sigma\,\mathsf{tie}\,b\rceil = \lceil\Sigma\rceil\,\mathsf{tie}\,b$ and $\lceil\Sigma\,\mathsf{sc}\,a\rceil = \lceil\Sigma\rceil\,\mathsf{sc}\,a$.
3. $\langle\Sigma\,\mathsf{tie}\,b\rangle = \langle\Sigma\rangle\,\mathsf{tie}\,b$ and $\langle\Sigma\,\mathsf{sc}\,a\rangle = \langle\Sigma\rangle\,\mathsf{sc}\,a$.
4. $M^{ctr}_{\Sigma\,\mathsf{tie}\,b} = M^{ctr}_{\Sigma.B} = M^{ctr}_{\Sigma}$ and $M_{\Sigma\,\mathsf{sc}\,a} = M_{\Sigma}$. $\square$

4.6. Consistency in the box domain

For a composite net, the property of having an empty control marking is directly linked to the same property of the composed nets.

Proposition 4.6. Let Ω be any binary ABC operator box, and Σ_1, Σ_2 and Σ be any boxes.

1. $M^{ctr}_{\Omega(\Sigma_1,\Sigma_2)} = \varnothing \;\; \textit{iff} \;\; M^{ctr}_{\Sigma_1} = \varnothing = M^{ctr}_{\Sigma_2}$.
2. $M^{ctr}_{\Sigma'} = \varnothing \;\; \textit{iff} \;\; M^{ctr}_{\Sigma} = \varnothing$, for each Σ' of the form $\Sigma\,\mathsf{sc}\,a$ or $\Sigma\,\mathsf{tie}\,b$ or $\Sigma.B$.

Proof:
Follows from proposition A.1 in the appendix. □

Moreover, the operation of net substitution always returns a syntactically valid object provided that it is applied to operands belonging to the correct domain.

Theorem 4.7. Let Ω be any binary ABC operator box and $\boldsymbol{\Sigma} \in dom_\Omega$. Then $\Omega(\boldsymbol{\Sigma})$ is a box with a clean and ac-free marking. Moreover, if all the dynamic boxes (if any) in $\boldsymbol{\Sigma}$ have quasi-safe markings, then the marking of $\Omega(\boldsymbol{\Sigma})$ is also quasi-safe.

Proof:
Follows from theorem A.4 in the appendix and the definitions. □

Theorem 4.8. Let Σ be a static or dynamic box, $a \in \mathbb{A}$, $b \in \mathbb{B}$ and $B \in \mathsf{mult}(\mathbb{B})$. Then $\Sigma\,\mathsf{sc}\,a$, $\Sigma\,\mathsf{tie}\,b$ and $\Sigma.B$ are boxes with clean and ac-free markings. Moreover, if Σ has a quasi-safe marking, then the markings of these boxes are also quasi-safe.

Proof:
Follows from theorem A.4 in the appendix and the definitions. □

Proposition 4.9. Let Ω be an ABC operator box and $\boldsymbol{\Sigma} \in dom_\Omega$.
Then $\langle\boldsymbol{\Sigma}\rangle \in dom_\Omega$ and $\Omega(\langle\boldsymbol{\Sigma}\rangle) = \langle\Omega(\boldsymbol{\Sigma})\rangle$.

Proof:
Follows directly from proposition 4.1(2) and the definition of net refinement. □

We finally observe that if one makes no use of buffer stuffing, buffer restriction nor the basic nets Σ_{ab^+}, Σ_{ab^-} and $\Sigma_{ab^\pm}$, then the net operations described above are similar to those defined in the standard box algebra (see [3, 4, 5]), except for the additional buffer places, which are all isolated and unmarked.

5. Behaviour and structure of composite boxes

In this section we investigate how various characteristics of composite nets depend on similar characteristics of the nets being composed.

5.1. Structural equivalence

We first want to capture all those cases where applying an operator box Ω to two different Ω-tuples of boxes yields the same result. We start by defining five auxiliary relations $\equiv^i_\Omega$ which identify pairs of Ω-tuples containing boxes with particular combinations of initial and/or final markings, in such a way that the result of applying Ω leads to the same marking for both tuples.

For an operator box Ω and two Ω-tuples of boxes, $\boldsymbol{\Sigma}$ and $\boldsymbol{\Theta}$, we have:[6]

[6] In view that we do not consider operator boxes of arity higher than two in the main body of the paper, some of the auxiliary relations below could be simplified. However, we will need this slightly more general formulation in the appendix.

- $\boldsymbol{\Sigma} \equiv^0_\Omega \boldsymbol{\Theta}$ if $\boldsymbol{\Sigma}$ and $\boldsymbol{\Theta}$ are the same except that $\Sigma_v = \overline{\Psi}$ and $\Theta_v = \underline{\Psi}$, for some box Ψ and a transition $v \in T_\Omega$ satisfying ${}^\bullet v = v^\bullet$. This relation is needed for the iteration operator: it does not matter whether we take the first operand in an initial marking, or in a final marking.

- $\boldsymbol{\Sigma} \equiv^1_\Omega \boldsymbol{\Theta}$ if $\boldsymbol{\Sigma}$ and $\boldsymbol{\Theta}$ are the same except that $\Sigma_v = \overline{\Theta}_v$ and $\overline{\Sigma}_w = \Theta_w$ (or $\overline{\Sigma}_v = \Theta_v$ and $\Sigma_w = \overline{\Theta}_w$), for some distinct transitions $v, w \in T_\Omega$ satisfying ${}^\bullet v = {}^\bullet w$. This relation is needed for the choice and iteration operators: it does not matter whether we take the first operand in an initial marking, or the second operand in an initial marking.

- $\boldsymbol{\Sigma} \equiv^2_\Omega \boldsymbol{\Theta}$ if $\boldsymbol{\Sigma}$ and $\boldsymbol{\Theta}$ are the same except that $\Sigma_v = \underline{\Theta}_v$ and $\underline{\Sigma}_w = \Theta_w$ (or $\underline{\Sigma}_v = \Theta_v$ and $\Sigma_w = \underline{\Theta}_w$), for some distinct transitions $v, w \in T_\Omega$ satisfying $v^\bullet = w^\bullet$. This relation, involving final markings, is similar to the previous one but intended only for the choice operator.

- $\boldsymbol{\Sigma} \equiv^3_\Omega \boldsymbol{\Theta}$ if $\boldsymbol{\Sigma}$ and $\boldsymbol{\Theta}$ are the same except that $\Sigma_v = \underline{\Theta}_v$ and $\overline{\Sigma}_w = \Theta_w$ (or $\underline{\Sigma}_v = \Theta_v$ and $\Sigma_w = \overline{\Theta}_w$), for some distinct transitions $v, w \in T_\Omega$ satisfying $v^\bullet = {}^\bullet w$. This relation is needed for the sequence and iteration operators after the first component has reached a final state; the relation states that this is equivalent to the second operand being in an initial state.

- $\boldsymbol{\Sigma} \equiv^4_\Omega \boldsymbol{\Theta}$ if $\boldsymbol{\Sigma} = \boldsymbol{\Theta}$. This relation is simply the identity on Ω-tuples.

- $\boldsymbol{\Sigma} \equiv^5_\Omega \boldsymbol{\Theta}$ if, for each $v \in T_\Omega$, there are is a box Ψ_v and two multisets over $\mathbb{B}$, B_v and B'_v, such that $\Sigma_v = \Psi_v.B_v$, $\Theta_v = \Psi_v.B'_v$ and $\sum_{v \in T_\Omega} B_v = \sum_{v \in T_\Omega} B'_v$. In other words, $\boldsymbol{\Sigma}$ and $\boldsymbol{\Theta}$ are the same except perhaps the distribution of tokens in the open buffer places corresponding to the same b but coming from different components.[7] This relation is intended for the binary operators as they always merge the corresponding open buffers, adding their markings.

By combining these auxiliary relations, we obtain the relation capturing all situations where applying an operator box Ω leads to the same result:[8]

$$\equiv_\Omega \stackrel{\text{df}}{=} \equiv^5_\Omega \circ \bigcup_{i=0}^{4} \equiv^i_\Omega .$$

One can easily see that, for the operator boxes of ABC, we have:

$$\begin{array}{lcllcl}
\equiv_{\Omega_{||}} & = & \equiv^5_{\Omega_{||}} & \equiv_{\Omega_\Box} & = & \equiv^5_{\Omega_\Box} \circ (\equiv^1_{\Omega_\Box} \cup \equiv^2_{\Omega_\Box} \cup \equiv^4_{\Omega_\Box}) \\
\equiv_{\Omega_{\mathsf{sc}\,a}} & = & id_{\mathsf{abox}} & \equiv_{\Omega_\circledast} & = & \equiv^5_{\Omega_\circledast} \circ (\equiv^0_{\Omega_\circledast} \cup \equiv^1_{\Omega_\circledast} \cup \equiv^3_{\Omega_\circledast} \cup \equiv^4_{\Omega_\circledast}) \\
\equiv_{\Omega_{\mathsf{tie}\,b}} & = & id_{\mathsf{abox}} & \equiv_{\Omega_;} & = & \equiv^5_{\Omega_;} \circ (\equiv^3_{\Omega_;} \cup \equiv^4_{\Omega_;})
\end{array} \qquad (2)$$

where id_{abox} is the identity relation on boxes. For example, $\equiv_{\Omega_{||}}$ relates all those pairs of boxes which may differ only by the distribution of tokens in the open buffer places; whereas $\equiv_{\Omega_;}$ is less restrictive by also allowing one pair to have its first component in the exit marking and the other pair to have its second component in the entry marking (intuitively, the state reached after the termination of the first part of a sequential composition is the same as that before the beginning of the second part).

[7] This distribution only concerns *open* buffer places since buffer stuffi ng never changes the marking of *closed* buffer places.

[8] The composition of two binary relations R, R' on a set X is defi ned as $R \circ R' = \{(x, y) \mid \exists z \in X : (x, z) \in R \wedge (z, y) \in R'\}$.

Note that the relation $\equiv_\Omega$ is reflexive and symmetric, but in general it is not transitive; see, *e.g.*, the following pairs of boxes:

$$(\overline{\overline{\Sigma_1}}, \Sigma_2) \equiv_{\Omega_\Box} (\overline{\Sigma_1}, \overline{\Sigma_2}) \equiv_{\Omega_\Box} (\Sigma_1, \overline{\overline{\Sigma_2}}) \not\equiv_{\Omega_\Box} (\overline{\overline{\Sigma_1}}, \Sigma_2) \ .$$

But this is only due to the fact that we defined it in a too large domain (*i.e.*, the set of all pairs of boxes), and restricting $\equiv_\Omega$ to the application domain of Ω makes it transitive.

Proposition 5.1. Let Ω be an ABC operator box, $\boldsymbol{\Sigma} \in dom_\Omega$, and $\boldsymbol{\Theta}$ be an Ω-tuple of boxes.

1. If $\boldsymbol{\Sigma} \equiv_\Omega \boldsymbol{\Theta}$, then $\boldsymbol{\Theta} \in dom_\Omega$ and $\lceil \boldsymbol{\Sigma} \rceil = \lceil \boldsymbol{\Theta} \rceil$.
   Moreover, if all the boxes in $\boldsymbol{\Sigma}$ are quasi-safe then so are all the boxes in $\boldsymbol{\Theta}$.
2. $\equiv_\Omega$ is an equivalence relation on dom_Ω.
3. If $\lceil \boldsymbol{\Sigma} \rceil = \lceil \boldsymbol{\Theta} \rceil$, then $\Omega(\boldsymbol{\Sigma}) = \Omega(\boldsymbol{\Theta})$ *iff* $\boldsymbol{\Sigma} \equiv_\Omega \boldsymbol{\Theta}$.

Proof:
Follows from proposition A.6 in the appendix. □

It is interesting to note that the precondition $\lceil \boldsymbol{\Sigma} \rceil = \lceil \boldsymbol{\Theta} \rceil$ in the last part of the above proposition cannot be dropped; for example, we have $\Sigma_{ab^+} \not\equiv_{\Omega_{\mathsf{sc}\, a}} \Sigma_{ab^-}$ and $\Sigma_{ab^+}\, \mathsf{sc}\, a = \Sigma_{ab^-}\, \mathsf{sc}\, a$ (no transition is left in the nets by the scoping operation). Fortunately, whenever we need to apply proposition 5.1(3), we will deal with nets which may differ only by their markings, so this result will be sufficient for our needs.

A number of static properties of composite nets can now be stated.

Proposition 5.2. Let Ω be a binary ABC operator, Σ and Θ be static boxes, Φ be a static or dynamic box, $a \in \mathbb{A}$, $b \in \mathbb{B}$ and $B, B' \in \mathsf{mult}(\mathbb{B})$.

1. $\overline{\Sigma \,\Box\, \Theta} = \overline{\Sigma} \,\Box\, \Theta = \Sigma \,\Box\, \overline{\Theta}$ and $\underline{\Sigma \,\Box\, \Theta} = \underline{\Sigma} \,\Box\, \Theta = \Sigma \,\Box\, \underline{\Theta}$.
2. $\overline{\Sigma \circledast \Theta} = \overline{\Sigma} \circledast \Theta = \underline{\Sigma} \circledast \Theta = \Sigma \circledast \overline{\Theta}$ and $\underline{\Sigma \circledast \Theta} = \Sigma \circledast \underline{\Theta}$.
3. $\overline{\Sigma\,;\,\Theta} = \overline{\Sigma}\,;\,\Theta$, $\underline{\Sigma}\,;\,\Theta = \Sigma\,;\,\overline{\Theta}$ and $\Sigma\,;\,\underline{\Theta} = \underline{\Sigma\,;\,\Theta}$.
4. $\overline{\Sigma \| \Theta} = \overline{\Sigma} \| \overline{\Theta}$ and $\underline{\Sigma} \| \underline{\Theta} = \underline{\Sigma \| \Theta}$.
5. $\overline{\Sigma\, \mathsf{sc}\, a} = \overline{\Sigma}\, \mathsf{sc}\, a$ and $\underline{\Sigma\, \mathsf{sc}\, a} = \underline{\Sigma}\, \mathsf{sc}\, a$.
6. $\overline{\Sigma\, \mathsf{tie}\, b} = \overline{\Sigma}\, \mathsf{tie}\, b$ and $\underline{\Sigma\, \mathsf{tie}\, b} = \underline{\Sigma}\, \mathsf{tie}\, b$.
7. $(\Phi\, \mathsf{sc}\, a).B = (\Phi.B)\, \mathsf{sc}\, a$.
8. $(\Phi\, \mathsf{tie}\, b).B = (\Phi.B)\, \mathsf{tie}\, b$ provided that $b \notin B$.
9. If $\boldsymbol{\Sigma} = (\Sigma_1, \Sigma_2) \in dom_\Omega$, then $\Omega(\Sigma_1.B, \Sigma_2.B') = \Omega(\boldsymbol{\Sigma}).(B + B')$.

Proof:
Follows from proposition A.5 in the appendix, proposition 5.1(3), and the definitions. □

5.2. Dynamic properties of composite nets

In the results presented below, we capture the *behavioural compositionality* of our model, *i.e.*, the way the behaviours of composite nets (in terms of enabled steps) are related to the behaviours of their constituting nets. Basically, we want to establish what steps are enabled by $\Omega(\boldsymbol{\Sigma})$, knowing the steps enabled by the boxes in $\boldsymbol{\Sigma}$.

Proposition 5.3. Let $\Omega_{\mathsf{bin}} \in \{\Omega_{||}, \Omega_{\Box}, \Omega_{\circledast}, \Omega_{;}\}$ and $\boldsymbol{\Sigma} \in dom_{\Omega_{\mathsf{bin}}}$. Then $\mathsf{enabled}(\Omega_{\mathsf{bin}}(\boldsymbol{\Sigma}))$ comprises exactly all sets of transitions $U = (v_1^{\mathsf{bin}} \blacktriangleleft U_1) \cup (v_2^{\mathsf{bin}} \blacktriangleleft U_2)$ such that there is a pair of boxes $\boldsymbol{\Theta}$ satisfying $\boldsymbol{\Theta} \equiv_{\Omega_{\mathsf{bin}}} \boldsymbol{\Sigma}$ and $U_i \in \mathsf{enabled}(\Theta_i)$, for $i \in \{1,2\}$. Moreover, $\Omega_{\mathsf{bin}}(\boldsymbol{\Sigma})\,[U\rangle\,\Omega_{\mathsf{bin}}(\boldsymbol{\Phi})$, where $\Theta_i\,[U_i\rangle\,\Phi_i$, for $i \in \{1,2\}$.

Proof:
For $\Omega_{||}$ the result follows from the definitions, and $\equiv_{\Omega_{||}}$ is needed in order to rearrange tokens on the open buffer places in Σ_1 and Σ_2, to ensure that the steps U_1 and U_2 are enabled separately. For the other Ω's, the result follows from proposition A.7 in the appendix. □

Note that, for $\Omega_{\mathsf{bin}} \neq \Omega_{||}$, at least one of the steps U_i, $i \in \{1,2\}$, is necessarily empty because the corresponding box is static.

Proposition 5.4. Let Σ be a static or dynamic box, $a \in \mathbb{A}$, $b \in \mathbb{B}$, $B \in \mathsf{mult}(\{b\})$ and $B' \in \mathsf{mult}(\mathbb{B})$.

1. $\mathsf{enabled}(\Sigma\,\mathsf{sc}\,a)$ comprises exactly all sets of transitions

$$U = (v^{\mathsf{sc}\,a} \blacktriangleleft Z) \cup \{v^{\mathsf{sc}\,a} \triangleleft \{v_1, w_1\}, \ldots, v^{\mathsf{sc}\,a} \triangleleft \{v_k, w_k\}\}$$

   such that $Z \cup V \cup W \in \mathsf{enabled}(\Sigma)$, where the steps $V \stackrel{\text{df}}{=} \{v_1, \ldots, v_k\}$, $W \stackrel{\text{df}}{=} \{w_1, \ldots, w_k\}$ and Z satisfy $a, \widehat{a} \notin \lambda_\Sigma(Z)$, and for all $i \in \{1, \ldots, k\}$, $\lambda_\Sigma(v_i) = \{a\}$ and $\lambda_\Sigma(w_i) = \{\widehat{a}\}$.
   Moreover, $\Sigma\,\mathsf{sc}\,a\,[U\rangle\,\Phi\,\mathsf{sc}\,a$, where $\Sigma\,[V \cup W \cup Z\rangle\,\Phi$.

2. $\mathsf{enabled}(\Sigma\,\mathsf{tie}\,b)$ comprises exactly all sets of transitions $U = v^{\mathsf{tie}\,b} \blacktriangleleft V$ such that $V \in \mathsf{enabled}(\Sigma)$.
   Moreover, $\Sigma\,\mathsf{tie}\,b\,[U\rangle\,\Phi\,\mathsf{tie}\,b$, where $\Sigma\,[V\rangle\,\Phi$.

3. $\mathsf{enabled}((\Sigma\,\mathsf{tie}\,b).B)$ comprises exactly all sets of transitions $U \in \mathsf{enabled}(\Sigma\,\mathsf{tie}\,b)$.
   Moreover, $(\Sigma\,\mathsf{tie}\,b).B\,[U\rangle\,\Phi.B$, where $\Sigma\,\mathsf{tie}\,b\,[U\rangle\,\Phi$.

4. $\mathsf{enabled}(\Sigma)$ is a subset of $\mathsf{enabled}(\Sigma.B')$.
   Moreover, if $\Sigma\,[U\rangle\,\Phi$ then $\Sigma.B'\,[U\rangle\,\Phi.B'$.

Proof:
Follows directly from the definitions. □

That the last property does not hold for the equality of sets may be illustrated by a simple counter-example: the dynamic box $\overline{\Sigma_{ab^-}}$ only allows the empty step, while $\overline{\Sigma_{ab^-}}.b\,[\{v^{ab^-}\}\rangle\,\underline{\Sigma_{ab^-}}$.

The behaviours of the basic asynchronous boxes of ABC are captured below.

Proposition 5.5. Let $B \in \mathsf{mult}(\mathbb{B})$ and $\Sigma = \overline{\Sigma_\alpha}.B$, where Σ_α is one of the basic boxes in figure 7.

1. For $\alpha \in \{a, ab^+\}$, the non-empty steps of Σ are respectively

$$\Sigma\,[\{v^a\}\rangle\, \underline{\Sigma_a}.B \quad \text{and} \quad \Sigma\,[\{v^{ab^+}\}\rangle\, \underline{\Sigma_{ab^+}}.b.B\ .$$

2. For $\alpha \in \{ab^-, ab^\pm\}$, if $b \notin B$ then Σ has no non-empty step; otherwise the non-empty steps are respectively

$$\Sigma\,[\{v^{ab^-}\}\rangle\, \underline{\Sigma_{ab^-}}.(B - \{b\}) \quad \text{and} \quad \Sigma\,[\{v^{ab^\pm}\}\rangle\, \underline{\Sigma_{ab^\pm}}.B\ .$$

Moreover, if $\Sigma = \Sigma_\alpha.B$ or $\Sigma = \underline{\Sigma_\alpha}.B$, then Σ has no non-empty step.

Proof:
Follows directly from the definitions. □

Various important consequences may be derived from the results presented above; in particular that the way static and dynamic boxes are composed in ABC guarantees that the result is a static or dynamic box when the domain of application of the operators is respected.

Theorem 5.6. Let Ω be an operator box of ABC and $\boldsymbol{\Sigma} \in dom_\Omega$. Then $\Omega(\boldsymbol{\Sigma})$ is a static or dynamic box. Moreover,

- if all the boxes in $\boldsymbol{\Sigma}$ are static then $\Omega(\boldsymbol{\Sigma})$ is also static;
- if all the boxes in $\boldsymbol{\Sigma}$ are quasi-safe then $\Omega(\boldsymbol{\Sigma})$ is also quasi-safe.

Proof:
Let $\Theta = \langle\Omega(\boldsymbol{\Sigma})\rangle$. Then, by proposition 4.9, $\Theta = \Omega(\langle\boldsymbol{\Sigma}\rangle)$ and the following are satisfied:

(i) Every net derivable from Θ is of the form $\Omega(\boldsymbol{\Theta})$, where $\boldsymbol{\Theta} \in dom_\Omega$. Moreover, if all the boxes in $\boldsymbol{\Sigma}$ are quasi-safe then all the boxes in $\boldsymbol{\Theta}$ are also quasi-safe.

(ii) If no box in $\boldsymbol{\Sigma}$ is dynamic, then every net derivable from $\overline{\Theta}$ or $\underline{\Theta}$ is of the form $\Omega(\boldsymbol{\Theta})$, where $\boldsymbol{\Theta} \in dom_\Omega$. Moreover, if all the boxes in $\boldsymbol{\Sigma}$ are quasi-safe then all the boxes in $\boldsymbol{\Theta}$ are also quasi-safe.

Indeed, (i) follows from propositions 4.2, 5.1(1), 5.3 and 5.4(1,2); and (ii) follows from (i) together with proposition 5.2(1–6).

Then the main result follows from (i,ii), proposition 4.6 and theorems 4.7 and 4.8. □

We end this section providing a simple syntactic definition of a class of compositional nets built using various net operations introduced in this paper. We call them *ABC boxes* (denoted by box_{ABC}), and distinguish: ABC boxes *with* control tokens (denoted by box^{ctr}_{ABC}), and ABC boxes *without* control tokens (denoted by $\mathsf{box}^{noctr}_{ABC}$). They are defined as the smallest sets of boxes satisfying the following (below B is any multiset over $\mathbb{B}$):

- Each basic box in figure 7 belongs to $\mathsf{box}^{noctr}_{ABC}$.
- If $\Sigma \in \mathsf{box}^{noctr}_{ABC}$ then $\overline{\Sigma}, \underline{\Sigma} \in \mathsf{box}^{ctr}_{ABC}$.

- If $\Sigma \in \mathsf{box}_{ABC}^{noctr}$ then $\Sigma.B \in \mathsf{box}_{ABC}^{noctr}$, and if $\Sigma \in \mathsf{box}_{ABC}^{ctr}$ then $\Sigma.B \in \mathsf{box}_{ABC}^{ctr}$.
- If $\Sigma \in \mathsf{box}_{ABC}$ then $\lfloor\Sigma\rfloor, \lceil\Sigma\rceil \in \mathsf{box}_{ABC}^{noctr}$.
- If Ω is an operator box of ABC and $\boldsymbol{\Sigma}$ is an Ω-tuple of ABC boxes then $\Omega(\boldsymbol{\Sigma})$ is an ABC box, provided that the tuple would have belonged to dom_Ω if we replaced abox^{stc} and abox^{dyn} respectively by $\mathsf{box}_{ABC}^{noctr}$ and box_{ABC}^{ctr} in the definition of dom_Ω. Moreover, if all the boxes in $\boldsymbol{\Sigma}$ belong to $\mathsf{box}_{ABC}^{noctr}$ then so does $\Omega(\boldsymbol{\Sigma})$; otherwise $\Omega(\boldsymbol{\Sigma})$ belongs to box_{ABC}^{ctr}.

It is then possible to show the following.

Theorem 5.7. Every ABC box without (with) control tokens is a quasi-safe static (resp. dynamic) box.

Proof:
Let Σ be an ABC box. The proof proceeds by induction on the number k of applications of various operators used in the process of constructing Σ (since, in principle, an ABC box may be constructed in many ways, we may always choose the smallest k with such a property).

In the base case ($k = 0$), Σ is a basic asynchronous box in figure 7, and it is straightforward to see that the result holds. In the inductive step, we consider three cases.

Case 1: $\Sigma = \overline{\Theta}$ or $\Sigma = \underline{\Theta}$. Then $\Theta \in \mathsf{box}_{ABC}^{noctr}$ and so, by the induction hypothesis, Θ is a quasi-safe static box. Thus, by proposition 4.3(7), Σ is a quasi-safe dynamic box.

Case 2: $\Sigma = \Theta.B$ and $\Theta \in \mathsf{box}_{ABC}^{noctr}$ (the case $\Theta \in \mathsf{box}_{ABC}^{ctr}$ is similar). By the induction hypothesis, Θ is a quasi-safe static box. Thus, by proposition 4.3(5), Σ is a quasi-safe static box.

Case 3: $\Sigma = \lfloor\Theta\rfloor$ or $\Sigma = \lceil\Theta\rceil$. Then $\Theta \in \mathsf{box}_{ABC}^{noctr} \cup \mathsf{box}_{ABC}^{ctr}$ and so, by the induction hypothesis, Θ is a quasi-safe static or dynamic box. Thus, by proposition 4.3(8), Σ is a quasi-safe static box.

Case 4: $\Sigma = \Omega(\boldsymbol{\Sigma})$. Then, by the induction hypothesis, $\boldsymbol{\Sigma}$ is an Ω-tuple of quasi-safe static and/or dynamic boxes. Moreover, by the definition of ABC boxes, $\boldsymbol{\Sigma} \in dom_\Omega$ and so, by theorem 5.6, $\Omega(\boldsymbol{\Sigma})$ is a quasi-safe static or dynamic box. Finally, using proposition 4.6, one can see that $\Sigma \in \mathsf{box}_{ABC}^{noctr}$ *iff* Σ is a static box. □

We finally note that any net derivable from an ABC box is also an ABC box. One can show this using propositions 5.3, 5.4(1,2) and 5.5, as well as the fact that if $\boldsymbol{\Sigma}$ is an Ω-tuple of ABC boxes, for some ABC operator box, and $\boldsymbol{\Sigma} \equiv_\Omega \boldsymbol{\Theta}$, then $\boldsymbol{\Theta}$ is also an Ω-tuple of ABC boxes.

6. An algebra of asynchronous box expressions

We consider the algebra of process expressions over the signature:

$$\mathcal{A} \cup \{\, \mathsf{sc}\, a \mid a \in \mathbb{A}\} \cup \{\, \mathsf{tie}\, b \mid b \in \mathbb{B}\} \cup \{\, .b \mid b \in \mathbb{B}\} \cup \{\, \overline{\cdot}\,, \underline{\cdot}\,\} \cup \{\, \|\,, \;;\,, \Box\,, \circledast\,\}\,, \qquad (3)$$

where $\mathcal{A} = \mathbb{A}_\tau \cup \{\, ab^+\,,\, ab^-\,,\, ab^\pm \mid a \in \mathbb{A}_\tau \wedge b \in \mathbb{B}\}$ are the constants; the unary operators $\mathsf{sc}\, a$, $\mathsf{tie}\, b$ and $.b$ are used in the postfix mode; $\overline{\cdot}$ and $\underline{\cdot}$ are two positional unary operators (the position of the argument being given by the dot); and the binary operators $\|$, $;$, $\Box$ and $\circledast$ are used in the infix mode.

There are two classes of process expressions corresponding to the static and dynamic boxes, viz. the *static* and *dynamic* expressions, denoted respectively by aexpr^{stc} and aexpr^{dyn}. Collectively, we will refer to them as the (asynchronous) *box expressions*, aexpr. Their syntax is given by:

$$\begin{array}{lllllllllll} \mathsf{aexpr}^{stc} & E & ::= & \alpha & | & E\,\mathsf{sc}\,a & | & E\,\mathsf{tie}\,b & | & E.b & | & E\|E & | & E \square E \\ & & & E;E & | & E \circledast E & & & & & & & & \\ \mathsf{aexpr}^{dyn} & D & ::= & \overline{E} & | & \underline{E} & | & D\,\mathsf{sc}\,a & | & D\,\mathsf{tie}\,b & | & D.b & | & D\|D \\ & & & D \square E & | & E \square D & | & D;E & | & E;D & | & D \circledast E & | & E \circledast D \end{array} \tag{4}$$

where $\alpha \in \mathcal{A}$, $a \in \mathbb{A}$ and $b \in \mathbb{B}$. Moreover, we will use F to denote any static or dynamic expression.[9]

We also use the notations $\lfloor F \rfloor$ and $\lceil F \rceil$ yielding static expressions, where $\lfloor F \rfloor$ is F with all occurrences of overbars and underbars removed, and $\lceil F \rceil$ is $\lfloor F \rfloor$ with all occurrences of $.b$ removed. Note that we do not need terms of the form $F.B$ since $F.\{b, \ldots, b'\}$ would be equivalent to $F.b \cdots .b'$ (but such terms can be used as a convenient shorthand).

Essentially, an asynchronous box expression encodes the structure of a box net, together with the current marking of the control places (using overbars and underbars) and of the buffer places (using the $.b$'s). Thus, a box expression $\overline{E}$ represents E in its initial state (in terms of nets, this corresponds to the initially marked box of E). Similarly, $\underline{E}$ represents E in its final state. Note that the $.b$ notation is needed for static as well as for dynamic box expressions because the dormant part of a dynamic box expression may still have $.b$'s which are later needed in the active part. For instance, the expression $ab^+.b\,;\overline{fb^-}$ has a static component with a $.b$ in it, and may be transformed into an equivalent $ab^+;\overline{fb^-.b}$.

6.1. Denotational semantics

The denotational semantics of box expressions is defined by means of the evaluation mapping box, from the (syntactical) asynchronous algebra of box expressions into boxes, following the syntax (4). Below, $\alpha \in \mathcal{A}$, $b \in \mathbb{B}$, una stands for any unary operator ($\mathsf{sc}\,a$, $\mathsf{tie}\,b$ or $.b$), and bin for any binary operator ($\|$, $\square$, ; or $\circledast$).

$$\begin{array}{c} \mathsf{box}(\alpha) \stackrel{\text{df}}{=} \Sigma_\alpha \qquad \mathsf{box}(\overline{E}) \stackrel{\text{df}}{=} \overline{\mathsf{box}(E)} \qquad \mathsf{box}(\underline{E}) \stackrel{\text{df}}{=} \underline{\mathsf{box}(E)} \\ \mathsf{box}(F\,\mathsf{una}) \stackrel{\text{df}}{=} \mathsf{box}(F)\,\mathsf{una} \qquad \mathsf{box}(F_1\,\mathsf{bin}\,F_2) \stackrel{\text{df}}{=} \mathsf{box}(F_1)\,\mathsf{bin}\,\mathsf{box}(F_2)\ . \end{array} \tag{5}$$

The semantical mapping always returns a box, and the property of corresponding to a static or dynamic box has been captured by the syntax (4).

Theorem 6.1. Let F be a box expression.

1. $\mathsf{box}(F)$ is a static or dynamic box.
2. $\mathsf{box}(F)$ is a static box *iff* F is a static box expression.

Proof:
Follows from theorem 5.6, by induction on the structure of F. □

[9] Note that, *w.r.t.* the original PBC and PNA algebras, the above syntax uses slightly different symbols to denote scoping ($E\,\mathsf{sc}\,a$ instead of $[a : E]$) and iteration ($E \circledast E'$ instead of $\langle E * E' \rangle$). However, their meanings remain unchanged.

What is more, it follows directly from the definitions that $\mathsf{box}(F)$ is an ABC box and so, by theorem 5.7, it is quasi-safe.

Proposition 6.2. Let F be a box expression. Then $\lfloor F \rfloor$ and $\lceil F \rceil$ are box expressions satisfying the following.

1. $\mathsf{box}(\lfloor F \rfloor) = \lfloor \mathsf{box}(F) \rfloor$.
2. $\mathsf{box}(\lceil F \rceil) = \lceil \mathsf{box}(F) \rceil$.

Proof:
Follows from the syntax rules (4) and translation rules (5), by induction on the structure of F. □

6.2. Structural similarity relation

We define the *structural similarity* relation on box expressions, denoted by $\equiv$, as the least equivalence relation on box expressions such that all the equations in table 1 are satisfied. Using the rules IPAR, IC1L, IS1 and CON2, one can derive $\overline{(a \,\square\, b) \| (d;e)} \equiv (\overline{a} \,\square\, b) \| (\overline{d};e)$, as in figure 10.

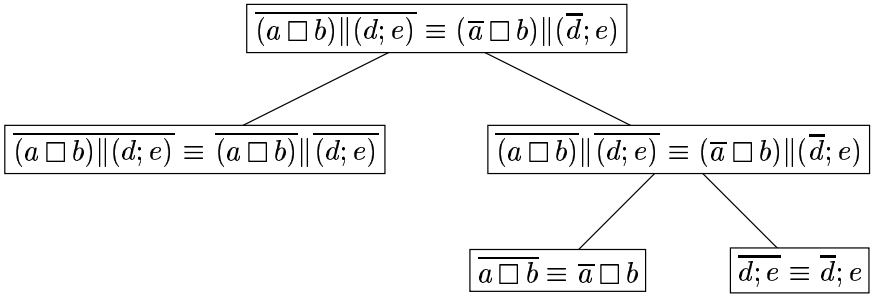


Figure 10. The derivation tree of $\overline{(a \,\square\, b) \| (d;e)} \equiv (\overline{a} \,\square\, b) \| (\overline{d};e)$.

The rules in table 1 either directly follow those of the PNA model or capture the fact that an asynchronous message, produced by the ab^+ expression and represented by $.b$, can freely move within a box expression in order to be received by some action of the form ab^- (see the rules B1, OPL and OPR). However, no $.b$ may ever cross the boundary imposed by the $\mathsf{tie}\, b$ operator. Indeed, the rule B1 excludes una from being $\mathsf{tie}\, b$, but not from being $\mathsf{tie}\, b'$, provided that $b \neq b'$. The same rule, with $\mathsf{una} = .b'$, implies that $F.b.b' \equiv F.b'.b$, *i.e.*, the commutativity of the buffer stuffing operator. Similarly, with $\mathsf{una} = .b$, the rules E1 and X1 lead to $\overline{E}.b \equiv \overline{E.b}$ and $\underline{E}.b \equiv \underline{E.b}$, while the combination of the rules OPR and OPL leads to $(F.b)\ \mathsf{bin}\ F' \equiv F\ \mathsf{bin}(F'.b)$, showing further how buffer stuffing may be moved around.

It may be observed that, due to the rules CON1-2, ENT and EX, the equivalence relation so defined is in fact a congruence for all the operators of the algebra. It is easy to see that the structural similarity relation is closed in the domain of expressions, in the sense that, if a box expression matches one of the sides of any rule then, the other side defines a legal box expression. Moreover, it preserves the types of box expressions (static or dynamic), and captures the fact that box expressions have the same net translation.

Proposition 6.3. Let F_1 and F_2 be box expressions.

CON1	$\dfrac{F \equiv F'}{F\,\mathsf{una} \equiv F'\,\mathsf{una}}$	CON2	$\dfrac{F_1 \equiv F_1', F_2 \equiv F_2'}{F_1\,\mathsf{bin}\,F_2 \equiv F_1'\,\mathsf{bin}\,F_2'}$
ENT	$\dfrac{E \equiv E'}{\overline{E} \equiv \overline{E'}}$	EX	$\dfrac{E \equiv E'}{\underline{E} \equiv \underline{E'}}$
OPL	$(F.b)\,\mathsf{bin}\,F' \equiv (F\,\mathsf{bin}\,F').b$	OPR	$F\,\mathsf{bin}(F'.b) \equiv (F\,\mathsf{bin}\,F').b$
E1	$\overline{E}\,\mathsf{una} \equiv \overline{E\,\mathsf{una}}$	X1	$\underline{E}\,\mathsf{una} \equiv \underline{E\,\mathsf{una}}$
B1	$(F.b)\,\mathsf{una} \equiv (F\,\mathsf{una}).b$ if $\mathsf{una} \neq \mathsf{tie}\,b$	IS1	$\overline{E_1; E_2} \equiv \overline{E_1}; E_2$
IS2	$\underline{E_1}; E_2 \equiv E_1; \overline{E_2}$	IS3	$E_1; \underline{E_2} \equiv \underline{E_1; E_2}$
IPAR1	$\overline{E_1 \| E_2} \equiv \overline{E_1} \| \overline{E_2}$	IPAR2	$\underline{E_1} \| \underline{E_2} \equiv \underline{E_1 \| E_2}$
IC1L	$\overline{E_1 \,\square\, E_2} \equiv \overline{E_1} \,\square\, E_2$	IC1R	$\overline{E_1 \,\square\, E_2} \equiv E_1 \,\square\, \overline{E_2}$
IC2L	$\underline{E_1} \,\square\, E_2 \equiv \underline{E_1 \,\square\, E_2}$	IC2R	$E_1 \,\square\, \underline{E_2} \equiv \underline{E_1 \,\square\, E_2}$
IIT1	$\overline{E_1 \circledast E_2} \equiv \overline{E_1} \circledast E_2$	IIT2	$\underline{E_1} \circledast E_2 \equiv \overline{E_1} \circledast E_2$
IIT3	$\underline{E_1} \circledast E_2 \equiv E_1 \circledast \overline{E_2}$	IIT4	$\overline{E_1} \circledast E_2 \equiv E_1 \circledast \overline{E_2}$
IIT5	$E_1 \circledast \underline{E_2} \equiv \underline{E_1 \circledast E_2}$		

Table 1. Structural similarity relation for ABC, where $b \in \mathbb{B}$, una stands for any unary ABC operator and bin stands for any binary ABC operator.

1. If $F_1 \equiv F_2$, then $\lfloor F_1 \rfloor \equiv \lfloor F_2 \rfloor$, $\lceil F_1 \rceil = \lceil F_2 \rceil$ and $\mathsf{box}(F_1) = \mathsf{box}(F_2)$.
2. If $\lfloor F_1 \rfloor \equiv \lfloor F_2 \rfloor$, then $\lceil F_1 \rceil = \lceil F_2 \rceil$ and $\lfloor \mathsf{box}(F_1) \rfloor = \lfloor \mathsf{box}(F_2) \rfloor$.
3. If $\lceil F_1 \rceil \equiv \lceil F_2 \rceil$, then $\lceil F_1 \rceil = \lceil F_2 \rceil$ and $\lceil \mathsf{box}(F_1) \rceil = \lceil \mathsf{box}(F_2) \rceil$.
4. If $F_1 \equiv \overline{F_2}$, then $\lfloor F_1 \rfloor \equiv \lfloor F_2 \rfloor$, $\lceil F_1 \rceil = \lceil F_2 \rceil$ and $\mathsf{box}(F_1) = \overline{\mathsf{box}(F_2)}$.
5. If $F_1 \equiv \underline{F_2}$, then $\lfloor F_1 \rfloor \equiv \lfloor F_2 \rfloor$, $\lceil F_1 \rceil = \lceil F_2 \rceil$ and $\mathsf{box}(F_1) = \underline{\mathsf{box}(F_2)}$.

Proof:
Follows from the definitions (in particular, table 1, equations (2) and the translation rules (5)) and propositions 4.3, 5.2 and 6.2, by induction on the derivation of the equivalence. □

A dynamic box expression F is *initial* (*final*) if $M^{ctr}_{\mathsf{box}(F)} = {}^{\circ}\mathsf{box}(F)$ (resp. $M^{ctr}_{\mathsf{box}(F)} = \mathsf{box}(F)^{\circ}$).

Proposition 6.4. Let F be a dynamic box expression.

1. F is initial *iff* $F \equiv \overline{\lfloor F \rfloor}$.
2. F is final *iff* $F \equiv \underline{\lfloor F \rfloor}$.

Proof:
The ($\Longleftarrow$) in part (1) can be shown as follows. From proposition 6.3(1) and the translation rules (5), it follows that $\mathsf{box}(F) = \mathsf{box}(\overline{\lfloor F \rfloor}) = \overline{\mathsf{box}(\lfloor F \rfloor)}$. Moreover, by theorem 6.1(2), $\mathsf{box}(\lfloor F \rfloor)$ is static and so ${}^\circ\mathsf{box}(F) = {}^\circ\mathsf{box}(\lfloor F \rfloor) = M^{ctr}_{\overline{\mathsf{box}(\lfloor F \rfloor)}} = M^{ctr}_{\mathsf{box}(F)}$, and so F is initial. For part (2) the proof is similar.

To prove the ($\Longrightarrow$) implications, we proceed by induction on the structure of F. In the base step, $F = \overline{E}$ or $F = \underline{E}$, where E is a static expression, the result clearly holds. In the inductive step, we assume that F is initial and consider two cases.

Case 1: $F = F'\,\mathsf{una}$, for some unary ABC operator una. Then, from F being initial and the translation rules (5) as well as proposition 4.5(4) and ${}^\circ(\mathsf{box}(F')\,\mathsf{una}) = {}^\circ\mathsf{box}(F')$, we have that F' is also initial. Hence, by the induction hypothesis, $F' \equiv \overline{\lfloor F' \rfloor}$. We then obtain, using the rules E1 and CON1, that $F = F'\,\mathsf{una} \equiv \overline{\lfloor F' \rfloor}\,\mathsf{una} \equiv \overline{\lfloor F' \rfloor\,\mathsf{una}} \equiv \overline{\lfloor F'\,\mathsf{una} \rfloor} \equiv \overline{\lfloor F \rfloor}$.

Case 2: $F = F'\,\mathsf{bin}\,F''$, for some binary ABC operator bin. We present the argument for choice; other operators can be dealt with in a similar way. From F being initial and the translation rules (5), it follows that $M^{ctr}_{\mathsf{box}(F') \,\square\, \mathsf{box}(F'')} = {}^\circ(\mathsf{box}(F') \,\square\, \mathsf{box}(F''))$. Hence, by lemma A.3(3) and without loss generality, $M^{ctr}_{\mathsf{box}(F')} = {}^\circ\mathsf{box}(F')$ and F'' is static and so $\lfloor F'' \rfloor = F''$. Thus F' is initial and so, by the induction hypothesis and the rules IC1L and CON2, we obtain $F = F' \,\square\, F'' \equiv \overline{\lfloor F' \rfloor} \,\square\, F'' \equiv \overline{\lfloor F' \rfloor \,\square\, F''} \equiv \overline{\lfloor F' \,\square\, F'' \rfloor} \equiv \overline{\lfloor F \rfloor}$. □

A static expression E is *tidy* if each occurrence of the unary buffer stuffing operator $.b$ is either one of the $.b_i$'s such that $E = E'.b_1 \cdots .b_k$, or it appears within a subexpression of the form $E'.b \cdots .b\,\mathsf{tie}\,b$. Moreover, a dynamic expression F is in *tidy* if $\lfloor F \rfloor$ is tidy and all occurrences of the overbar and underbar operators are applied to atomic actions. Note that thanks to the rules in table 1, one can always transform a given static or dynamic expression F into an expression $F' \equiv F$ which is tidy. Intuitively, in a tidy expression each occurrence of the buffer stuffing is pushed up the syntax tree as far as possible, and each overbar and underbar as much as possible down the syntax tree.

Proposition 6.5. Let $F = F'.b_1 \cdots .b_k$ be a tidy expression such that $F' = \alpha$ or $F' = \overline{\alpha}$ or $F' = \underline{\alpha}$ or $F' = F''\,\mathsf{una}$ or $F' = F''\,\mathsf{bin}\,F'''$, where $k \geq 0$ and una is an instance of scoping or buffer restriction. Then $M^{opb}_{\mathsf{box}(F)} = \{b_1, \ldots, b_k\}$ where each b_i identifies the unique open buffer place labelled by b_i. Moreover, in the last two cases $M^{opb}_{\mathsf{box}(F'')} = \varnothing$, while in the last case $M^{opb}_{\mathsf{box}(F''')} = \varnothing$.

Proof:
Follows directly from the definitions and translation rules (5). □

Proposition 6.6. Let F_1 and F_2 be box expressions such that $\lceil F_1 \rceil = \lceil F_2 \rceil$.

1. $\mathsf{box}(F_1) = \mathsf{box}(F_2)$ *iff* $F_1 \equiv F_2$.
2. $\mathsf{box}(F_1) = \overline{\mathsf{box}(F_2)}$ *iff* $F_1 \equiv \overline{F_2}$.
3. $\mathsf{box}(F_1) = \underline{\mathsf{box}(F_2)}$ *iff* $F_1 \equiv \underline{F_2}$.

Proof:
We first observe that parts (2) and (3) follow directly from part (1) and the translation rules (5). For part (1), the ($\Longleftarrow$) implication results from proposition 6.3, while the ($\Longrightarrow$) implication can be shown by induction on the structure of $\lceil F_1 \rceil = \lceil F_2 \rceil$. In the base case, we have $\lceil F_1 \rceil = \alpha = \lceil F_2 \rceil$ and the result is straightforward. In the inductive step, we consider two cases. Below, without loss of generality, we assume that both F_1 and F_2 are tidy.

Case 1: $\lceil F_1 \rceil = F' \,\mathsf{una}\, = \lceil F_2 \rceil$, where una is an instance of the scoping or the tie operators. Then, F_1 and F_2 are of the form $F_1 = F_1' \,\mathsf{una}\, .b_1 \cdots .b_k$ and $F_2 = F_2' \,\mathsf{una}\, .b_1' \cdots .b_l'$, where $\lceil F_1' \rceil = F' = \lceil F_2' \rceil$ and both F_1' and F_2' are tidy. By proposition 6.5, $\{b_1, \ldots, b_k\} = M^{opb}_{\mathsf{box}(F_1)} = M^{opb}_{\mathsf{box}(F_2)} = \{b_1', \ldots, b_l'\}$. Hence, by proposition 6.2(1) and the translation rules (5), $\mathsf{box}(F_1') \,\mathsf{una}\, = \mathsf{box}(F_2') \,\mathsf{una}$. Thus, by proposition 4.4, $\mathsf{box}(F_1') = \mathsf{box}(F_2')$. As a result, by the induction hypothesis, $F_1' \equiv F_2'$ and so $F_1 \equiv F_2$, by the rules in table 1.

Case 2: $\lceil F_1 \rceil = E' \,\mathsf{bin}\, E'' = \lceil F_2 \rceil$, for some binary ABC operator bin. We present the argument for the choice operator; other operators can be dealt with in a similar way. We have that $F_1 = (F_1' \Box F_1'').b_1 \cdots .b_k$ and $F_2 = (F_2' \Box F_2'').b_1' \cdots .b_l'$, where $\lceil F_1' \rceil = E' = \lceil F_2' \rceil$, $\lceil F_1'' \rceil = F'' = \lceil E_2'' \rceil$ and F_1', F_2', F_1'' and F_2'' are tidy. By proposition 6.5, $\{b_1, \ldots, b_k\} = M^{opb}_{\mathsf{box}(F_1)} = M^{opb}_{\mathsf{box}(F_2)} = \{b_1', \ldots, b_l'\}$. Hence, by the translation rules (5), $\mathsf{box}(F_1') \Box \mathsf{box}(F_1'') = \mathsf{box}(F_2') \Box \mathsf{box}(F_2'')$ and so, by proposition 5.1(3), $(\mathsf{box}(F_1'), \mathsf{box}(F_1'')) \equiv_{\Omega_\Box} (\mathsf{box}(F_2'), \mathsf{box}(F_2''))$. Now, by proposition 6.5, the open buffer places of all the nets in the last equivalence are empty (and so $\equiv^5_{\Omega_\Box}$ does not contribute in this case to this relationship). Hence, by the definition of $\equiv_{\Omega_\Box}$, we have two options: $\mathsf{box}(F_1') = \mathsf{box}(F_2')$ and $\mathsf{box}(F_1'') = \mathsf{box}(F_2'')$, where we can apply the induction hypothesis to get $F_1' \equiv F_2'$ and $F_1'' \equiv F_2''$, and then show the result using the rules in table 1; or, without loss of generality, F_1' is initial and F_1'' static while F_2' is static and F_2'' initial expressions. From proposition 6.4 it then follows that $F_1' \equiv \overline{F_2'}$ and $\overline{F_2''} \equiv F_1''$, and the result follows from the rules in table 1. □

The precondition $\lceil F_1 \rceil = \lceil F_2 \rceil$ is needed in the last result, which can be seen by taking $F_1 \stackrel{\text{df}}{=} a \,\mathsf{sc}\, a$ and $F_2 \stackrel{\text{df}}{=} a \,\mathsf{sc}\, a \,\mathsf{sc}\, b$. Indeed, in this case $\mathsf{box}(F_1) = \mathsf{box}(F_2)$, due to our rules for generating place identities and the fact that no transition is left in the nets by the scoping operations, but $F_1 \not\equiv F_2$.

Next we define the operational semantics of box expressions. It is defined by means of labelled transition systems, whose labels are multisets of actions. In order to compare this semantics with the denotational semantics, we will first define an auxiliary transition based operational semantics, whose labels are multisets of the names of transitions in the denotational semantics of a given box expression. Since this auxiliary operational semantics is defined using the denotational one, it will be relatively easy to prove its correspondence with the original operational semantics for the box defining the denotational semantics of the expression. We will then see that the (label based) operational semantics for box expressions can be obtained from the auxiliary transition based semantics, in the same way that the derived labelled transition semantics of a box can be obtained from the original operational semantics based on multisets of transitions. As a consequence, also the two label based semantics will coincide.

6.3. Transition based operational semantics

Consider the set $\mathbb{T}$ of all transition trees in the boxes derived through the box mapping. It is easy to check that each $t \in \mathbb{T}$ has always the same label in all the boxes derived through the box mapping where it occurs; we shall denote this label by $\lambda(t)$.

The first operational semantics we will consider has moves of the form $F \xrightarrow{U} F'$ such that F and F' are box expressions and $U \in \mathbb{U} \stackrel{\text{df}}{=} \mathsf{mult}(\mathbb{T})$. Our goal is to generate exactly those steps U which are legal for the boxes associated to F and F', as it will be proved in theorem 6.8.

Formally, we define a ternary relation $\longrightarrow$ which is the least relation comprising all $(F, U, F') \in \mathsf{aexpr} \times \mathbb{U} \times \mathsf{aexpr}$ such that the relations in table 2 hold (note that $F \xrightarrow{U} F'$ means that (F, U, F') belongs to $\longrightarrow$). In the definition of EOP we make no restriction on U_1 and U_2, but the domain of application of bin ensures that this rule will always be used with the correct combination of static/dynamic box expressions and so, for instance, in case of the choice operator, U_1 or U_2 will be empty.

EA1 $\quad \overline{a} \xrightarrow{\{v^a\}} \underline{a}$

EA2 $\quad \overline{ab^+} \xrightarrow{\{v^{ab^+}\}} \underline{ab^+.b}$

EA3 $\quad \overline{ab^-.b} \xrightarrow{\{v^{ab^-}\}} \underline{ab^-}$

EA4 $\quad \overline{ab^\pm.b} \xrightarrow{\{v^{ab^\pm}\}} \underline{ab^\pm.b}$

EQ1 $\quad F \xrightarrow{\varnothing} F$

EQ2 $\quad \dfrac{F \equiv F',\ F' \xrightarrow{U} F'',\ F'' \equiv F'''}{F \xrightarrow{U} F'''}$

EBUF $\quad \dfrac{F \xrightarrow{U} F'}{F.b \xrightarrow{U} F'.b}$

ETIE $\quad \dfrac{F \xrightarrow{U} F'}{F\ \mathsf{tie}\ b \xrightarrow{v^{\mathsf{tie}\ b} \blacktriangleleft U} F'\ \mathsf{tie}\ b}$

ESC $\quad \dfrac{D \xrightarrow{\{t_1, u_1, \ldots, t_k, u_k, z_1, \ldots, z_l\}} D'}{D\ \mathsf{sc}\ a \xrightarrow{\{y_1, \ldots, y_k, x_1, \ldots, x_l\}} D'\ \mathsf{sc}\ a}$ *where* $\begin{cases} \lambda(t_i) = a \in \mathbb{A} \\ \lambda(u_i) = \widehat{a} \\ \lambda(z_j) \notin \{a, \widehat{a}\} \\ y_i = v^{\mathsf{sc}\ a} \lhd \{t_i, u_i\} \\ x_j = v^{\mathsf{sc}\ a} \lhd z_j \\ for\ \ i \leq k,\ j \leq l \end{cases}$

EOP $\quad \dfrac{F_1 \xrightarrow{U_1} F_1',\ F_2 \xrightarrow{U_2} F_2'}{F_1\ \mathsf{bin}\ F_2 \xrightarrow{(v_1^{\mathsf{bin}} \blacktriangleleft U_1) \cup (v_2^{\mathsf{bin}} \blacktriangleleft U_2)} F_1'\ \mathsf{bin}\ F_2'}$

Table 2. Transition based operational semantics for ABC, where $a \in \mathbb{A}_\tau$ (except in ESC, where $a \in \mathbb{A}$), $b \in \mathbb{B}$ and bin stands for any binary ABC operator.

We now show that an empty move always relates two structurally equivalent box expressions.

Proposition 6.7. Let F and F' be two box expressions. Then $F \xrightarrow{\varnothing} F'$ *iff* $F \equiv F'$.

Proof:
($\Longrightarrow$) Consider the derivation tree $\mathcal{T}$ for $F \xrightarrow{\varnothing} F'$. Since no rule in table 2 can ever produce an empty step from a non-empty one, the leaves of $\mathcal{T}$ must refer to instances of the rule EQ1. We can then proceed by induction on the structure of $\mathcal{T}$, taking into account the rules CON1 and CON2 in table 1 (for applications of the rules EBUF, ETIE, ESC and EOP), and the fact that $\equiv$ is transitive (for applications of the rule EQ2).

($\Longleftarrow$) Follows from the rules EQ1 and EQ2, with $F' = F'' = F'''$. □

Next, we show that a move of the operational semantics transforms a box expression into another expression with a structurally equivalent underlying static expression, and the move generated is a valid step for the corresponding boxes. We interpret this as establishing the soundness of the operational semantics of box expressions. We then reverse the implication obtaining the completeness of the operational semantics.

Theorem 6.8. Let F be a box expression.

1. If $F \xrightarrow{U} F'$, then F' is a box expression such that $\mathsf{box}(F)\ [U\rangle\ \mathsf{box}(F')$ and $\lceil F \rceil = \lceil F' \rceil$.
2. If $\mathsf{box}(F)\ [U\rangle\ \Sigma$, then there is a box expression F' such that $\mathsf{box}(F') = \Sigma$ and $F \xrightarrow{U} F'$.

Proof:
(1) Consider the derivation tree $\mathcal{T}$ for $F \xrightarrow{U} F'$. We will show that part (1) holds for all the derivations associated with the nodes of $\mathcal{T}$. To start with, the leaves of $\mathcal{T}$ must refer to instances of the rules EQ1 and EA1-4, and the result holds due to proposition 5.5 and $\Sigma\ [\varnothing\rangle\ \Sigma$, for every box Σ. We can then proceed by induction on the structure of $\mathcal{T}$, in each node using the induction hypothesis, and the following: for a node where EQ2 was applied, we use proposition 5.3(1); for a node where EBUF was applied, we use the translation rules (5); for a node where ESC or ETIE was applied, we use proposition 5.4(1,2) and the translation rules (5); and for a node where EOP was applied, we use proposition 5.3 and the translation rules (5).

(2) If $U = \varnothing$ then, due to $\Sigma = \mathsf{box}(F)$, the result holds for $F' = F$. Otherwise, we proceed by induction on the maximal depth h of the transition trees in U (such an inductive argument is valid since U is a finite set of finite trees as M_Σ is ac-free).

In the base step ($h = 0$), the possible U's are determined as in proposition 5.5, and the result follows from the translation rules (5), EA1, EA2, EA3 EA4, EBUF EQ2, E1 and B1. In the inductive step ($h > 0$), due to the rules in table 1, one of the following cases holds.

Case 1: $F \equiv G\,\mathsf{una}$, where una is an instance of scoping or buffer restriction. Then the result holds by the induction hypothesis, proposition 5.4(1,2) and the rules ESC and ETIE.

Case 2: $F \equiv G\,\mathsf{tie}\,b.b \cdots .b$. Then we first apply proposition 5.4(3) to conclude that U is in fact an enabled step for $\mathsf{box}(G\,\mathsf{tie}\,b)$, and the result holds by the induction hypothesis, proposition 5.4(2) and the rules ETIE and EBUF.

Case 2: $F \equiv G\,\mathsf{bin}\,H$. where bin is an instance of a binary operator. Then the result holds by the induction hypothesis, propositions 5.3 the rule EOP and propositions 6.6 and 6.7. □

Table 3 shows the derivation of the step sequence representing scenario I in section 2, together with a part of the derivation tree for the move labelled by $\xrightarrow{\{t_2,t_4\}}$ (see also figure 6).

6.4. Consistency of the denotational and operational semantics

The consistency between the denotational and the operational semantics of box expressions will be formulated in terms of the transition systems they generate. This will be possible since, thanks to theorem 6.8, we are now in a position to relate transition systems generated by a box expression and the corresponding box.

$$
\begin{array}{lll}
\overline{s;(((pr^+ \circledast f)\|(pr^+ \circledast f)\|(cr^- \circledast f))\,\mathsf{tie}\,r)} & & \\
\equiv & \overline{s};(((pr^+ \circledast f)\|(pr^+ \circledast f)\|(cr^- \circledast f))\,\mathsf{tie}\,r) & \text{IS1} \\
\xrightarrow{\{t_1\}} & \underline{s};(((pr^+ \circledast f)\|(pr^+ \circledast f)\|(cr^- \circledast f))\,\mathsf{tie}\,r) & \text{EA1, EOP} \\
\equiv & s;\overline{(((pr^+ \circledast f)\|(pr^+ \circledast f)\|(cr^- \circledast f))\,\mathsf{tie}\,r)} & \text{IS2, E1} \\
\equiv & s;(((\overline{pr^+} \circledast f)\|(\overline{pr^+} \circledast f)\|(\overline{cr^-} \circledast f))\,\mathsf{tie}\,r) & \text{IPAR1, IIT1} \\
\xrightarrow{\{t_2,t_3\}} & s;(((\underline{pr^+.r} \circledast f)\|(\underline{pr^+.r} \circledast f)\|(\overline{cr^-} \circledast f))\,\mathsf{tie}\,r) & \text{EA2, EOP, ETIE} \\
\equiv & s;(((\underline{pr^+.r} \circledast f)\|(\underline{pr^+} \circledast f)\|(\overline{cr^-} \circledast f).r)\,\mathsf{tie}\,r) & \text{X1, OPL-R} \\
\equiv & s;(((\underline{pr^+.r} \circledast f)\|(\underline{pr^+} \circledast f)\|(\overline{cr^-}.r \circledast f))\,\mathsf{tie}\,r) & \text{OPL} \\
\equiv & s;(((\overline{pr^+}.r \circledast f)\|(\underline{pr^+} \circledast f)\|(\overline{cr^-.r} \circledast f))\,\mathsf{tie}\,r) & \text{IIT2, E1} \\
\xrightarrow{\{t_2,t_4\}} & s;(((\underline{pr^+.r}.r \circledast f)\|(\underline{pr^+} \circledast f)\|(\underline{cr^-} \circledast f))\,\mathsf{tie}\,r) & \text{EA2-3, EOP, ETIE} \\
\equiv & s;(((pr^+.r.r \circledast \overline{f})\|(pr^+ \circledast \overline{f})\|(cr^- \circledast \overline{f}))\,\mathsf{tie}\,r) & \text{X1, IIT3} \\
\xrightarrow{\{t_5,t_6,t_7\}} & s;(((pr^+.r.r \circledast \underline{f})\|(pr^+ \circledast \underline{f})\|(cr^- \circledast \underline{f}))\,\mathsf{tie}\,r) & \text{EA1, EOP, ETIE} \\
\equiv & \underline{s;(((pr^+.r.r \circledast f)\|(pr^+ \circledast f)\|(cr^- \circledast f))\,\mathsf{tie}\,r)} & \text{IIT5, IPAR2, X1}
\end{array}
$$

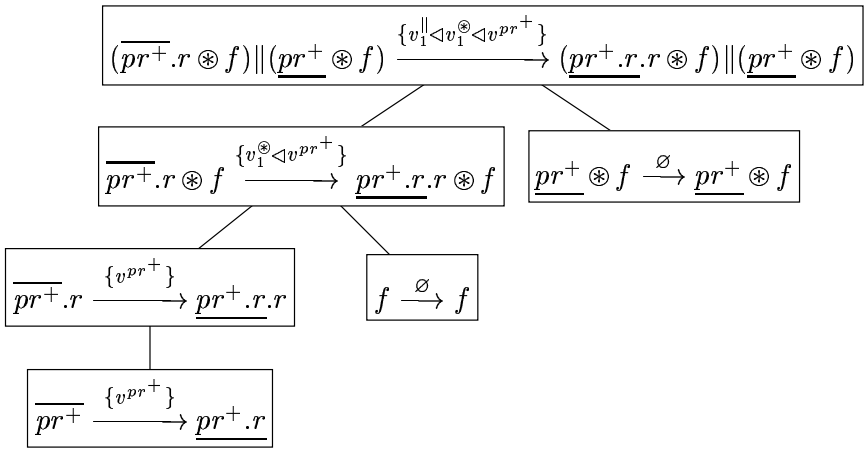


Table 3. Execution scenario I and part of the derivation tree for the move labelled by $\xrightarrow{\{t_2,t_4\}}$. The identities of various transitions are as follows: $t_1 = v_1^{\cdot} \lhd v^s$, $t_2 = v_2^{\cdot} \lhd v_1^{\mathsf{tie}\,r} \lhd v_1^{\|} \lhd v_1^{\|} \lhd v_1^{\circledast} \lhd v^{pr^+}$, $t_3 = v_2^{\cdot} \lhd v_1^{\mathsf{tie}\,r} \lhd v_1^{\|} \lhd v_2^{\|} \lhd v_1^{\circledast} \lhd v^{pr^+}$, $t_4 = v_2^{\cdot} \lhd v_1^{\mathsf{tie}\,r} \lhd v_2^{\|} \lhd v_1^{\circledast} \lhd v^{cr^-}$, $t_5 = v_2^{\cdot} \lhd v_1^{\mathsf{tie}\,r} \lhd v_1^{\|} \lhd v_1^{\|} \lhd v_2^{\circledast} \lhd v^f$, $t_6 = v_2^{\cdot} \lhd v_1^{\mathsf{tie}\,r} \lhd v_1^{\|} \lhd v_2^{\|} \lhd v_2^{\circledast} \lhd v^f$ and $t_7 = v_2^{\cdot} \lhd v_1^{\mathsf{tie}\,r} \lhd v_2^{\|} \lhd v_2^{\circledast} \lhd v^f$.

Let D be a dynamic box expression. We will use $\lceil D \rangle$ to denote all the box expressions derivable from D, *i.e.*, the least set of expressions containing D such that if $D' \in \lceil D \rangle$ and $D' \xrightarrow{U} D''$, for some $U \in \mathbb{U}$, then $D'' \in \lceil D \rangle$. Moreover, $[D]_{\equiv}$ will denote the equivalence class of $\equiv$ containing D. The *full transition system* of D is $\mathsf{fts}_D \stackrel{\text{df}}{=} (V, L, A, init)$, where $V \stackrel{\text{df}}{=} \{[D']_{\equiv} \mid D' \in \lceil D \rangle\}$ is the set of states; $L \stackrel{\text{df}}{=} \mathbb{U}$ is the set of arc labels; $A \stackrel{\text{df}}{=} \left\{([D']_{\equiv}, U, [D'']_{\equiv}) \in V \times \mathbb{U} \times V \mid D' \xrightarrow{U} D''\right\}$ is the set of arcs; and $init \stackrel{\text{df}}{=} [D]_{\equiv}$ is the initial state. For a static box expression E, $\mathsf{fts}_E \stackrel{\text{df}}{=} \mathsf{fts}_{\overline{E}}$.

Note that we base transition systems of box expressions on the equivalence classes of $\equiv$, rather than on box expressions themselves, since we may have $D \xrightarrow{\varnothing} D'$ for two equivalent but different expressions D and D', whereas in the domain of boxes, $\Sigma \lceil \varnothing \rangle \Theta$ always implies $\Sigma = \Theta$.

We now state a fundamental result which demonstrates that the operational and denotational semantics of a box expression capture the same behaviour, in arguably the strongest sense.

Theorem 6.9. For every box expression F, $\mathsf{iso}_F \stackrel{\text{df}}{=} \left\{([F']_{\equiv}, \mathsf{box}(F')) \mid [F']_{\equiv} \text{ is a node of } \mathsf{fts}_F\right\}$ is an isomorphism between the full transition systems fts_F and $\mathsf{fts}_{\mathsf{box}(F)}$. Moreover, iso_F preserves the property of being in an initial or final state.[10]

Proof:
Follows from theorem 6.8 (implying, in particular, that for each $D' \in \lceil D \rangle$, we have $\lceil D' \rceil = \lceil D \rceil$), propositions 6.6(1) and 6.4, and the translation rules (5). □

6.5. Label based operational semantics

First, we retain the structural similarity relation $\equiv$ on box expressions without any change. Next, we define moves of the form $F \xrightarrow{\Gamma} F'$, where F and F' are box expressions as before, and Γ is a finite multiset in $\mathbb{L} \stackrel{\text{df}}{=} \mathsf{mult}(\mathbb{A}_\tau)$, as shown in table 4. (Formally, we define a ternary relation $\longrightarrow$ which is the least relation comprising all $(F, \Gamma, F') \in \mathsf{aexpr} \times \mathsf{mult}(\mathbb{A}_\tau) \times \mathsf{aexpr}$ such that the relations in table 4 hold, and $F \xrightarrow{\Gamma} F'$ means that (F, Γ, F') belongs to $\longrightarrow$.)

The two operational semantics are clearly related; essentially, each label based move is a transition based move with only transitions labels being recorded.

Proposition 6.10. Let F be a box expression and $\Gamma \in \mathbb{L}$. Then $F \xrightarrow{\Gamma} F'$ *iff* there is $U \in \mathbb{U}$ such that $F \xrightarrow{U} F'$ and $\lambda(U) = \Gamma$.

Proof:
Follows directly from the definitions. □

The results concerning transition based operational semantics directly extend to the label based one. Let F be a box expression. In view of proposition 6.10, the label based operational semantics of F is faithfully captured by the *labelled transition system* of F, denoted by lts_F, and defined as fts_F with each arc label U changed to $\lambda(U)$. The consistency result for the label based operational semantics can then be formulated thus.

[10] In terms of box expression (*cf.* proposition 6.4) or box net (*cf.* section 3.3), not *w.r.t.* the transition system.

LA1	$\overline{a} \xrightarrow{\{a\}} \underline{a}$	LA2	$\overline{ab^+} \xrightarrow{\{a\}} \underline{ab^+.b}$
LA3	$\overline{ab^-.b} \xrightarrow{\{a\}} \underline{ab^-}$	LA4	$\overline{ab^\pm.b} \xrightarrow{\{a\}} \underline{ab^\pm.b}$
LQ1	$F \xrightarrow{\varnothing} F$	LQ2	$\dfrac{F \equiv F',\ F' \xrightarrow{\Gamma} F'',\ F'' \equiv F'''}{F \xrightarrow{\Gamma} F'''}$
LBUF	$\dfrac{F \xrightarrow{\Gamma} F'}{F.b \xrightarrow{\Gamma} F'.b}$	LTIE	$\dfrac{F \xrightarrow{\Gamma} F'}{F \,\mathsf{tie}\, b \xrightarrow{\Gamma} F' \,\mathsf{tie}\, b}$
LOP	$\dfrac{F_1 \xrightarrow{\Gamma_1} F_1',\ F_2 \xrightarrow{\Gamma_2} F_2'}{F_1 \,\mathsf{bin}\, F_2 \xrightarrow{\Gamma_1+\Gamma_2} F_1' \,\mathsf{bin}\, F_2'}$	LSC	$\dfrac{D \xrightarrow{\Gamma+k\cdot\{a,\widehat{a}\}} D'}{D \,\mathsf{sc}\, a \xrightarrow{\Gamma+k\cdot\{\tau\}} D' \,\mathsf{sc}\, a}\ a, \widehat{a} \notin \Gamma,\ k \in \mathbb{N}$

Table 4. Label based operational semantics for ABC, where $a \in \mathbb{A}_\tau$ (except in LSC, where $a \in \mathbb{A}$), $b \in \mathbb{B}$ and bin stands for any binary ABC operator.

Theorem 6.11. For every box expression F, $\mathsf{iso}_F \stackrel{\text{df}}{=} \left\{ ([F']_\equiv, \mathsf{box}(F')) \;\middle|\; [F']_\equiv \text{ is a node of } \mathsf{lts}_F \right\}$ is an isomorphism between the labelled transition systems lts_F and $\mathsf{lts}_{\mathsf{box}(F)}$. Moreover, iso_F preserves the property of being in an initial or final state.[11]

Proof:
Follows from the definitions, theorem 6.9 and proposition 6.10. □

The rules of the label based operational semantics are put into work in tables 5 and 6, where we use the second and third scenarios introduced in section 2.

To further illustrate how the buffer restriction operator controls the way in which tokens in the buffer places may be used, let us consider the sixth line in table 5:

$$(pr^+; (((\overline{pr^+} \circledast f) \| (\overline{cr^-} \circledast f)) \,\mathsf{tie}\, r; cr^-.r)) \,\mathsf{tie}\, r \ .$$

Notice that if we were able to move the $.r$ from the end of the expression towards $\overline{cr^-}$, resulting in $\overline{cr^-.r}$, then we would have been able to execute $\overline{cr^-}$. However, the B1 rule has been designed so that it is impossible to move $.r$ inside the scope of the internal tie r operator. This is fully consistent with the net semantics of the corresponding box shown in the middle of figure 5 page 9 which, after executing the topmost p-labelled transition cannot execute the c-labelled transition whose input place is the internal b-labelled place.

[11] In terms of box expression (*cf.* proposition 6.4) or box net (*cf.* section 3.3), not *w.r.t.* the transition system.

$\overline{(pr^+;(((pr^+ \circledast f)\|(cr^- \circledast f))\,\mathsf{tie}\,r;cr^-))\,\mathsf{tie}\,r}$		
$\equiv$	$(\overline{pr^+};(((pr^+ \circledast f)\|(cr^- \circledast f))\,\mathsf{tie}\,r;cr^-))\,\mathsf{tie}\,r$	E1, IS1
$\xrightarrow{\{p\}}$	$(\underline{pr^+.r};(((pr^+ \circledast f)\|(cr^- \circledast f))\,\mathsf{tie}\,r;cr^-))\,\mathsf{tie}\,r$	LA1, LOP, LTIE
$\equiv$	$(\underline{pr^+};(((pr^+ \circledast f)\|(cr^- \circledast f))\,\mathsf{tie}\,r;cr^-.r))\,\mathsf{tie}\,r$	X1, OPL-R
$\equiv$	$(pr^+;\overline{(((pr^+ \circledast f)\|(cr^- \circledast f))\,\mathsf{tie}\,r;cr^-.r)})\,\mathsf{tie}\,r$	IS1-2, E1
$\equiv$	$(pr^+;(((\overline{pr^+} \circledast f)\|(\overline{cr^-} \circledast f))\,\mathsf{tie}\,r;cr^-.r))\,\mathsf{tie}\,r$	IPAR1, IIT1
$\xrightarrow{\{p\}}$	$(pr^+;(((\underline{pr^+.r} \circledast f)\|(\overline{cr^-} \circledast f))\,\mathsf{tie}\,r;cr^-.r))\,\mathsf{tie}\,r$	LA1, LOP, LTIE
$\equiv$	$(pr^+;(((\overline{pr^+}.r \circledast f)\|(\overline{cr^-} \circledast f))\,\mathsf{tie}\,r;cr^-.r))\,\mathsf{tie}\,r$	IIT2, E1
$\xrightarrow{\{p\}}$	$(pr^+;(((\underline{pr^+.r}.r \circledast f)\|(\overline{cr^-} \circledast f))\,\mathsf{tie}\,r;cr^-.r))\,\mathsf{tie}\,r$	LA1, LOP, LTIE
$\equiv$	$(pr^+;(((\underline{pr^+.r} \circledast f)\|(\overline{cr^-.r} \circledast f))\,\mathsf{tie}\,r;cr^-\,\mathsf{tie}\,r$	OPL-R, E1
$\xrightarrow{\{c\}}$	$(pr^+;(((\underline{pr^+.r} \circledast f)\|(\underline{cr^-} \circledast f))\,\mathsf{tie}\,r;cr^-.r))\,\mathsf{tie}\,r$	LA3, LOP, LTIE
$\equiv$	$(pr^+;(((pr^+.r \circledast \overline{f})\|(cr^- \circledast \overline{f}))\,\mathsf{tie}\,r;cr^-.r))\,\mathsf{tie}\,r$	IIT3
$\xrightarrow{\{f,f\}}$	$(pr^+;(((pr^+.r \circledast \underline{f})\|(cr^- \circledast \underline{f}))\,\mathsf{tie}\,r;cr^-.r))\,\mathsf{tie}\,r$	LA1, LOP, LTIE
$\equiv$	$(pr^+;(((pr^+.r \circledast f)\|(cr^- \circledast f))\,\mathsf{tie}\,r;\overline{cr^-.r}))\,\mathsf{tie}\,r$	IIT5, IPAR2, X1, IS2
$\xrightarrow{\{c\}}$	$(pr^+;(((pr^+.r \circledast f)\|(cr^- \circledast f))\,\mathsf{tie}\,r;\underline{cr^-}))\,\mathsf{tie}\,r$	LA3, LOP, LTIE
$\equiv$	$\underline{(pr^+;(((pr^+.r \circledast f)\|(cr^- \circledast f))\,\mathsf{tie}\,r;cr^-))\,\mathsf{tie}\,r}$	IS3, X1

Table 5. Execution scenario II.

7. Conclusions

In this paper, we proposed a framework which supports two consistent (in a very strong sense, since the corresponding transition systems are isomorphic and not only bisimilar) concurrent semantics for a class of process expressions with both synchronous and asynchronous communication. Since the unbounded nature of the buffer places means that 1-safeness can no longer be a feature of the resulting nets, we replaced it by auto-concurrency-freeness which is still a theoretically well-motivated property. As a direct consequence of this decision, some awkward constraints introduced in the previous box algebra models in order to remain framework of 1-safe nets are no longer necessary. Note that the model we obtained can be used, in particular, to give the semantics of a programming language with timing constraints and exceptions [18, 19].

With respect to other process algebras, one can make the following observations. First, contrary to what happens in CCS and ACP-like algebras (see for instance [1]), in PBC-like algebras a process expression keeps its underlying structure (here captured by the $\lceil . \rceil$ skeleton, see theorem 6.8(1)) when evolving. This reflects the fact that, in the corresponding Petri net, the structure of the net does not change under the firing rule: only the marking evolves; and, indeed, the dynamic features found in box expressions (*i.e.*, the overbars, underbars and $.b$'s) cope precisely with the dynamic changes of the current marking of the system. Hence, with this view in mind, most of the definitions, features and results of

	$\overline{(pr^+ \circledast f) \| (cr^- \circledast f) \| (tr^\pm \circledast f) \| (tr^\pm \circledast f)}$	
$\equiv$	$(\overline{pr^+} \circledast f) \| (\overline{cr^-} \circledast f) \| (\overline{tr^\pm} \circledast f) \| (\overline{tr^\pm} \circledast f)$	IPAR1, IIT1
$\xrightarrow{\{p\}}$	$(\underline{pr^+.r} \circledast f) \| (\overline{cr^-} \circledast f) \| (\overline{tr^\pm} \circledast f) \| (\overline{tr^\pm} \circledast f)$	LA2, LOP
$\equiv$	$(\overline{pr^+.r} \circledast f) \| (\overline{cr^-} \circledast f) \| (\overline{tr^\pm} \circledast f) \| (\overline{tr^\pm} \circledast f)$	IIT2, E1
$\xrightarrow{\{p\}}$	$(\underline{pr^+.r.r} \circledast f) \| (\overline{cr^-} \circledast f) \| (\overline{tr^\pm} \circledast f) \| (\overline{tr^\pm} \circledast f)$	LA2, LOP
$\equiv$	$(\overline{pr^+} \circledast f) \| (\overline{cr^-.r} \circledast f) \| (\overline{tr^\pm.r} \circledast f) \| (\overline{tr^\pm} \circledast f)$	OPL-R, IIT2, E1
$\xrightarrow{\{c,p,t\}}$	$(\underline{pr^+.r} \circledast f) \| (\underline{cr^-} \circledast f) \| (\underline{tr^\pm.r} \circledast f) \| (\overline{tr^\pm} \circledast f)$	LA2-4, LOP
$\equiv$	$(\overline{pr^+} \circledast f) \| (\overline{cr^-} \circledast f) \| (\overline{tr^\pm.r} \circledast f) \| (\overline{tr^\pm.r} \circledast f)$	IIT2, IPAR1-2, OPL-R, E1
$\xrightarrow{\{t,t\}}$	$(\overline{pr^+} \circledast f) \| (\overline{cr^-} \circledast f) \| (\underline{tr^\pm.r} \circledast f) \| (\underline{tr^\pm.r} \circledast f)$	LA4, LOP

Table 6. Execution scenario III.

the present paper are perhaps not difficult to grasp intuitively, but the details of the proofs needed to underpin this intuition were less easy to obtain. This also allowed to get rather strong results about the relationship between the process algebraic and Petri net views, leading to isomorphic transition systems, and not only to homomorphisms or bisimultions for a fragment of the theory. The idea to use 'floating' buffer symbols flowing more or less freely between the subterms and representing dangling asynchronous messages is not very surprising either (see for instance [8]), but its interplay with the tie operator was less standard and its handling required some care. Finally, the rules EOP in table 2 and LOP in table 4 may look surprising in the context of the standard process algebras. The reason is that most of the operational semantics is in fact hidden in the structural equivalence $\equiv$, which is much stronger than a bisimulation relation (like, for instance, that in the π-calculus, see [24]); in particular, it exactly captures those views of the system that may indeed generate evolutions (static subexpressions are 'frozen' in each view), and correctly identifies process expressions with the same corresponding Petri nets.

Acknowledgements

We would like to thank the three anonymous referees for their very thorough reviews, and for suggesting several truly useful improvements and modifications. This research was partially supported by the ARC JIP and EPSRC BEACON projects.

References

[1] J. Baeten and W. P. Weijland: *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press (1990).

[2] T. Basten and M. Voorhoeve: An Algebraic Semantics for Hierarchical P/T Nets. Proc. of *ICATPN'95*, G. DeMichelis and M. Diaz (Eds.). Springer, Lecture Notes in Computer Science 935 (1995) 45–65.

[3] E. Best, R. Devillers and J. Hall: The Petri Box Calculus: a New Causal Algebra with Multilabel Communication. In: *Advances in Petri Nets 1992*, G. Rozenberg (Ed.). Springer, Lecture Notes in Computer Science 609 (1992) 21–69.

[4] E. Best, R. Devillers and M. Koutny: A Unified Model for Nets and Process Algebras. In: *Handbook of Process Algebra*, J. A. Bergstra, A. Ponse, S. A. Smolka, (Eds.). Elsevier (2001) 873–944.

[5] E. Best, R. Devillers and M. Koutny: *Petri Net Algebra*. EATCS Monographs on TCS, Springer (2001).

[6] E. Best and R. P. Hopkins: B(PN)^2 – a Basic Petri Net Programming Notation. Proc. of *PARLE '93*, A. Bode, M. Reeve and G. Wolf (Eds.). Springer, Lecture Notes in Computer Science 694 (1993) 379–390.

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

[8] M. Broy and E-R. Olderog: Trace-Oriented Models of Concurrency. In: *Handbook of Process Algebra*, J. A. Bergstra, A. Ponse, S. A. Smolka, (Eds.). Elsevier (2001) 101–195.

[9] 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.

[10] R. Devillers, H. Klaudel, M. Koutny, E. Pelz and F. Pommereau: Operational Semantics for PBC with Asynchronous Communication. Proc. of *HPC'02*, A. Tentner (Ed.). SCS (2002) 314–319.

[11] R. J. van Glabbeek and F. V. Vaandrager: Petri Net Models for Algebraic Theories of Concurrency. Proc. of *PARLE'87*, J. W. de Bakker, A. J. Nijman and P. C. Treleaven (Eds.). Springer, Lecture Notes in Computer Science 259 (1987) 224–242.

[12] U. Goltz and R. Loogen: A Non-interleaving Semantic Model for Nondeterministic Concurrent Processes. *Fundamentae Informaticae* 14 (1991) 39–73.

[13] R. Gorrieri and U. Montanari: On the Implementation of Concurrent Calculi in Net Calculi: two Case Studies. *Theoretical Computer Science* 141(1-2) (1995) 195–252.

[14] C. A. R. Hoare: *Communicating Sequential Processes*. Prentice Hall (1985).

[15] P. W. Hoogers, H. C. M. Kleijn and P. S. Thiagarajan: An Event Structure Semantics for General Petri Nets. *Theoretical Computer Science* 153 (1996) 129–170.

[16] R. Janicki and P. E. Lauer: *Specification and Analysis of Concurrent Systems - the COSY Approach*. EATCS Monographs on TCS, Springer (1992).

[17] H. Klaudel: Parameterized M-expression semantics of parallel procedures. Proc. of *DAPSYS'00*, Kluwer Academic Publishers (2000) 85–94.

[18] H. Klaudel and F. Pommereau: Asynchronous links in the PBC and M-nets. Proc. of *ASIAN'99*, P. S. Thiagarajan and R. Yap (Eds.). Springer, Lecture Notes in Computer Science 1742 (1999) 190–200.

[19] H. Klaudel and F. Pommereau: A concurrent and Compositional Petri Net Semantics of Preemption. Proc. of *IFM'2000*, W. Grieskamp, T. Santen and B. Stoddart (Eds.). Springer, Lecture Notes in Computer Science 1945 (2000) 318–337.

[20] M. Koutny and E. Best: Fundamental Study: Operational and Denotational Semantics for the Box Algebra. *Theoretical Computer Science* 211 (1999) 1–83.

[21] R. Milner: *Communication and Concurrency*. Prentice Hall (1989).

[22] U. Montanari and D. Yankelevich: Combining CCS and Petri Nets via Structural Axioms. *Fundamentae Informaticae* 20(1-3) (1994) 193–229.

[23] E. R. Olderog: *Nets, Terms and Formulas*. Cambridge Tracts in Theoretical Computer Science 23, Cambridge University Press (1991).

[24] J. Parrow: An Introduction to the π-Calculus. In: *Handbook of Process Algebra*, J. A. Bergstra, A. Ponse, S. A. Smolka, (Eds.). Elsevier (2001) 479–543.

[25] G. D. Plotkin: A Structural Approach to Operational Semantics. Technical Report FN-19, Computer Science Department, University of Aarhus (1981).

[26] F. Pommereau: FIFO buffers in tie sauce. Proc. of *DAPSYS'00*, Kluwer Academic Publishers (2000) 95–104.

[27] W. Reisig: Deterministic Buffer Synchronization of Sequential Processes. *Acta Informatica* 18 (1982) 117–134.

[28] W. Reisig: *Petri Nets. An Introduction*. EATCS Monographs on TCS, Springer (1985).

[29] D. Taubner: *Finite Representation of CCS and TCSP Programs by Automata and Petri Nets*. Springer, Lecture Notes in Computer Science 369 (1989).

A. Generalisations of the results from the main body of the paper

In order to avoid proving properties separately for each of the ABC operators, and also to prepare the ground for possible future extensions of ABC, we will first slightly extend the framework.

First, $\Omega_{\Box}$, $\Omega_{\circledast}$ and $\Omega_{;}$ will be considered as instances of a more general class of operator boxes. A *sequential* operator box Ω_{sq} is an operator box such that: no place is isolated; there is exactly one e-labelled place, and exactly one x-labelled place; and, for every transition $v \in T_{\Omega_{sq}}$, $|{}^{\bullet}v| = |v^{\bullet}| = 1$ and $\lambda_{\Omega_{sq}}(v) = \varphi_{id}$. That is, Ω_{sq} can be thought of as a *finite automaton* in which each transition will be substituted by a potentially complex box by the net substitution operation. We assume that all the transitions have basic transition identities, and that two distinct operator boxes have disjoint sets of nodes. The domain of application of Ω_{sq} is the set $dom_{\Omega_{sq}}$ comprising all Ω_{sq}-tuples of static and dynamic boxes such that at most one box is dynamic. An example of a sequential operator box (other than those mentioned above) is Ω_{*} shown on the left of figure 11, which was used in [5] to model an iteration construct with explicit initialisation and termination.

Similarly, scoping can be considered as an instance of a more general class of operator boxes. A unary *communication interface* operator box Ω_{φ}, shown on the right of figure 11, is parameterised by an interface function $\varphi : \mathsf{mult}(\mathbb{A}_{\tau})\backslash\{\varnothing\} \to \mathbb{A}_{\tau}$, and has the domain of application $dom_{\Omega_{\varphi}} \stackrel{\text{df}}{=} \mathsf{abox}^{stc} \cup \mathsf{abox}^{dyn}$. To ensure the finiteness of the nets created, we assume that there is no finite set of labels $A \subseteq \mathbb{A}_{\tau}$ such that $\mathsf{mult}(A) \cap dom_{\varphi}$ is infinite; it may be checked that $\varphi_{\mathsf{sc}\,a}$ fulfills this constraint. The role of Ω_{φ} will be to effect the change of synchronous communication interface specified by φ.

In what follows the sequential operator boxes, communication interface operator boxes, buffer restriction operator boxes, and the parallel composition operator box will be referred to as *general* operator boxes.[12]

[12] The current framework as well as results could be extended to other operator boxes, but extra conditions (similar to those discussed in [5]) would then be necessary in order not to lose the properties established for ABC. However, the theory is already expressive as it stands at the moment as it can be used to model the standard construct used by high level concurrent programming languages.

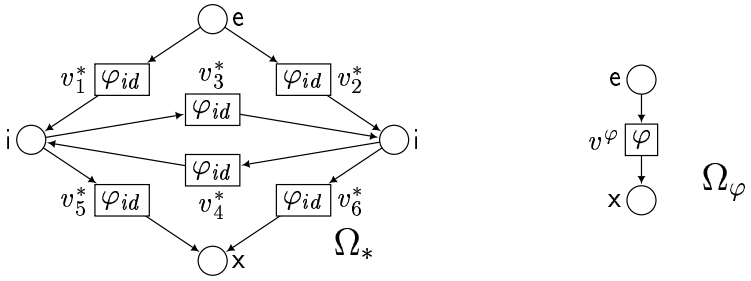


Figure 11. Sequential and interface operator boxes.

A.1. Net substitution

The net substitution operation defined in section 4.4 for the binary ABC operator boxes may be readily extended to operators of any arity.

Let Ω be a sequential operator with transitions $v_1, \ldots, v_n$, and $\boldsymbol{\Sigma} = (\Sigma_1, \ldots, \Sigma_n) = (\Sigma_{v_1}, \ldots, \Sigma_{v_n})$ be a tuple of boxes in dom_Ω. Then $\Omega(\boldsymbol{\Sigma}) = \Phi$ is a labelled net whose components are defined as in section 4.4 for a binary ABC operator Ω, with the following two modifications:

- The set of transitions of Φ is the set of all trees $v_i \lhd t$ with $t \in T_{\Sigma_i}$ and $i \in \{1, \ldots, n\}$.
- For every $b \in \mathbb{B}$, there is a unique b-labelled place $p^b \stackrel{\text{df}}{=} (p^b_{v_1}, \ldots, p^b_{v_n}) \in S_{\Omega(\boldsymbol{\Sigma})}$, where each $p^b_{v_i}$ is the unique b-labelled place of Σ_i. The marking of p^b is the sum of the markings of the $p^b_{v_i}$'s.

Moreover, in the proofs, we will denote by $\mathbb{P}\langle s\rangle$ the set of all places p as defined just before the formula (1) in section 4.4, for a given s; and by $\mathbb{P}\langle s, q\rangle$ (resp. $\mathbb{P}\langle s, q, q'\rangle$) the sets of all those $p \in \mathbb{P}\langle s\rangle$ in which q (resp. q and q') is present.

For a communication interface operator Ω_φ, the intuition behind a multiset Γ in the domain of φ is that some interface change can be applied to any finite set of transitions whose labels match the argument, *i.e.*, the non-empty multiset of actions Γ. More precisely, such transitions can be synchronised to yield a new transition labelled $\varphi(\Gamma)$. (Note that, since sequential operators as well as the parallel one, use the interface function φ_{id}, no transition label is changed for them.) Hence, the application of a communication interface operator Ω_φ to a box Σ results in a labelled net which is like Σ with the only difference that the set of transitions comprises all trees $t \stackrel{\text{df}}{=} v^\varphi \lhd \{t_1, \ldots, t_l\}$ such that $\{t_1, \ldots, t_l\} \in \mathsf{mult}(T_\Sigma)$ and the multiset $\Lambda \stackrel{\text{df}}{=} \{\lambda_\Sigma(t_1), \ldots, \lambda_\Sigma(t_l)\}$ belongs to the domain of φ. The label of t is $\varphi(\Lambda)$, and for a place p of $\Omega_\varphi(\Sigma)$, which is also a place of Σ, the weight function is given by:

$$W_{\Omega_\varphi(\Sigma)}(p, t) \stackrel{\text{df}}{=} \sum_{i=1}^{l} W_\Sigma(p, t_i) \ ,$$

and similarly for $W_{\Omega_\varphi(\Sigma)}(t, p)$.

We then obtain the following proposition, lemmata and general result on net substitution.

Proposition A.1. Let Ω be any sequential operator box or the parallel composition operator box $\Omega_{||}$, $\boldsymbol{\Sigma}$ be any Ω-tuple of boxes, and Σ be any box.

1. $M^{ctr}_{\Omega(\boldsymbol{\Sigma})} = \varnothing$ *iff* $M^{ctr}_{\Sigma_{v_i}} = \varnothing$, for each $v_i \in T_\Omega$.
   Note: This property does not rely on the fact that $\boldsymbol{\Sigma}$ belongs to dom_Ω, and it would hold for any operator based on net substitution.

2. If Σ' is $\Omega_\varphi(\Sigma)$ or Σ tie b or $\Sigma.B$, then $M^{ctr}_{\Sigma'} = \varnothing$ *iff* $M^{ctr}_{\Sigma} = \varnothing$.

Proof:
(1) Follows from the definitions and, in particular, from the control-restrictedness of Ω and the ex-restrictedness of each box Σ_{v_i}. Indeed, due to such constraints, for every $e \in {}^\circ\Sigma_{v_i}$, if $s \in {}^\bullet v_i$ then each $p \in \mathbb{P}\langle s, e\rangle \neq \varnothing$ is a control place in $\Omega(\boldsymbol{\Sigma})$ such that $M_{\Sigma_{v_i}}(e) \leq M_{\Omega(\boldsymbol{\Sigma})}(p)$. The situation is similar for $x \in \Sigma^\circ_{v_i}$. Moreover, the internal places of Σ_{v_i} and their markings remain unchanged.

(2) Follows immediately from the fact that the control places of Σ and their markings remain unchanged. □

Lemma A.2. If M is a clean marking of a box Σ such that $M(e) + M(x) \geq 1$, for all $e \in {}^\circ\Sigma$ and $x \in \Sigma^\circ$, then $M^{ctr} \in \{{}^\circ\Sigma, \Sigma^\circ\}$.

Proof:
Suppose that $M^{ctr} \neq {}^\circ\Sigma$ and $M^{ctr} \neq \Sigma^\circ$. Then, since M is clean, there are $e \in {}^\circ\Sigma$ and $x \in \Sigma^\circ$ such that $M(e) = M(x) = 0$, a contradiction. □

Lemma A.3. Let $\Omega_{sq}(\boldsymbol{\Sigma})$ be a legal application of a sequential operator box Ω_{sq}, and $z \in T_{\Omega_{sq}}$ be such that $\Sigma = \Sigma_z$ is a dynamic box. Moreover, let s'' be a place in Ω_{sq}, $M \stackrel{\text{df}}{=} M_{\Omega_{sq}(\boldsymbol{\Sigma})}$, ${}^\bullet z = \{s\}$ and $z^\bullet = \{s'\}$.

1. If there is a place in $\mathbb{P}\langle s''\rangle$ which is marked at M then $s'' \in \{s, s'\}$.

2. If $M^{ctr}_\Sigma = {}^\circ\Sigma$ then $M^{ctr} = \mathbb{P}\langle s\rangle$, and if $M^{ctr}_\Sigma = \Sigma^\circ$ then $M^{ctr} = \mathbb{P}\langle s'\rangle$.

3. If $M^{ctr} \geq \mathbb{P}\langle s''\rangle$ then $M^{ctr} = \mathbb{P}\langle s''\rangle$, and one of the following holds:

   (a) $s'' = s \neq s'$ and $M^{ctr}_\Sigma = {}^\circ\Sigma$.

   (b) $s'' = s' \neq s$ and $M^{ctr}_\Sigma = \Sigma^\circ$.

   (c) $s'' = s = s'$ and $M^{ctr}_\Sigma \in \{{}^\circ\Sigma, \Sigma^\circ\}$.

Proof:
(1) Follows from the definition of $\Omega_{sq}(\boldsymbol{\Sigma})$ and the fact that all control places in the nets Σ_v, for $v \neq z$, are empty.

(2) Suppose that $M^{ctr}_\Sigma = {}^\circ\Sigma$ (the case $M^{ctr}_\Sigma = \Sigma^\circ$ is symmetric). Then the control places marked in $\Omega_{sq}(\boldsymbol{\Sigma})$ are exactly those in whose construction the places from ${}^\circ\Sigma$ were used. Moreover, $M(q) = M_\Sigma(e)$, for all $e \in {}^\circ\Sigma$ and $q \in \mathbb{P}\langle s, e\rangle$. Hence $M^{ctr}_\Sigma = \mathbb{P}\langle s\rangle$.

(3) From (1) it follows that $s'' \in \{s, s'\}$. Suppose that $s'' = s \neq s'$ (the case $s'' = s' \neq s$ is symmetric). Then $M(q) = M_\Sigma(e)$, for all $e \in {}^\circ\Sigma$ and $q \in \mathbb{P}\langle s'', e\rangle$. Hence $M_\Sigma \geq {}^\circ\Sigma$, and so $M^{ctr}_\Sigma = {}^\circ\Sigma$ since M_Σ is clean. Moreover, by part (2), $M^{ctr} = \mathbb{P}\langle s''\rangle$.

Suppose now that $s'' = s = s'$. Then $1 \leq M(q) = M_\Sigma(e) + M_\Sigma(x)$, for all $e \in {}^\circ\Sigma$, $x \in \Sigma^\circ$ and $q \in \mathbb{P}\langle s'', e, x\rangle$. Hence, by lemma A.2, $M^{ctr}_\Sigma \in \{{}^\circ\Sigma, \Sigma^\circ\}$. Moreover, by part (2), $M^{ctr} = \mathbb{P}\langle s''\rangle$. □

Theorem A.4. Let Ω be any general operator box and $\boldsymbol{\Sigma} \in dom_\Omega$. Then $\Omega(\boldsymbol{\Sigma})$ is a box with a clean and ac-free marking. Moreover, if all the dynamic boxes (if any) in $\boldsymbol{\Sigma}$ have quasi-safe markings, then the marking of $\Omega(\boldsymbol{\Sigma})$ is also quasi-safe.

Proof:
The result is straightforward for operators other than the sequential ones, so suppose that Ω is such an operator. Then $\Omega(\boldsymbol{\Sigma})$ is ex-restricted since $\mathbb{P}\langle s\rangle \neq \varnothing$ for every $s \in S_\Omega$ (due to the ex-restrictedness of the boxes in $\boldsymbol{\Sigma}$) and, in particular, for $s \in {}^\circ\Omega \cup \Omega^\circ$. Moreover, the $\mathbb{B}$-restrictedness follows directly from definitions, and the control-restrictedness from the same property of the boxes in $\boldsymbol{\Sigma}$, and the fact that $\mathbb{P}\langle s,p\rangle \neq \varnothing$ whenever $s \in S_\Omega$ and $p \in {}^\circ\Sigma_v$ for $v \in s^\bullet$ (or $p \in \Sigma_v^\circ$ for $v \in {}^\bullet s$). That $M_{\Omega(\boldsymbol{\Sigma})}$ is clean follows from lemma A.3(3), with $\{s\} = {}^\circ\Omega$ or $\{s\} = \Omega^\circ$. To prove that it is ac-free (and quasi-safe), we proceed as follows: whenever the additional hypothesis is fulfilled, if all the components of $\boldsymbol{\Sigma}$ are static boxes then, by definition, all the control places of $\Omega(\boldsymbol{\Sigma})$ are unmarked and the property follows immediately, from the control-restrictedness of $\Omega(\boldsymbol{\Sigma})$. So, let us assume that Σ_w is the unique dynamic box in $\boldsymbol{\Sigma}$ and take a transition $v \lhd t$ of $\Omega(\boldsymbol{\Sigma})$. We distinguish two cases:

Case 1: $v \neq w$. If ${}^\bullet t$ contains an internal (i-labelled) place p, then p is also an internal place of $\Omega(\boldsymbol{\Sigma})$, $p \in {}^\bullet(v \lhd t)$ and $M_{\Omega(\boldsymbol{\Sigma})}(p) = M_{\Sigma_v}(p) = 0$. Otherwise, ${}^\bullet t$ contains a place p in ${}^\circ\Sigma_v \cup \Sigma_v^\circ$, and we assume that $p \in {}^\circ\Sigma_v$ (the case $p \in \Sigma_v^\circ$ is symmetric). We also assume that ${}^\bullet v = \{s\}$, and so $\mathbb{P}\langle s,p\rangle \subseteq {}^\bullet(v \lhd t)$. Then we have the following cases:

- If $s \notin {}^\bullet w \cup w^\bullet$ then, by the definition of $\Omega(\boldsymbol{\Sigma})$, for every $q \in \mathbb{P}\langle s,p\rangle$ we have $M_{\Omega(\boldsymbol{\Sigma})}(q) = 0$.
- If $w \in {}^\bullet s \setminus s^\bullet$ then, due to the cleanness of Σ_w, either $M^{ctr}_{\Sigma_w} = \Sigma_w^\circ$, or there is $x \in \Sigma_w^\circ$ such that $M_{\Sigma_w}(x) = 0$. In the former case, for every $q \in \mathbb{P}\langle s,p\rangle$, we have $M_{\Omega(\boldsymbol{\Sigma})}(q) = 1$. In the latter case, for every $q \in \mathbb{P}\langle s,p,x\rangle$, we have $M_{\Omega(\boldsymbol{\Sigma})}(q) = 0$.
- If $w \in s^\bullet \setminus {}^\bullet s$ then, due to the cleanness of Σ_w, either $M^{ctr}_{\Sigma_w} = {}^\circ\Sigma_w$, or there is $e \in {}^\circ\Sigma_w$ such that $M_{\Sigma_w}(e) = 0$. In the former case, for every $q \in \mathbb{P}\langle s,p\rangle$, we have $M_{\Omega(\boldsymbol{\Sigma})}(q) = 1$. In the latter case, for every $q \in \mathbb{P}\langle s,p,e\rangle$, we have $M_{\Omega(\boldsymbol{\Sigma})}(q) = 0$.
- If $w \in s^\bullet \cap {}^\bullet s$ then, due to the cleanness of Σ_w, either $M^{ctr}_{\Sigma_w} \in \{{}^\circ\Sigma_w, \Sigma_w^\circ\}$, or there are $e \in {}^\circ\Sigma_w$ and $x \in \Sigma_w^\circ$ such that $M_{\Sigma_w}(e) = M_{\Sigma_w}(x) = 0$. In the former case, for every $q \in \mathbb{P}\langle s,p\rangle$, we have $M_{\Omega(\boldsymbol{\Sigma})}(q) = 1$. In the latter case, for every $q \in \mathbb{P}\langle s,p,e,x\rangle$, we have $M_{\Omega(\boldsymbol{\Sigma})}(q) = 0$.

Case 2: $v = w$. Since Σ_w is an ac-free labelled net, there is a control place p of Σ_w such that $M_{\Sigma_w}(p) < 2 \cdot W_{\Sigma_w}(p,t)$; in particular, this means that $1 \leq W_{\Sigma_w}(p,t)$ and so $p \in {}^\bullet t$. Besides, if Σ_w is quasi-safe, we can choose p such that $M_{\Sigma_w}(p) \leq 1$.

If p is internal, then p is also an internal place of $\Omega(\boldsymbol{\Sigma})$, $p \in {}^\bullet(w \lhd t)$ and

$$M_{\Omega(\boldsymbol{\Sigma})}(p) = M_{\Sigma_w}(p) < 2 \cdot W_{\Sigma_w}(p,t) = 2 \cdot W_{\Omega(\boldsymbol{\Sigma})}(p, w \lhd t)\ .$$

Moreover, in the quasi-safe case we have $M_{\Omega(\boldsymbol{\Sigma})}(p) = M_{\Sigma_v}(p) \leq 1$.

If $p \in {}^\circ\Sigma_w$ (the case $p \in \Sigma_w^\circ$ is symmetric), let s be the place in Ω such that $\{s\} = {}^\bullet w$. We then observe that the following hold:

- If $w^\bullet \neq \{s\}$ then, for every $q \in \mathbb{P}\langle s,p\rangle$ we have

  $$M_{\Omega(\boldsymbol{\Sigma})}(q) = M_{\Sigma_w}(p) < 2 \cdot W_{\Sigma_w}(p,t) = 2 \cdot W_{\Omega(\boldsymbol{\Sigma})}(q, w \lhd t)\ .$$

  Moreover, $M_{\Omega(\boldsymbol{\Sigma})}(q) = M_{\Sigma_w}(p) \leq 1$ in the quasi-safe case.

- If ${}^\bullet w = \{s\} = w^\bullet$ then, due to the cleanness of Σ_w, either $M^{ctr}_{\Sigma_w} = \Sigma^\circ_w$, or there is $x \in \Sigma^\circ_w$ such that $M_{\Sigma_w}(x) = 0$. In the former case, for every $q \in \mathbb{P}\langle s, p\rangle$ we have $M_{\Omega(\boldsymbol{\Sigma})}(q) = 1$. In the latter case, for every $q \in \mathbb{P}\langle s, p, x\rangle$, we have

$$M_{\Omega(\boldsymbol{\Sigma})}(q) = M_{\Sigma_w}(p) < 2 \cdot W_{\Sigma_w}(p,t) \leq 2 \cdot W_{\Omega(\boldsymbol{\Sigma})}(q, w \lhd t)\ .$$

  Moreover, $M_{\Omega(\boldsymbol{\Sigma})}(q) = M_{\Sigma_w}(p) \leq 1$ in the quasi-safe case.

□

Proposition A.5. Let Ω be a sequential operator box, and $\boldsymbol{\Sigma}$ be an Ω-tuple of static boxes.

1. If $v \in T_\Omega$ is such that ${}^\circ\Omega = {}^\bullet v$ or ${}^\circ\Omega = v^\bullet$, then $\overline{\Omega(\boldsymbol{\Sigma})} = \Omega(\boldsymbol{\Sigma}')$, where $\boldsymbol{\Sigma}'$ is $\boldsymbol{\Sigma}$ with Σ_v replaced respectively by $\overline{\Sigma_v}$ or $\underline{\Sigma_v}$.
2. If $v \in T_\Omega$ is such that $\Omega^\circ = {}^\bullet v$ or $\Omega^\circ = v^\bullet$, then $\underline{\Omega(\boldsymbol{\Sigma})} = \Omega(\boldsymbol{\Sigma}')$, where $\boldsymbol{\Sigma}'$ is $\boldsymbol{\Sigma}$ with Σ_v replaced respectively by $\overline{\Sigma_v}$ or $\underline{\Sigma_v}$.

Proof:
Follows from the definitions. □

A.2. Structural equivalence

For a general operator box Ω the six auxiliary relations $\equiv^i_\Omega$ and the relation $\equiv_\Omega$ are defined as in section 5.1 for the ABC operator boxes.

Proposition A.6. Let Ω be a general operator box, and $\boldsymbol{\Sigma} \in dom_\Omega$.

1. If $\boldsymbol{\Sigma} \equiv_\Omega \boldsymbol{\Theta}$, then $\boldsymbol{\Theta} \in dom_\Omega$ and $\lceil \boldsymbol{\Sigma} \rceil = \lceil \boldsymbol{\Theta} \rceil$. Moreover, if all the boxes in $\boldsymbol{\Sigma}$ are quasi-safe then so are all the boxes in $\boldsymbol{\Theta}$.
2. $\equiv_\Omega$ is an equivalence relation on dom_Ω.
3. If $\boldsymbol{\Theta} \in dom_\Omega$ and $\lceil \boldsymbol{\Sigma} \rceil = \lceil \boldsymbol{\Theta} \rceil$, then $\Omega(\boldsymbol{\Sigma}) = \Omega(\boldsymbol{\Theta})$ *iff* $\boldsymbol{\Sigma} \equiv_\Omega \boldsymbol{\Theta}$.

Proof:
(1) The result is obvious for all unary operator boxes. For the parallel operator box, it follows from the observation that $\equiv_{\Omega_\parallel} = \equiv^5_{\Omega_\parallel}$ only modifies the marking of the open buffer places, and consequently preserves the property of being a static or dynamic component (see also proposition 4.3(5,6)). For a sequential operator box Ω, it suffices to observe that the relations $\equiv^i_\Omega$ only act on the markings of the components and preserve the number of static and dynamic boxes in $\boldsymbol{\Sigma}$. Indeed, in the case of $\equiv^5_\Omega$ this follows from proposition 4.3(5,6), and in the remaining cases can be shown by case analysis. Take, for instance, the relation $\equiv^1_\Omega$ with $(\Sigma_v, \Sigma_w) = (\overline{\Psi_v}, \Psi_w)$ and $(\Theta_v, \Theta_w) = (\Psi_v, \overline{\Psi_w})$. Then, by the definition of dom_Ω, $(\Sigma_v, \Sigma_w) \in \mathsf{abox}^{dyn} \times \mathsf{abox}^{stc}$. Thus, by proposition 4.3(7), we have $\Psi_v \in \mathsf{abox}^{stc}$, and so $(\Theta_v, \Theta_w) \in \mathsf{abox}^{stc} \times \mathsf{abox}^{dyn}$. Hence $\boldsymbol{\Theta} \in dom_\Omega$.

The second part follows from the definition of a quasi-safe box, and the observation that the boxes in $\boldsymbol{\Theta}$ have clean markings (since $\boldsymbol{\Theta} \in dom_\Omega$).

(2) The result is again obvious for all unary operator boxes. For the parallel operator box, it is easy to check that $\equiv_{\Omega_{\|}} = \equiv^5_{\Omega_{\|}}$ is reflexive, symmetric and transitive.

Next let us consider any sequential operator Ω. That $\equiv_\Omega$ is reflexive follows from $id_{\mathsf{abox}} \subseteq (\equiv^5_\Omega \circ \equiv^4_\Omega)$. Moreover, it is easy to see that each $\equiv^i_\Omega$ is symmetric and $(\equiv^5_\Omega \circ \equiv^i_\Omega) = (\equiv^i_\Omega \circ \equiv^5_\Omega)$, for $0 \leq i \leq 4$. Hence $\equiv_\Omega$ is also symmetric. That it is also transitive on dom_Ω results from the following two facts holding for all $0 \leq j, k \leq 4$, where in the second case, the domains of the relations are restricted to dom_Ω:

$$\begin{array}{rcl} (\equiv^5_\Omega \circ \equiv^j_\Omega \circ \equiv^5_\Omega \circ \equiv^k_\Omega) & = & (\equiv^5_\Omega \circ \equiv^5_\Omega \circ \equiv^j_\Omega \circ \equiv^k_\Omega) \;=\; (\equiv^5_\Omega \circ \equiv^j_\Omega \circ \equiv^k_\Omega) \\ (\equiv^j_\Omega \circ \equiv^k_\Omega) & \subseteq & \bigcup_{i=0}^{4} \equiv^i_\Omega \,. \end{array}$$

The former fact is obvious, while the latter can be shown by case analysis (using, in particular, the property that in any $\boldsymbol{\Sigma} \in dom_\Omega$, each transition in Ω has only one pre-place and one post-place, and at most one Σ_v is a dynamic box). For example, one can easily check that:

$$\begin{array}{lcllcllcl} (\equiv^1_\Omega \circ \equiv^0_\Omega) & \subseteq & \equiv^3_\Omega & (\equiv^1_\Omega \circ \equiv^1_\Omega) & \subseteq & (\equiv^1_\Omega \cup \equiv^4_\Omega) & (\equiv^1_\Omega \circ \equiv^2_\Omega) & = & \varnothing \\ (\equiv^1_\Omega \circ \equiv^3_\Omega) & \subseteq & \equiv^3_\Omega & (\equiv^1_\Omega \circ \equiv^4_\Omega) & = & \equiv^1_\Omega \,. & & & \end{array}$$

(3) Since $\lceil \boldsymbol{\Sigma} \rceil = \lceil \boldsymbol{\Theta} \rceil$, $\boldsymbol{\Sigma}$ and $\boldsymbol{\Theta}$ may only differ in their markings. For a communication interface operator box, the property is obvious, since it only modifies the transitions (and the corresponding arcs), in a way that only depends on the transition labels. For a buffer restriction operator box, the property is also obvious since it only modifies the status of the same place in Σ and Θ, and adds the same empty place to both of them. For the parallel operator box, the property results from the fact that the marking of the non-open buffer places is unchanged, while the marking of the open buffer places is modified in a way that is exactly preserved by $\equiv_{\Omega_{\|}} = \equiv^5_{\Omega_{\|}}$.

Finally, we consider the case of a sequential operator box Ω. Then $\lceil \Omega(\boldsymbol{\Sigma}) \rceil = \lceil \Omega(\boldsymbol{\Theta}) \rceil$ since $\lceil \boldsymbol{\Sigma} \rceil = \lceil \boldsymbol{\Theta} \rceil$. Moreover, by the definition of net substitution,

$$M^{opb}_{\Omega(\boldsymbol{\Sigma})} = M^{opb}_{\Omega(\boldsymbol{\Theta})} \quad \Longleftrightarrow \quad \sum_{v \in T_\Omega} M^{opb}_{\Sigma_v} = \sum_{v \in T_\Omega} M^{opb}_{\Theta_v} \,,$$

where we identify the marking of the open buffer places with the multiset of buffer symbols it represents. This just corresponds to the property preserved by $\equiv^5_\Omega$. Now, the closed buffer places (with their marking) are the same in $\boldsymbol{\Sigma}$ and $\Omega(\boldsymbol{\Sigma})$, as well as in $\boldsymbol{\Theta}$ and $\Omega(\boldsymbol{\Theta})$, and they are not influenced by $\equiv_\Omega$, so that we only need to consider the control places.

We first consider the ($\Longleftarrow$) direction of the property.
If all the boxes in $\boldsymbol{\Sigma}$ are static, then $\boldsymbol{\Sigma} \equiv^5_\Omega \boldsymbol{\Theta}$ and we clearly have $\Omega(\boldsymbol{\Sigma}) = \Omega(\boldsymbol{\Theta})$. Next we prove that for $0 \leq i \leq 4$, if $\boldsymbol{\Sigma} \equiv^i_\Omega \boldsymbol{\Theta}$ then we have also $M^{ctr}_{\Omega(\boldsymbol{\Sigma})} = M^{ctr}_{\Omega(\boldsymbol{\Theta})}$. We consider the following three cases (the proof for $i \in \{2, 3\}$ is similar to that for $i = 1$):

- $\boldsymbol{\Sigma} \equiv^0_\Omega \boldsymbol{\Theta}$. Suppose that v and Ψ are as in the definition of $\equiv^0_\Omega$. Without loss of generality, we may assume that $\Sigma_v = \overline{\Psi}$ and $\Theta_v = \underline{\Psi}$. Then by the cleanness of Σ_v and Θ_v, we have $M_{\Sigma_v} = {}^\circ\Sigma_v$ and $M_{\Theta_v} = \Theta_v^\circ$. Hence, by lemma A.3(2), $M^{ctr}_{\Omega(\boldsymbol{\Sigma})} = \mathbb{P}\langle s \rangle = M^{ctr}_{\Omega(\boldsymbol{\Theta})}$, where $\{s\} = {}^\bullet v = v^\bullet$.

- $\boldsymbol{\Sigma} \equiv^1_\Omega \boldsymbol{\Theta}$. Suppose that v, w and Ψ_v, Ψ_w are as in the definition of $\equiv^1_\Omega$. Without loss of generality, we may assume that $\Sigma_v = \overline{\Psi}$ and $\Theta_w = \overline{\Psi}$. Then by the cleanness of Σ_v and Θ_w, we have $M_{\Sigma_v} = {}^\circ\Sigma_v$ and $M_{\Theta_w} = {}^\circ\Theta_w$. Hence, by lemma A.3(2), $M^{ctr}_{\Omega(\boldsymbol{\Sigma})} = \mathbb{P}\langle s\rangle = M^{ctr}_{\Omega(\boldsymbol{\Theta})}$, where $\{s\} = {}^\bullet v = {}^\bullet w$.

- $\boldsymbol{\Sigma} \equiv^4_\Omega \boldsymbol{\Theta}$. Then $\boldsymbol{\Sigma} = \boldsymbol{\Theta}$ and we are done.

We now consider the ($\Longrightarrow$) direction of the property.
If $\boldsymbol{\Sigma}$ is a tuple of static boxes, then $M^{ctr}_{\Omega(\boldsymbol{\Sigma})} = \varnothing = M^{ctr}_{\Omega(\boldsymbol{\Theta})}$, and so $\boldsymbol{\Theta}$ is also a tuple of static boxes, since if there is a marked i-labelled place p in some Θ_v then p would be also a marked i-labelled place in $\Omega(\boldsymbol{\Theta}) = \Omega(\boldsymbol{\Sigma})$. Similarly, if there is a marked entry place e in some Θ_v with ${}^\bullet v = \{s\}$ then each place $q \in \mathbb{P}\langle s, e\rangle$ would be also marked in $\Omega(\boldsymbol{\Theta}) = \Omega(\boldsymbol{\Sigma})$ (the argument is similar if there is a marked exit place in some Θ_v). Hence, in this case, $\boldsymbol{\Sigma} \equiv^5_\Omega \boldsymbol{\Theta}$, and so $\boldsymbol{\Sigma} \equiv_\Omega \boldsymbol{\Theta}$. Let us now assume that there is one dynamic box in $\boldsymbol{\Sigma}$ corresponding to some $v \in T_\Omega$, with ${}^\bullet v = \{s\}$ and $v^\bullet = \{s'\}$.

For every i-labelled place p in every Σ_z we have $p \in \Omega(\boldsymbol{\Sigma}) = \Omega(\boldsymbol{\Theta})$ and $M_{\Sigma_z}(p) = M_{\Omega(\boldsymbol{\Sigma})}(p) = M_{\Omega(\boldsymbol{\Theta})}(p) = M_{\Theta_z}(p)$.

For the places in any $\mathbb{P}\langle s''\rangle$, where $s'' \in S_\Omega$, *i.e.*, those constructed from the entry/exit places of $\boldsymbol{\Sigma}$ and $\boldsymbol{\Theta}$, we distinguish two cases:

Case 1: $M^{ctr}_{\Sigma_v} \notin \{{}^\circ\Sigma_v, \Sigma_v^\circ\}$. By the cleanness of Σ_v, there is $e \in {}^\circ\Sigma_v$ and $x \in \Sigma_v^\circ$ with $M_{\Sigma_v}(e) = 0 = M_{\Sigma_v}(x)$. Hence, for every place s'' in Ω and every place q belonging to

$$\begin{array}{llll} \mathbb{P}\langle s''\rangle & \text{if } s'' \notin \{s, s'\}\ , & \mathbb{P}\langle s'', e\rangle & \text{if } s'' = s \neq s'\ , \\ \mathbb{P}\langle s'', x\rangle & \text{if } s'' = s' \neq s\ , & \mathbb{P}\langle s'', e, x\rangle & \text{if } s'' = s = s'\ , \end{array}$$

we have that $M_{\Omega(\boldsymbol{\Sigma})}(q) = 0 = M_{\Omega(\boldsymbol{\Theta})}(q)$, and all the places in $\boldsymbol{\Theta}$ used to construct q have the empty marking. In particular, this applies to e, x and all the entry/exit places of Θ_w for $w \neq v$. As a consequence, for every entry place $e' \neq e$ in Θ_v, if we replace e by e' in either a place $q \in \mathbb{P}\langle s, e\rangle$ or $q \in \mathbb{P}\langle s, e, x\rangle$ above, we will have $M_{\Omega(\boldsymbol{\Sigma})}(q) = M_{\Sigma_v}(e') = M_{\Omega(\boldsymbol{\Theta})}(q) = M_{\Theta_v}(e')$, and similarly for any exit place $x' \neq x$ in Θ_v. Hence, $\boldsymbol{\Sigma} \equiv^5_\Omega \boldsymbol{\Theta}$.

Case 2: $M^{ctr}_{\Sigma_v} = {}^\circ\Sigma_v$ (the case $M^{ctr}_{\Sigma_v} = \Sigma_v^\circ$ is similar). Now, by lemma A.2(2), for every $q \in \mathbb{P}\langle s\rangle$, $M_{\Omega(\boldsymbol{\Sigma})}(q) = 1 = M_{\Omega(\boldsymbol{\Theta})}(q)$. Let us choose one of those q's: exactly one of the entry/exit places used to construct q must be marked with one token in $\boldsymbol{\Theta}$, while the other ones have the empty marking. Let us assume that this is due to an exit place $x \in \Theta_w^\circ$, so that $M_{\Theta_w}(x) = 1$ and $w^\bullet = \{s\}$ (the case where the token comes from an entry place e in some ${}^\circ\Theta_w$ with ${}^\bullet w = \{s\}$ is similar). If we replace the x in q by any other exit place x' of Θ_w, we will obtain another place in $\mathbb{P}\langle s\rangle$, with a marking 1 which may only come from x'. Thus, by the cleanness of Θ_w, $M^{ctr}_{\Theta_w} = \Theta_w^\circ$. Moreover, Θ_x must be a static box, for every $x \in T_\Omega \setminus \{w\}$. So we conclude that if $v = w$, then $\boldsymbol{\Sigma}\ (\equiv^5_\Omega \circ \equiv^0_\Omega)\ \boldsymbol{\Theta}$, and if $v \neq w$, then $\boldsymbol{\Sigma}\ (\equiv^5_\Omega \circ \equiv^3_\Omega)\ \boldsymbol{\Theta}$. □

Proposition A.7. Let Ω be an n-ary sequential operator box and $\boldsymbol{\Sigma} \in dom_\Omega$.

1. If $\Sigma_i\ [U_i\rangle\ \Theta_i$ (for $i \leq n$), then $\boldsymbol{\Theta} \in dom_\Omega$, and

$$\Omega(\boldsymbol{\Sigma})\ [(v_1 \blacktriangleleft U_1) \cup \ldots \cup (v_n \blacktriangleleft U_n)\rangle\ \Omega(\boldsymbol{\Theta})\ .$$

2. If $\Omega(\boldsymbol{\Sigma})\,[U\rangle\,\Theta$, then there are $\boldsymbol{\Psi}, \boldsymbol{\Phi} \in dom_\Omega$ and steps $U_1, \ldots, U_n$ such that $\boldsymbol{\Sigma} \equiv_\Omega \boldsymbol{\Psi}$, $\Psi_i\,[U_i\rangle\,\Phi_i$ (for $i \leq n$), $\Theta = \Omega(\boldsymbol{\Phi})$ and

$$U = (v_1 \blacktriangleleft U_1) \cup \ldots \cup (v_n \blacktriangleleft U_n)\ .$$

Note I: As a consequence, $\mathsf{enabled}(\Omega(\boldsymbol{\Sigma}))$ comprises exactly all sets $(v_1 \blacktriangleleft U_1) \cup \ldots \cup (v_n \blacktriangleleft U_n)$ of transitions such that there is $\boldsymbol{\Psi} \equiv_\Omega \boldsymbol{\Sigma}$ and $U_i \in \mathsf{enabled}(\Psi_i)$ (for $i \leq n$).
Note II: For the sequential operators, at most one U_i is non-empty.

Proof:
Below, for any box Φ and $b \in \mathbb{B}$, b will represent the only b-labelled open buffer place of Φ.

(1) Since $\Sigma_i\,[U_i\rangle\,\Theta_i$ (for $i \leq n$), each Θ_i is a static (resp. dynamic) box *iff* so is Σ_i. Hence $\boldsymbol{\Theta} \in dom_\Omega$ as $\boldsymbol{\Sigma} \in dom_\Omega$. If $\boldsymbol{\Sigma}$ are static boxes, then each U_j is empty, $\boldsymbol{\Sigma} = \boldsymbol{\Theta}$ and $\Omega(\boldsymbol{\Sigma})\,[\varnothing\rangle\,\Omega(\boldsymbol{\Theta})$. Otherwise, $\boldsymbol{\Sigma}$ has exactly one dynamic component Σ_i. We then observe that, for all $j \neq i$, $U_j = \varnothing$ and $\Theta_j = \Sigma_j$. Moreover,

- For every internal or closed buffer place p in Σ_i, we have:

$$\begin{array}{rcccl} M_{\Omega(\boldsymbol{\Sigma})}(p) = M_{\Sigma_i}(p) & \geq & \sum_{t \in U_i} W_{\Sigma_i}(p,t) & = & \sum_{t \in U_i} W_{\Omega(\boldsymbol{\Sigma})}(p, v_i \lhd t) \quad \text{and} \\ & & \sum_{t \in U_i} W_{\Sigma_i}(t,p) & = & \sum_{t \in U_i} W_{\Omega(\boldsymbol{\Sigma})}(v_i \lhd t, p)\ . \end{array}$$

- For every $b \in \mathbb{B}$, we have:

$$\begin{array}{rcccl} M_{\Omega(\boldsymbol{\Sigma})}(b) \geq M_{\Sigma_i}(b) & \geq & \sum_{t \in U_i} W_{\Sigma_i}(b,t) & = & \sum_{t \in U_i} W_{\Omega(\boldsymbol{\Sigma})}(b, v_i \lhd t) \quad \text{and} \\ & & \sum_{t \in U_i} W_{\Sigma_i}(t,b) & = & \sum_{t \in U_i} W_{\Omega(\boldsymbol{\Sigma})}(v_i \lhd t, b)\ . \end{array}$$

- For every place $s \in S_\Omega$ and every place $q = (e_1, \ldots, e_k, x_1, \ldots, x_m) \in \mathbb{P}\langle s \rangle$, taking $P \stackrel{\text{df}}{=} \{e_1, \ldots, e_k, x_1, \ldots, x_m\} \cap S_{\Sigma_i}$, we have:

$$\begin{array}{rcccl} M_{\Omega(\boldsymbol{\Sigma})}(q) = \sum_{p \in P} M_{\Sigma_i}(p) & \geq & \sum_{p \in P} \sum_{t \in U_i} W_{\Sigma_i}(p,t) & = & \sum_{t \in U_i} W_{\Omega(\boldsymbol{\Sigma})}(q, v_i \lhd t) \quad \text{and} \\ & & \sum_{p \in P} \sum_{t \in U_i} W_{\Sigma_i}(t,p) & = & \sum_{t \in U_i} W_{\Omega(\boldsymbol{\Sigma})}(v_i \lhd t, q)\ . \end{array}$$

Hence $\Omega(\boldsymbol{\Sigma})\,[(v_1 \blacktriangleleft U_1) \cup \ldots \cup (v_n \blacktriangleleft U_n)\rangle\,\Omega(\boldsymbol{\Theta})$.

(2) If $\Omega(\boldsymbol{\Sigma})\,[U\rangle\,\Theta$, then there exists n steps $U_1, \ldots, U_n$ such that $U = (v_1 \blacktriangleleft U_1) \cup \ldots \cup (v_n \blacktriangleleft U_n)$. If $U = \varnothing$, then each U_i is empty too, and the property obviously holds, with $\boldsymbol{\Psi} = \boldsymbol{\Phi} = \boldsymbol{\Sigma}$. This will be the case, in particular, if $\boldsymbol{\Sigma}$ are static boxes. Otherwise, we will assume that Σ_i is the only dynamic box in $\boldsymbol{\Sigma}$ and consider two cases:

Case 1: $M_{\Sigma_i}^{ctr} \notin \{{}^\circ\Sigma_i, \Sigma_i^\circ\}$. By the cleanness of Σ_i, there is $e \in {}^\circ\Sigma_i$ and $x \in \Sigma_i^\circ$ with $M_{\Sigma_i}(e) = 0 = M_{\Sigma_i}(x)$. For all $j \neq i$ and $t \in T_{\Sigma_j}$, we observe that the following hold:

- If q is an internal place in ${}^\bullet t$, we have that $q \in {}^\bullet(v_j \lhd t)$ and $M_{\Omega(\boldsymbol{\Sigma})}(q) = M_{\Sigma_i}(q) = 0$.
- If p is an entry place in ${}^\bullet t$ and ${}^\bullet v_j = \{s\}$ (or p is an exit place in ${}^\bullet t$ and $v_j^\bullet = \{s\}$), for every

$$\begin{array}{llll} q \in \mathbb{P}\langle s,p\rangle & \text{if } {}^\bullet v_i \neq \{s\} \neq v_i^\bullet\ , & q \in \mathbb{P}\langle s,p,e\rangle & \text{if } {}^\bullet v_i = \{s\} \neq v_i^\bullet\ , \\ q \in \mathbb{P}\langle s,p,x\rangle & \text{if } {}^\bullet v_i \neq \{s\} = v_i^\bullet\ , & q \in \mathbb{P}\langle s,p,e,x\rangle & \text{if } {}^\bullet v_i = \{s\} = v_i^\bullet\ , \end{array}$$

  we have that $q \in {}^\bullet(v_j \lhd t)$ and $M_{\Omega(\boldsymbol{\Sigma})}(q) = 0$.

Hence $U_j = \varnothing$, for all $j \neq i$.

Let Ψ_i be Σ_i modified as follows: for each $b \in \mathbb{B}$, $M_{\Psi_i}(b) = M_{\Omega(\boldsymbol{\Sigma})}(b)$; moreover, for all $j \neq i$, let Ψ_j be Σ_j with the marking of all the open buffer places set to 0. We then have $\boldsymbol{\Sigma} \equiv_\Omega \boldsymbol{\Psi}$, and the following hold:

- For every internal or closed buffer place p in S_{Ψ_i}, we have:

$$M_{\Psi_i}(p) = M_{\Sigma_i}(p) = M_{\Omega(\boldsymbol{\Sigma})}(p) \geq \sum_{t \in U_i} W_{\Omega(\boldsymbol{\Sigma})}(p, v_i \lhd t) = \sum_{t \in U_i} W_{\Psi_i}(p,t)\ .$$

- For every $b \in \mathbb{B}$, we have:

$$M_{\Psi_i}(b) = M_{\Omega(\boldsymbol{\Sigma})}(b) \geq \sum_{t \in U_i} W_{\Omega(\boldsymbol{\Sigma})}(b, v_i \lhd t) = \sum_{t \in U_i} W_{\Psi_i}(b,t)\ .$$

- For every entry place p in S_{Ψ_i} (the situation is symmetric if p is an exit place), for every $q \in \mathbb{P}\langle s,p\rangle$ if ${}^\bullet v_i = \{s\} \neq v_i^\bullet$, we have:

$$M_{\Psi_i}(p) = M_{\Sigma_i}(p) = M_{\Omega(\boldsymbol{\Sigma})}(q) \geq \sum_{t \in U_i} W_{\Omega(\boldsymbol{\Sigma})}(q, v_i \lhd t) = \sum_{t \in U_i} W_{\Psi_i}(p,t)\ ;$$

  and for every $q \in \mathbb{P}\langle s,p,x\rangle$ if ${}^\bullet v_i = \{s\} = v_i^\bullet$, we have:

$$\begin{aligned} M_{\Psi_i}(p) = M_{\Sigma_i}(p) = M_{\Omega(\boldsymbol{\Sigma})}(q) &\geq \sum_{t \in U_i} W_{\Omega(\boldsymbol{\Sigma})}(q, v_i \lhd t) \\ &= \sum_{t \in U_i} \big(W_{\Psi_i}(p,t) + W_{\Psi_i}(x,t)\big) \geq \sum_{t \in U_i} W_{\Psi_i}(p,t)\ . \end{aligned}$$

Hence $\Psi_i\,[U_i\rangle\,\Phi_i$, and $\Psi_j\,[U_j\rangle\,\Phi_j$ for all $j \neq i$, so that from the first part of the proposition, we obtain that $\Theta = \Omega(\boldsymbol{\Phi})$.

Case 2: $M_{\Sigma_i}^{ctr} = {}^\circ\Sigma_i$ (the case $M_{\Sigma_i}^{ctr} = \Sigma_i^\circ$ is similar). Suppose that ${}^\bullet v_i = \{s\}$. Then, by lemma A.3(2), $M_{\Omega(\boldsymbol{\Sigma})}^{ctr} = \mathbb{P}\langle s\rangle$. Hence, for every $(v \lhd t) \in U$, we have the following: ${}^\bullet t$ does not contain any internal places; if ${}^\bullet t \cap {}^\circ\Sigma_v \neq \varnothing$ then ${}^\bullet v = \{s\}$; and if ${}^\bullet t \cap \Sigma_v^\circ \neq \varnothing$ then $v^\bullet = \{s\}$.

Suppose now that $v \lhd t, v' \lhd t' \in U$, where $v \neq v'$. Then assume that there is $e \in {}^\bullet t \cap {}^\circ\Sigma_v$ and $e' \in {}^\bullet t' \cap {}^\circ\Sigma_{v'}$. We have, for every $q \in \mathbb{P}\langle s,e,e'\rangle$, that

$$1 = M_{\Omega(\boldsymbol{\Sigma})}(q) \geq W_{\Omega(\boldsymbol{\Sigma})}(q, v \lhd t) + W_{\Omega(\boldsymbol{\Sigma})}(q, v' \lhd t') = W_{\Sigma_v}(e,t) + W_{\Sigma_v'}(e',t') \geq 2\ .$$

Hence we have obtained a contradiction (the situation is similar if e is replaced by an $x \in {}^\bullet t \cap \Sigma_v^\circ$ and/or e' is replaced by an $x' \in {}^\bullet t' \cap \Sigma_{v'}^\circ$). As a consequence, there is a unique non-empty U_j. We then consider two cases:

- $j = i$. We may first observe that it is not possible to have $t, t' \in U_i$ (we do not exclude here the case where $t = t'$) with $e \in {}^\bullet t \cap {}^\circ\Sigma_i$ and $x \in {}^\bullet t' \cap \Sigma_i^\circ$ because either $v_i^\bullet = \{s'\} \neq \{s\}$ and for $q \in \mathbb{P}\langle s', x\rangle$ we have

$$M_{\Omega(\boldsymbol{\Sigma})}(q) = 0 < 1 \leq W_{\Sigma_i}(x, t') = W_{\Omega(\boldsymbol{\Sigma})}(q, v_i \lhd t') \ ,$$

or $v_i^\bullet = \{s\}$ and, for $q \in \mathbb{P}\langle s, e, x\rangle$, we have

$$M_{\Omega(\boldsymbol{\Sigma})}(q) = 1 < 2 \leq W_{\Sigma_i}(e, t) + W_{\Sigma_i}(x, t') = W_{\Omega(\boldsymbol{\Sigma})}(q, v_i \lhd t) + W_{\Omega(\boldsymbol{\Sigma})}(q, v_i \lhd t') \ .$$

Hence the control pre-places of the transitions in U_i are either all in ${}^\circ\Sigma_i$ or all in Σ_i°. In the first case, we may not have $t \neq t' \in U_i$ with some $e \in {}^\bullet t \cap {}^\bullet t' \cap {}^\circ\Sigma_i$ since then, for every $q \in \mathbb{P}\langle s, e\rangle$, we would have had:

$$M_{\Omega(\boldsymbol{\Sigma})}(q) = 1 < 2 \leq W_{\Sigma_i}(e, t) + W_{\Sigma_i}(e, t') = W_{\Omega(\boldsymbol{\Sigma})}(q, v_i \lhd t) + W_{\Omega(\boldsymbol{\Sigma})}(q, v_i \lhd t') \ .$$

Now, for all $e \in {}^\circ\Sigma_i$ and $q \in \mathbb{P}\langle s, e\rangle$, we have:

$$M_{\Sigma_i}(e) = M_{\Omega(\boldsymbol{\Sigma})}(q) \geq \sum_{t \in U_i} W_{\Omega(\boldsymbol{\Sigma})}(q, v_i \lhd t) = \sum_{t \in U_i} W_{\Sigma_i}(e, t) \ .$$

Moreover, for every closed buffer place $p \in S_{\Sigma_i}$, we have:

$$M_{\Sigma_i}(p) = M_{\Omega(\boldsymbol{\Sigma})}(p) \geq \sum_{t \in U_i} W_{\Omega(\boldsymbol{\Sigma})}(p, v_i \lhd t) = \sum_{t \in U_i} W_{\Sigma_i}(p, t) \ .$$

Define Ψ_i to be Σ_i modified as follows: for each $b \in \mathbb{B}$, $M_{\Psi_i}(b) = M_{\Omega(\boldsymbol{\Sigma})}(b)$; moreover, for all $m \neq i$, let Ψ_m be Σ_m with the marking of all its open buffer places set to 0. We then have, for every $b \in \mathbb{B}$,

$$M_{\Psi_i}(b) = M_{\Omega(\boldsymbol{\Sigma})}(b) \geq \sum_{t \in U_i} W_{\Omega(\boldsymbol{\Sigma})}(b, v_i \lhd t) = \sum_{t \in U_i} W_{\Psi_i}(b, t) \ .$$

Hence $\Psi_i\,[U_i\rangle\,\Phi_i$ and $\Psi_m\,[U_m\rangle\,\Phi_m$, for all $m \neq i$, with $\boldsymbol{\Psi} \equiv_\Omega \boldsymbol{\Sigma}$, and so from the first part of the proposition, $\Theta = \Omega(\boldsymbol{\Phi})$.

In the second case, *i.e.*, ${}^\bullet U_i \subseteq \Sigma_i^\circ$, we must have ${}^\bullet v_i = \{s\} = v_i^\bullet$ and we may replace in $\boldsymbol{\Sigma}$ the tokens in the entry places of Σ_i by one token in each exit place of Σ_i, leading to an Ω-equivalent tuple of arguments (see the definition of $\equiv_\Omega^0$), to which we may apply the previous analysis, when $M_{\Sigma_i}^{ctr} = \Sigma_i^\circ$.

- $j \neq i$. Then either ${}^\bullet v_j = \{s\}$ and we replace $\boldsymbol{\Sigma}$ by $\boldsymbol{\Psi}$ where the only modifications consist in replacing $M_{\Psi_i}^{ctr}$ by the empty marking and $M_{\Psi_j}^{ctr}$ by ${}^\circ\Psi_j$ (see the definition of $\equiv_\Omega^1$), or $v_j^\bullet = \{s\}$ and we replace $\boldsymbol{\Sigma}$ by $\boldsymbol{\Psi}$ where the only modifications consist in replacing $M_{\Psi_i}^{ctr}$ by the empty marking and $M_{\Psi_j}^{ctr}$ by Ψ_j° (see the definition of $\equiv_\Omega^3$), and we have the same situation as before with $\boldsymbol{\Sigma} \equiv_\Omega \boldsymbol{\Psi}$. □

Proposition A.8. Let Σ be a static or dynamic box, and Ω_φ be a communication interface operator box.

1. If $\Sigma\,[U_1 \uplus \ldots \uplus U_k\rangle\,\Theta$ and each $\lambda_\Sigma(U_i)$ belongs to the domain of φ, then

$$\Omega_\varphi(\Sigma)\,[\{v^\varphi \lhd U_1, \ldots, v^\varphi \lhd U_k\}\rangle\,\Omega_\varphi(\Theta)\ .$$

2. If $\Omega_\varphi(\Sigma)\,[U\rangle\,\Psi$ then there is a box Θ and steps of transitions $U_1, \ldots, U_k$ such that $\Psi = \Omega_\varphi(\Theta)$, $U = \{v^\varphi \lhd U_1, \ldots, v^\varphi \lhd U_k\}$, and $\Sigma\,[U_1 \uplus \ldots \uplus U_k\rangle\,\Theta$.

Note: As a consequence, $\mathsf{enabled}(\Omega_\varphi(\Sigma))$ comprises exactly all $U = \{v^\varphi \lhd U_1, \ldots, v^\varphi \lhd U_k\}$ such that $U_1 \uplus \ldots \uplus U_k \in \mathsf{enabled}(\Sigma)$ and each $\lambda_\Sigma(U_i)$ belongs to the domain of φ.

Proof:
It can be simply checked applying the definitions. □