Asynchronous Box Calculus
Raymond Devillers, Hanna Klaudel, Maciej Koutny, Franck Pommereau

To cite this version:
Raymond Devillers, Hanna Klaudel, Maciej Koutny, Franck Pommereau. Asynchronous Box Calculus. Fundamenta Informaticae, 2003, 54, pp.1-50. hal-00114675

HAL Id: hal-00114675
https://hal.science/hal-00114675
Submitted on 17 Nov 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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*T
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 $E_1 \oplus E_2$ (either $E_1$ or $E_2$ can be executed); parallel composition $E_1 \parallel E_2$ ($E_1$ and $E_2$ can be executed concurrently); iteration $E_1 \otimes E_2$ ($E_1$ can be executed an arbitrary number of times, and then can be followed by $E_2$); and scoping $E \text{ sc } a$ (all handshake synchronisations involving pairs of $a$- and $\bar{a}$-labelled transitions are enforced, and after that the synchronising transitions may no longer be executed). We illustrate these constructs using an example
based on three process expressions:

\[
\begin{align*}
\text{CRITSECT} & \equiv (a_1; r_1) \Box (a_2; r_2) \otimes f \\
\text{USER}_1 & \equiv \alpha_1; r_1 \\
\text{USER}_2 & \equiv \alpha_2; r_2
\end{align*}
\]

modelling a critical section and two user processes. The atomic actions \(a_1\) and \(a_2\) (together with the matching actions \(\alpha_1\) and \(\alpha_2\)) model the granting of access to a shared resource, \(r_1\) and \(r_2\) (together with the matching \(r_1\) and \(r_2\)) model its release, and \(f\) models a final action. The system where these three processes operate in parallel is

\[
\text{PREMUX} \equiv \text{USER}_1 \parallel \text{CRITSECT} \parallel \text{USER}_2,
\]

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 \(a_1, \alpha_1,\) and \(\tau\) (similarly as in CCS, \(\tau\) represents an internal action).

The box expression PREMUX 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

\[
\text{MUX} \equiv \text{PREMUX} \text{ sc } a_1 \text{ sc } a_2 \text{ sc } r_1 \text{ sc } r_2,
\]

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 \(a_i. E \xrightarrow{a_i} E\) 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 PREMUX 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 MUX, i.e., that corresponding to the box expression MUX.
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

$$\text{USER}_1 \parallel \text{CRITSECT} \parallel \text{USER}_2 \equiv \text{USER}_1 \parallel \text{CRITSECT} \parallel \text{USER}_2 \equiv \text{USER}_1 \parallel \text{CRITSECT} \parallel \text{USER}_2,$$

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

$$((a_1; r_1) \sqcap (a_2; r_2)) \odot \mathcal{F} \frac{(f)}{\tau} ((a_1; r_1) \sqcap (a_2; r_2)) \odot f.$$

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):

- **PRODONE** \(\overset{\text{at}}{=} pr^+\)
- **CONSONE** \(\overset{\text{at}}{=} cr^-\)
- **TESTONE** \(\overset{\text{at}}{=} tr^\pm\).

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 $tr^\perp$ 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.

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 $\text{Syst} = \text{ProdOne} \parallel \text{ConsOne}$. 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 $\text{TestSyst} = \text{Syst} \parallel \text{TestOne}$ 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 $\text{Syst}$ (or $\text{TestSyst}$).

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 $\text{tie}$. 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 $\text{HideSyst} = \text{Syst} \text{tie} r$.

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.

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

![Figure 4. Asynchronous communication and buffer restriction.](image)
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 $X \equiv \cdots$ 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:

\[
\begin{align*}
\text{PROD} & \overset{df}{=} \text{PRODONE} \otimes f = pr^+ \otimes f \\
\text{CONS} & \overset{df}{=} \text{CONSONE} \otimes f = cr^- \otimes f \\
\text{TEST} & \overset{df}{=} \text{TESTONE} \otimes f = tr^\perp \otimes f ,
\end{align*}
\]

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 CONDONE 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 \cite{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:

\[
\text{SYSTI} \overset{df}{=} s;((\text{PROD} \parallel \text{PROD} \parallel \text{CONS}) \text{tie } r).
\]

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:

\[
\text{SYSTII} \overset{df}{=} (\text{PRODONE}; ((\text{PROD}\parallel\text{CONS}) \text{tie } r); \text{CONDONE}) \text{tie } r ,
\]

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:

\[
\text{SYSTIII} \overset{df}{=} \text{PROD} \parallel \text{CONS} \parallel \text{TEST} \parallel \text{TEST}.
\]

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 \( (i.e., b\text{-labelled}) \) 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 \( \text{SYSTI} \) rather than \( \text{SYST}_{I} \). 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.
Figure 5. Boxes for the three execution scenarios.

**Scenario I.** Consider the net $\Sigma_I$ 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:

$$\Sigma_I[\{t_1\} \{t_2, t_3\} \{t_2, t_4\} \{t_5, t_6, t_7\}] \Sigma'_I,$$

where $\Sigma'_I$ is $\Sigma_I$ 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

$$\Sigma_I[\{s\} \{p, p\} \{p, c\} \{f, f, f\}] \Sigma'_I.$$

Since each $x$-labelled place has exactly one token, we will say that $\Sigma'_I$ 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.
Scenario II. Consider the net $\Sigma_{II}$ 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:

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

where $\Sigma'_{II}$ is $\Sigma_{II}$ with one token in the internal buffer place and the $x$-labelled place, and no token elsewhere, so that $\Sigma'_{II}$ 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 $\Sigma_{II}$.

Scenario III. Consider the net $\Sigma_{III}$ 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} \[\{p\} \{p\} \{c, t, p\} \{t, t\}\] \Sigma'_{III},$$

where $\Sigma'_{III}$ is $\Sigma_{III}$ with two additional tokens in the buffer place. Notice that $\Sigma'_{III}$ 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 $\mu : X \rightarrow \mathbb{N}$, where $\mathbb{N} = \{0, 1, 2, \ldots\}$. We denote by $\text{mult}(X)$ the set of all finite multisets $\mu$ over $X$, i.e., those satisfying $\sum_{x \in X} \mu(x) < \infty$. We will write $\mu \leq \mu'$ if the domain $X$ of $\mu$ is included in that of the multiset $\mu'$, and $\mu(x) \leq \mu'(x)$, for all $x \in X$. An element $x \in X$ belongs to $\mu$, 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 $\mu$ 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 $\mu$ 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, $\mathbb{A}_r \doteq \mathbb{A} \cup \{\tau\}$ and for every $a \in \mathbb{A}$, $\hat{a}$ is an action in $\mathbb{A}$ such that $\hat{a} = a$. 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 $\varphi : \text{mult}(\mathbb{A}_r) \setminus \{\emptyset\} \rightarrow \mathbb{A}_r$. In particular, to introduce forced CCS-like synchronisation for each $a \in \mathbb{A}$ we will have $\varphi_{\text{sc}a}$ (called scoping interface function) whose domain is $\{\{c\} \mid c \in \mathbb{A}_r \setminus \{a, \hat{a}\}\} \cup \{\{a, \hat{a}\}\}$ and such that $\varphi_{\text{sc}a}(\{c\}) \doteq c$ for all $c \in \mathbb{A}_r \setminus \{a, \hat{a}\}$ and $\varphi_{\text{sc}a}(\{a, \hat{a}\}) \doteq \tau$. Another commonly used interface function is the identity $\varphi_{\text{id}}$ with the domain $\{\{a\} \mid a \in \mathbb{A}_r\}$ and such that $\varphi_{\text{id}}(\{a\}) \doteq a$, for all $a \in \mathbb{A}_r$.

\(^3\)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 $e, \times, i, b \notin \mathbb{B}$.
3.3. Labelled nets

A (marked) labelled net is, in the present framework, a tuple $\Sigma \overset{df}{=} (S, T, W, \lambda, M)$ 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 $\mathbb{N}$;
- $\lambda$ is a labelling for places and transitions such that $\lambda(s)$ is a symbol in $\{e, i, x\} \cup (\mathbb{B} \cup \{b\})$, for every place $s \in S$, and $\lambda(t)$ is an interface function $\varphi$ or an action in $A_r$, 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, $\Sigma$ 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 $s_0$ (resp. $s_1$). We shall also use $M^{cbr}$ and $M^{obr}$ to denote $M$ restricted to the control places and to the open buffer places, respectively. If $M^{cbr} = s_0$, we will say that $\Sigma$ is in an initial state (or marking), and if $M^{cbr} = s_1$, we will say that $\Sigma$ is in a final state (or marking). Finally, $\langle \Sigma \rangle$ is the labelled net obtained from $\Sigma$ 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 $\langle \rangle$ 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 $\Sigma$ with the index $\Sigma$; e.g., $T_\Sigma$ will denote the set of transitions of $\Sigma$. In order to simplify diagrams, we will omit isolated unmarked buffer places.

For every place (transition) $x$, we use $x_\bullet$ 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., $X^\bullet = \bigcup_{x \in X} x$.

3.4. Step sequences

A finite step sequence semantics of a labelled net $\Sigma$ captures the potential concurrency in the behaviour of the system modelled by $\Sigma$. A finite multiset of transitions $U$, called a step, is enabled by $\Sigma$ if for every place $s \in S$,

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

We denote by enabled($\Sigma$) the set of all steps enabled by $\Sigma$; notice that we always have $\emptyset \in \text{enabled}(\Sigma)$. A step $U \in \text{enabled}(\Sigma)$ can be executed, leading to a marking $M'$ given, for every place $s \in S$, by

$$M'(s) = M(s) - \sum_{t \in U} W(s, t) \cdot U(t) + \sum_{t \in U} W(t, s) \cdot U(t).$$
We will denote this by $\Sigma[U]\Sigma'$, where $\Sigma'$ is $\Sigma$ with the marking changed to $M'$. Transition labelling may be extended to steps, through the formula

$$\lambda(U) \equiv \sum_{t \in U} U(t) \cdot \lambda(t).$$

In particular, we will denote $\Sigma[\Gamma] \Sigma'$ whenever there is a multiset of transitions $U$ such that $\Sigma[U]\Sigma'$ 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 $\Sigma$ is a (possibly empty) sequence of steps, $\omega = U_1 \ldots U_k$, such that there are nets $\Sigma_1, \ldots, \Sigma_k$ satisfying $\Sigma[U_1] \Sigma_1[U_2] \Sigma_2[\ldots[U_k] \Sigma_k$. We will denote this by $\Sigma[\omega] \Sigma_k$ or $\Sigma_k \in [\Sigma]$, and call $\Sigma_k$ derivable from $\Sigma$ and its marking $M_{\Sigma_k}$ reachable from $M_{\Sigma}$ (with the convention that $\Sigma[\omega] \Sigma$ if $k = 0$, i.e., if $\omega$ is the empty step sequence). The definition of a labelled step sequence of $\Sigma$ is similar.

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

A marking $M$ of a labelled net $\Sigma$ is:

- clean if $M^{ctr} \geq ^* \Sigma \Rightarrow M^{ctr} = ^* \Sigma$ and $M^{ctr} \geq^\circ \Sigma \Rightarrow M^{ctr} = ^\circ \Sigma$.

- ac-free if, for every transition $t$, there is a control place $s \in \mathcal{C}$ 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 $s \in \mathcal{C}$ 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.\footnote{Note that one could introduce the two abstraction operators (i.e., scoping and tie) without using operator boxes as these are particularly simple (see figure 8); however, treating them in the same way as other operators leads to more uniform presentation.}

4.1. Asynchronous boxes and marking operators

An asynchronous box (or, shortly, a box) is a labelled net $\Sigma$ such that each transition is labelled by an action in $A_r$, and $\Sigma$ itself is:

- ex-restricted: there is at least one entry place and at least one exit place;
- **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 \( \cdot t \), and at least one control place in \( t^* \).

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

**Proposition 4.1.** Let \( \Sigma \) be a labelled net and \( \Theta = \langle \Sigma \rangle \).

1. \( \Sigma \) is a box iff \( \Theta \) is a box.

2. \( \Sigma \) is a static (dynamic) box iff \( \Theta \) is a static (dynamic) box.

3. \( \Sigma \) is quasi-safe iff \( \Theta \) is quasi-safe, whenever \( \Sigma \) is a static or dynamic box.

4. If \( \Sigma[U]\Sigma' \), then \( \Theta[U]\Theta' \), where \( \Theta' = \langle \Sigma' \rangle \).

5. If \( \Theta[U]\Theta' \), then \( \Sigma'[U]\Sigma'' \), for some \( \Sigma' \) and \( \Sigma'' \) such that \( \langle \Sigma' \rangle = \Theta \) and \( \langle \Sigma'' \rangle = \Theta' \).

6. If \( \Sigma \) is a dynamic box, then all the markings reachable from \( M_\Sigma \) are both clean and ac-free. Moreover, if \( \Sigma \) 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). \( \square \)

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 \( \Sigma \) be a labelled net and \( \Sigma[U]\Sigma' \).

1. If \( \Sigma \) is a static box, then \( U = \emptyset \) and \( \Sigma = \Sigma' \).

2. If \( \Sigma \) is a dynamic (and quasi-safe) box, then \( U \) is a set of transitions\(^5\) and \( \Sigma' \) is a dynamic (and quasi-safe) box.

\(^5\)But if \( \Sigma'[U]\Sigma' \), then the labelled step \( \Gamma \) may be a true multiset of actions; see, e.g., scenario I in section 2.
Proof:
(1) That $U = \emptyset$ follows from $\Sigma^{ctr} = (\Sigma)^{ctr} = \emptyset$ and the control-restrictedness of any box. Moreover, $\Sigma = \Sigma'$ follows from $U = \emptyset$.

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

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 $\Xi$, shown in figure 7, where $\Xi$ is static and quasi-safe. All the remaining static and dynamic boxes of the Asynchronous Box Calculus will be constructed from the $\Xi$'s through various operators.

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

- $\Sigma.B$ adds $B(b)$ tokens to the $b$-labelled open buffer place of $\Sigma$, for each $b \in \mathbb{B}$; in particular, $\Sigma.b \equiv \Sigma,\{b\}$ adds one token to the $b$-labelled buffer place of $\Sigma$. This operation will be called buffer stuffing.

- $\Sigma$ (resp. $\Sigma$) is $\Sigma$ with one additional token in each entry (resp. exit) place, i.e., $M_{\Sigma} \equiv M_{\Sigma} + \Sigma$ (resp. $M_{\Sigma} \equiv M_{\Sigma} - \Sigma$).

- $|\Sigma|$ is $\Sigma$ with all the tokens in its control places removed, and $|\Sigma|$ is $\Sigma$ with the empty marking. Both notations extend component-wise to tuples of boxes.

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

1. $\Sigma.\emptyset = \Sigma$, $\Sigma.B, B' = \Sigma.(B + B')$, $\Sigma.B = \Sigma.B$ and $\Sigma.B = \Sigma.B$.

2. $\langle |\Sigma| \rangle = \langle |\Sigma| \rangle = \langle |\Sigma| \rangle = \langle |\Sigma| \rangle$.

3. $\langle |\Sigma| \rangle = \langle |\Sigma| \rangle$, $\langle \Sigma \rangle = \langle \Sigma \rangle$, $\langle |\Sigma| \rangle = \langle |\Sigma| \rangle$ and $\langle \Sigma, B \rangle = \langle \Sigma \rangle = \langle \Sigma \rangle = \langle \Sigma \rangle$.

4. $\langle \Sigma, B \rangle = \langle \Sigma \rangle$, $\langle \Sigma \rangle = \langle |\Sigma| \rangle$, $\langle \Sigma \rangle = \langle |\Sigma| \rangle$ and $\langle |\Sigma| \rangle = \langle |\Sigma| \rangle$.

5. $\Sigma$ is static (and quasi-safe) iff $\Sigma.B$ is static (and quasi-safe).
6. $\Sigma$ is dynamic (and quasi-safe) iff $\Sigma.B$ is dynamic (and quasi-safe).

7. $\Sigma$ is dynamic (and quasi-safe) iff $\Sigma$ is static (and quasi-safe) iff $\Sigma$ is dynamic (and quasi-safe).

8. If $\Sigma$ is static or dynamic (and quasi-safe), then $[\Sigma]$ and $[\Sigma]$ 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 $\Sigma$ is dynamic iff $\Sigma$ is static (the other parts of the result can be shown in a similar way). Suppose that $\Sigma$ is a dynamic box. Then $M(\Sigma) = \emptyset$. This, together with the fact that $\Sigma$ is dynamic and $\circ(\Sigma) = \circ(\Sigma^*)$ and $(\Sigma)^o = (\Sigma)^o$, means that $\Sigma$ is a static box. Suppose now that $\Sigma$ is a static box. Then $M(\Sigma) = \emptyset$ and so $M(\Sigma) = \circ(\Sigma)$. Thus, by part (4) and the ex-restrictedness of any box, $M(\Sigma) = \circ(\Sigma) \neq \emptyset$. This, together with the fact that $\Sigma$ is static and $M(\Sigma) = \circ(\Sigma) = \circ(\Sigma)$ and $(\Sigma)^o = (\Sigma)^o$, means that $\Sigma$ is a dynamic box.

(8) Follows from the definitions and the fact that $M([\Sigma]) = M([\Sigma]) = \emptyset$. □

### 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 $\Sigma$ is $\text{fts}_\Sigma \triangleq (V, L, A, \text{init})$ where $V \triangleq [\Sigma]$ is the set of states; $L \triangleq 2^{T_\Sigma}$ is the set of arc labels; $A \triangleq \{(\Sigma', U, \Sigma'') \in V \times L \times V \mid \Sigma'[U] \Sigma''\}$ is the set of arcs; and $\text{init} \triangleq \Sigma$ is the initial state. For a static box $\Sigma$, $\text{fts}_\Sigma \triangleq \text{fts}_\Sigma$.

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

As usual, if $(V, L, A, \text{init})$ and $(V', L', A', \text{init}')$ are two transition systems, then an isomorphism between them is a bijection $\text{iso} : V \to V'$ such that $\text{iso}(\text{init}) = \text{init}'$ and, for all states $v, w \in V$ and labels $\ell \in L \cup L'$, $(v, \ell, w) \in A$ iff $(\text{iso}(v), \ell, \text{iso}(w)) \in A'$. 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 $\Omega$ is an unmarked, finite, simple, ex-restricted and control-restricted labelled net with only control places (hence it is not $B$-restricted) and such that every transition $v$ is labelled by an interface function. For every operator box $\Omega$, 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) $\Sigma = (\Sigma_1, \ldots, \Sigma_n)$, will be referred to as an $\Omega$-tuple (or, simply, a tuple); we will also use $\Sigma_i$ to denote $\Sigma_i$, for $i \leq n$. The notation $\Sigma$ will be used in the net substitution operation, denoted by $\Omega(\Sigma)$, 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 $a \in \mathcal{A}$ and $b \in \mathcal{B}$.

**Choice** $\Omega_\Box$, **iteration** $\Omega_\oplus$ and **sequence** $\Omega_\circ$. The first three operator boxes specify different ways to sequentially compose behaviours. They are all binary, with the domains of application $\text{dom} \, \Omega_\Box = \text{dom} \, \Omega_\oplus = \text{dom} \, \Omega_\circ = \text{dom} \, \Omega_\circ a = (\text{abox}_{\text{stc}})^2 \cup (\text{abox}_{\text{dyn}} \times \text{abox}_{\text{stc}}) \cup (\text{abox}_{\text{stc}} \times \text{abox}_{\text{dyn}})$, so that at most one of their two operands is ever dynamic. We will denote: $\Sigma_1 \Box \Sigma_2 \overset{d}{=} \Omega_\Box (\Sigma_1, \Sigma_2)$ and $\Sigma_1 \circ \Sigma_2 \overset{d}{=} \Omega_\circ (\Sigma_1, \Sigma_2)$.

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

**Scoping** $\Omega_{\text{sc}} a$. Parameterised by a communication action $a \in \mathcal{A}$, this is a unary operator with the domain of application $\text{dom} \, \Omega_{\text{sc}} a = \text{abox}_{\text{stc}} \cup \text{abox}_{\text{dyn}}$. The change of the synchronous communication interface it provides is captured by $\varphi_{\text{sc}} a$, already defined; essentially, this forces the synchronisation of the pairs of actions $a$ and $\bar{a}$. We will denote $\Sigma \text{sc} a \overset{d}{=} \Omega_{\text{sc}} a (\Sigma)$.

**Buffer restriction** $\Omega_{\text{tie}} b$. Parameterised by a buffer $b \in \mathcal{B}$, this unary operator has the domain of application $\text{dom} \, \Omega_{\text{tie}} b = \text{abox}_{\text{stc}} \cup \text{abox}_{\text{dyn}}$. Buffer restriction will hide the $b$-labelled open buffer place of the box it is applied to. We will denote $\Sigma \text{tie} b \overset{d}{=} \Omega_{\text{tie}} b (\Sigma)$.

**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 \( \eta \) of basic transition identities and a corresponding set of basic labelled trees with a single node labelled with an element of \( \eta \). 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:

- \( v \leftarrow T \), 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 \( v \).
- \( v \leftarrow t \) denotes the tree \( v \leftarrow \{ t \} \), and \( v \leftarrow T \) denotes the set of trees \( \{ v \leftarrow t \mid t \in T \} \).

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](image-url)

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 \( v_1 \leftarrow v_2 \leftarrow v_1 \leftarrow v_2 \leftarrow v_1 \leftarrow \{ v_1 \leftarrow v_2 \leftarrow v_1 \leftarrow \} \).

**Binary operators.** Let \( \Omega = \Omega_{bin} \) (for \( bin \in \{ \square, \otimes, :, \} \)) be a binary ABC operator box, \( v_i = v_i^{bin} \) (for \( i \in \{1, 2\} \)) be its transitions, and \( \Sigma = (\Sigma_1, \Sigma_2) = (\Sigma_{v_1}, \Sigma_{v_2}) \) be a pair of boxes. The result of
a simultaneous substitution of the transitions $v_1$ and $v_2$ in $\Omega$ by the boxes $\Sigma_1$ and $\Sigma_2$ is a labelled net $\Omega(\Sigma) = \Phi$ whose components are defined as follows.

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

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

and similarly for $W_{\Phi}(w \prec t, p)$.

For every place $s \in S_{\Omega}$ with $s = \{u_1, \ldots, u_k\}$ and $s^\bullet = \{w_1, \ldots, w_m\}$, we add to $S_{\Phi}$ all places of the form $p = (x_1, \ldots, x_k, e_1, \ldots, e_m)$, where each $x_i$ (if any) is an exit place of $\Sigma_{u_i}$, and each $e_j$ (if any) is an entry place of $\Sigma_{w_j}$. 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 $w \prec t$, the weight function is given by:

$$W_{\Phi}(p, w \prec t) \triangleq \begin{cases} W_{\Sigma_u}(x_i, t) + W_{\Sigma_w}(e_j, t) & \text{if } w \in s \setminus s^\bullet \text{ and } w = u_i = w_j \\
W_{\Sigma_u}(x_i, t) & \text{if } w \in s \setminus s^\bullet \text{ and } w = u_i \\
W_{\Sigma_w}(e_j, t) & \text{if } w \in s^\bullet \setminus s \text{ and } w = w_j \\
0 & \text{otherwise} \end{cases}$$

and similarly for $W_{\Phi}(w \prec t, p)$. 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 $p_b = (p_{v_1}^b, p_{v_2}^b) \in S_{\Phi}$, where $p_{v_1}^b$ and $p_{v_2}^b$ are the unique $b$-labelled places of respectively $\Sigma_1$ and $\Sigma_2$. The marking of $p_b^b$ is the sum of the markings of $p_{v_1}^b$ and $p_{v_2}^b$, and for each transition $w \prec t$, the weight function is given by:

$$W_{\Phi}(p_b^b, w \prec t) \triangleq W_{\Sigma_u}(p_w^b, t),$$

and similarly for $W_{\Phi}(w \prec t, p_b^b)$.

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

$$W_{\Omega_{\Sigma_c}(\Sigma)}(p, t) \triangleq W_{\Sigma}(p, t_1) + W_{\Sigma}(p, t_2),$$

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

**Buffer restriction.** Applying the buffer restriction operator $\Omega_{\Sigma_b}$ to a box $\Sigma$ results in a labelled net like $\Sigma$ with the only difference that the identity of each transition $t \in T_{\Sigma}$ is changed to $v_{\Sigma_b} \prec t$, the label of the only $b$-labelled place is changed to $b$, and a new unmarked isolated $b$-labelled place is added to $S_{\Omega_{\Sigma_b}(\Sigma)}$. 
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 \( \diamond \in \), if \( \Sigma \) has the same places with the same labels (but possibly different markings) as \( \Sigma' \), and the same holds for \( \Theta \) and \( \Theta' \), then the places and their labels (but not necessarily the markings) will be the same for the composite nets \( \Sigma \diamond \Theta \) and \( \Sigma' \diamond \Theta' \). 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 \( \| \) 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 \( \Sigma \) and \( \Theta \) be boxes such that \( |\Sigma| = |\Theta| \), \( a \in \mathcal{A} \), \( b \in \mathcal{B} \) and \( B \in \text{mult}(\mathbb{B}) \). Then

\[
\Sigma \text{sc} \ a = \Theta \text{sc} \ a \text{ iff } \Sigma \text{tie} \ b = \Theta \text{tie} \ b \text{ iff } \Sigma.B = \Theta.B \text{ iff } \Sigma = \Theta.
\]

**Proposition 4.5.** Let \( \Sigma \) be a box, \( a \in \mathcal{A} \), \( b \in \mathcal{B} \) and \( B \in \text{mult}(\mathbb{B}) \).

1. \( |\Sigma \text{tie} \ b| = |\Sigma| \text{tie} \ b \) and \( |\Sigma \text{sc} \ a| = |\Sigma| \text{sc} \ a \) and \( |\Sigma.B| = |\Sigma|.B \).
2. \( |\Sigma \text{tie} \ b| = |\Sigma| \text{tie} \ b \) and \( |\Sigma \text{sc} \ a| = |\Sigma| \text{sc} \ a \).
3. \( \langle \Sigma \text{tie} \ b\rangle = \langle \Sigma \rangle \text{tie} \ b \) and \( \langle \Sigma \text{sc} \ a \rangle = \langle \Sigma \rangle \text{sc} \ a \).
4. \( M_{\Sigma, B}^\text{ctr} = M_{\Sigma. B}^\text{ctr} = M_{\Sigma}^\text{ctr} \) and \( M_{\Sigma \text{sc} \ a} = M_{\Sigma} \).

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 \( \Omega \) be any binary ABC operator box, and \( \Sigma_1, \Sigma_2 \) and \( \Sigma \) be any boxes.

1. \( M_{\Omega(\Sigma_1, \Sigma_2)}^\text{ctr} = \emptyset \) iff \( M_{\Sigma_1}^\text{ctr} = \emptyset = M_{\Sigma_2}^\text{ctr} \).
2. \( M_{\Sigma'}^\text{ctr} = \emptyset \) iff \( M_{\Sigma}^\text{ctr} = \emptyset \), for each \( \Sigma' \) of the form \( \Sigma \text{sc} \ a \) or \( \Sigma \text{tie} \ b \) 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 \( \Omega \) be any binary ABC operator box and \( \Sigma \in \text{dom}_\Omega \). Then \( \Omega(\Sigma) \) is a box with a clean and ac-free marking. Moreover, if all the dynamic boxes (if any) in \( \Sigma \) have quasi-safe markings, then the marking of \( \Omega(\Sigma) \) is also quasi-safe.

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

**Theorem 4.8.** Let \( \Sigma \) be a static or dynamic box, \( a \in \mathbb{A}, b \in \mathbb{B} \) and \( B \in \text{mult}(\mathbb{B}) \). Then \( \Sigma \in a, \Sigma \in b \) and \( \Sigma \in B \) are boxes with clean and ac-free markings. Moreover, if \( \Sigma \) 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 \( \Omega \) be an ABC operator box and \( \Sigma \in \text{dom}_\Omega \).
Then \( \langle \Sigma \rangle \in \text{dom}_\Omega \) and \( \Omega(\langle \Sigma \rangle) = \langle \Omega(\Sigma) \rangle \).

**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 \( \Sigma_{ab^+}, \Sigma_{ab^-} \) and \( \Sigma_{ab^\beta} \), 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 \( \Omega \) to two different \( \Omega \)-tuples of boxes yields the same result. We start by defining five auxiliary relations \( \equiv_i^\Omega \) which identify pairs of \( \Omega \)-tuples containing boxes with particular combinations of initial and/or final markings, in such a way that the result of applying \( \Omega \) leads to the same marking for both tuples.

For an operator box \( \Omega \) and two \( \Omega \)-tuples of boxes, \( \Sigma \) and \( \Theta \), we have:

\[ \equiv_i^\Omega \]

---

\(^6\)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.
• \( \Sigma \equiv_0 \Theta \) if \( \Sigma \) and \( \Theta \) are the same except that \( \Sigma_v = \overline{\Psi} \) and \( \Theta_v = \overline{\Psi} \), for some box \( \Psi \) and a transition \( v \in T_\Omega \) satisfying \( \overline{v} = v^* \). 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.

• \( \Sigma \equiv_1 \Theta \) if \( \Sigma \) and \( \Theta \) are the same except that \( \Sigma_v = \overline{\Theta}_v \) and \( \Sigma_w = \Theta_w \) (or \( \Sigma_v = \Theta_v \) and \( \Sigma_w = \overline{\Theta}_w \)), for some distinct transitions \( v, w \in T_\Omega \) satisfying \( \overline{v} = \overline{w} \). 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.

• \( \Sigma \equiv_2 \Theta \) if \( \Sigma \) and \( \Theta \) are the same except that \( \Sigma_v = \Theta_v \) and \( \Sigma_w = \Theta_w \) (or \( \Sigma_v = \Theta_v \) and \( \Sigma_w = \overline{\Theta}_w \)), for some distinct transitions \( v, w \in T_\Omega \) satisfying \( v^* = w^* \). This relation, involving final markings, is similar to the previous one but intended only for the choice operator.

• \( \Sigma \equiv_3 \Theta \) if \( \Sigma \) and \( \Theta \) are the same except that \( \Sigma_v = \Theta_v \) and \( \Sigma_w = \overline{\Theta}_w \), for some distinct transitions \( v, w \in T_\Omega \) satisfying \( v^* = \overline{w} \). 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.

• \( \Sigma \equiv_4 \Theta \) if \( \Sigma = \Theta \). This relation is simply the identity on \( \Omega \)-tuples.

• \( \Sigma \equiv_5 \Theta \) if, for each \( v \in T_\Omega \), there are is a box \( \Psi_v \) and two multisets over \( B_v \) 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, \( \Sigma \) and \( \Theta \) 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 \( \Omega \) leads to the same result:\(^8\)

\[
\equiv_\Omega = \equiv_5 \circ \bigcup_{i=0}^4 \equiv_i^\Omega .
\]

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

\[
\begin{align*}
\equiv_{\|} &= \equiv_5 \\
\equiv_{\bowtie} &= \equiv_5 \\
\equiv_{\bowtie^*} &= \equiv_5 \\
\equiv_{\bowtie^{\text{M}}} &= \equiv_5 \\
\equiv_{\bowtie^{\text{M},\text{A}}} &= \equiv_5 \\
\equiv_{\bowtie^{\text{M},\text{B}}} &= \equiv_5 \\
\end{align*}
\]

where \( \equiv_5 \) is the identity relation on boxes. For example, \( \equiv_{\|} \) relates all those pairs of boxes which may differ only by the distribution of tokens in the open buffer places; whereas \( \equiv_{\bowtie} \) 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 stuff never changes the marking of closed buffer places.

\(^8\)The composition of two binary relations \( R, R' \) on a set \( X \) is defined as \( R \circ R' = \{(x, y) \mid \exists z \in X : (x, z) \in R \land (z, y) \in R'\} \).
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:

$$\left(\overline{\Sigma_1}, \Sigma_2\right) \equiv_\Omega \left(\overline{\Sigma_1}, \Sigma_2\right) \equiv_\Omega \left(\Sigma_1, \overline{\Sigma_2}\right) \neq_\Omega \left(\Sigma_1, \Sigma_2\right).$$

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 $\Omega$ makes it transitive.

**Proposition 5.1.** Let $\Omega$ be an ABC operator box, $\Sigma \in \text{dom}_\Omega$, and $\Theta$ be an $\Omega$-tuple of boxes.

1. If $\Sigma \equiv_\Omega \Theta$, then $\Theta \in \text{dom}_\Omega$ and $[\Sigma] = [\Theta]$.
   Moreover, if all the boxes in $\Sigma$ are quasi-safe then so are all the boxes in $\Theta$.
2. $\equiv_\Omega$ is an equivalence relation on $\text{dom}_\Omega$.
3. If $[\Sigma] = [\Theta]$, then $\Omega(\Sigma) = \Omega(\Theta)$ iff $\Sigma \equiv_\Omega \Theta$.

**Proof:**
Follows from proposition A.6 in the appendix. \hfill \Box

It is interesting to note that the precondition $[\Sigma] = [\Theta]$ in the last part of the above proposition cannot be dropped; for example, we have $\Sigma_{ab^+} \neq_\Omega \Sigma_{ab^-}$ and $\Sigma_{ab^+}.sc\ a = \Sigma_{ab^-}.sc\ a$ (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 $\Omega$ be a binary ABC operator, $\Sigma$ and $\Theta$ be static boxes, $\Phi$ be a static or dynamic box, $a \in A$, $b \in B$ and $B', B'' \in \text{mult}(B)$.

1. $\overline{\Sigma \square} \Theta = \overline{\Sigma} \square \Theta = \Sigma \square \Theta$ and $\overline{\Sigma \square} \Theta = \Sigma \square \Theta = \Sigma \square \Theta$.
2. $\overline{\Sigma \circ} \Theta = \overline{\Sigma} \circ \Theta = \Sigma \circ \Theta = \Sigma \circ \Theta$ and $\overline{\Sigma \circ} \Theta = \Sigma \circ \Theta$.
3. $\overline{\Sigma; \Theta} = \overline{\Sigma; \Theta} = \Sigma; \Theta = \Sigma; \Theta$ and $\overline{\Sigma; \Theta} = \Sigma; \Theta$.
4. $\overline{\Sigma||\Theta} = \overline{\Sigma||\Theta} = \Sigma||\Theta$.
5. $\overline{\Sigma.sc\ a} = \Sigma.sc\ a$ and $\overline{\Sigma.sc\ a} = \Sigma.sc\ a$.
6. $\overline{\Sigma.tie\ b} = \overline{\Sigma.tie\ b} = \Sigma.tie\ b$.
7. $(\Phi.sc\ a).B = (\Phi.B).sc\ a$.
8. $(\Phi.tie\ b).B = (\Phi.B).tie\ b$ provided that $b \notin B$.
9. If $\Sigma = (\Sigma_1, \Sigma_2) \in \text{dom}_\Omega$, then $\Omega(\Sigma_1.B, \Sigma_2.B') = \Omega(\Sigma).(B + B')$.

**Proof:**
Follows from proposition A.5 in the appendix, proposition 5.1(3), and the definitions. \hfill \Box
5.2. Dynamic properties of composite nets

In the results presented below, we capture the \textit{behavioural compositionality} of our model, \textit{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 $\Omega(\Sigma)$, knowing the steps enabled by the boxes in $\Sigma$.

**Proposition 5.3.** Let $\Omega_{bin} \in \{\Omega_{\parallel}, \Omega_{\square}, \Omega_{\oplus}, \Omega_{\oplus\ominus}\}$ and $\Sigma \in \text{dom}_{\Omega_{bin}}$. Then $\text{enabled}(\Omega_{bin}(\Sigma))$ comprises exactly all sets of transitions $U = (v_{1}^{bin} \downarrow U_{1}) \cup (v_{2}^{bin} \downarrow U_{2})$ such that there is a pair of boxes $\Theta$ satisfying $\Theta \equiv_{\Omega_{bin}} \Sigma$ and $U_{i} \in \text{enabled}(\Theta_{i})$, for $i \in \{1, 2\}$. Moreover, $\Omega_{bin}(\Sigma)[U] \Omega_{bin}(\Phi)$, where $\Theta_{i}[U_{i}] \Phi_{i}$, 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 $\Sigma_{1}$ and $\Sigma_{2}$, to ensure that the steps $U_{1}$ and $U_{2}$ are enabled separately. For the other $\Omega$’s, the result follows from proposition A.7 in the appendix.

Note that, for $\Omega_{bin} \neq \Omega_{||}$, 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 $\Sigma$ be a static or dynamic box, $a \in \mathbb{A}$, $b \in \mathbb{B}$, $B \in \text{mult}(\{b\})$ and $B' \in \text{mult}(\mathbb{B})$.

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

$$U = (v^{\text{sc } a} \downarrow Z) \cup \{v^{\text{sc } a} \downarrow \{v_{1}, w_{1}\}, \ldots, v^{\text{sc } a} \downarrow \{v_{k}, w_{k}\}\}$$

such that $Z \cup V \cup W \in \text{enabled}(\Sigma)$, where the steps $V \overset{\text{def}}{=} \{v_{1}, \ldots, v_{k}\}$, $W \overset{\text{def}}{=} \{w_{1}, \ldots, w_{k}\}$ and $Z$ satisfy $a, \partial \notin \lambda_{\Sigma}(Z)$, and for all $i \in \{1, \ldots, k\}$, $\lambda_{\Sigma}(v_{i}) = \{a\}$ and $\lambda_{\Sigma}(w_{i}) = \{\partial\}$. Moreover, $\Sigma \text{ sc } a[U] \Phi \text{ sc } a$, where $\Sigma[V \cup W \cup Z] \Phi$.

2. $\text{enabled}(\Sigma \text{ tie } b)$ comprises exactly all sets of transitions $U = v^{\text{tie } b} \downarrow V$ such that $V \in \text{enabled}(\Sigma)$.

Moreover, $\Sigma \text{ tie } b[U] \Phi \text{ tie } b$, where $\Sigma[V] \Phi$.

3. $\text{enabled}(\Sigma \text{ tie } b).B$ comprises exactly all sets of transitions $U \in \text{enabled}(\Sigma \text{ tie } b)$.

Moreover, $(\Sigma \text{ tie } b).B[U] \Phi.B$, where $\Sigma \text{ tie } b[U] \Phi$.

4. $\text{enabled}(\Sigma)$ is a subset of $\text{enabled}(\Sigma.B')$.

Moreover, if $\Sigma[U] \Phi$ then $\Sigma.B'[U] \Phi.B'$.

**Proof:**

Follows directly from the definitions.

That the last property does not hold for the equality of sets may be illustrated by a simple counterexample: the dynamic box $\Sigma_{ab}$ only allows the empty step, while $\Sigma_{ab}.b[\{a^{ab}^{-}\}] \Sigma_{ab}$. The behaviours of the basic asynchronous boxes of ABC are captured below.

**Proposition 5.5.** Let $B \in \text{mult}(\mathbb{B})$ and $\Sigma = \Sigma^{\alpha}_{B}$, where $\Sigma^{\alpha}$ is one of the basic boxes in figure 7.
1. For \( \alpha \in \{a, ab^+\} \), the non-empty steps of \( \Sigma \) are respectively
\[
\Sigma\{v^a\} \sum_{a} B \quad \text{and} \quad \Sigma\{v^{ab^+}\} \sum_{ab^+} b.B .
\]

2. For \( \alpha \in \{ab^-, ab^\pm\} \), if \( b \notin B \) then \( \Sigma \) has no non-empty step; otherwise the non-empty steps are respectively
\[
\Sigma\{v^{ab^-}\} \sum_{ab^-} (B - \{b\}) \quad \text{and} \quad \Sigma\{v^{ab^\pm}\} \sum_{ab^\pm} B .
\]
Moreover, if \( \Sigma = \sum_{\alpha} B \) or \( \Sigma = \sum_{\alpha} B \), then \( \Sigma \) 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 \( \Omega \) be an operator box of ABC and \( \Sigma \in dom_\Omega \). Then \( \Omega(\Sigma) \) is a static or dynamic box. Moreover,

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

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

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

(ii) If no box in \( \Sigma \) is dynamic, then every net derivable from \( \Sigma \) or \( \Theta \) is of the form \( \Omega(\Theta) \), where \( \Theta \in dom_\Omega \). Moreover, if all the boxes in \( \Sigma \) are quasi-safe then all the boxes in \( \Theta \) 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 \( \text{box}_{ABC} \)), and distinguish: ABC boxes with control tokens (denoted by \( \text{box}_{ABC}^{\text{ctr}} \)), and ABC boxes without control tokens (denoted by \( \text{box}_{ABC}^{\text{noctr}} \)). They are defined as the smallest sets of boxes satisfying the following (below \( B \) is any multiset over \( \mathbb{N} \)):

- Each basic box in figure 7 belongs to \( \text{box}_{ABC}^{\text{noctr}} \).
- If \( \Sigma \in \text{box}_{ABC}^{\text{noctr}} \) then \( \sum, \Sigma \in \text{box}_{ABC}^{\text{ctr}} \).
• If $\Sigma \in \text{box}_{ABC}^{\text{noctr}}$ then $\Sigma . B \in \text{box}_{ABC}^{\text{noctr}}$, and if $\Sigma \in \text{box}_{ABC}^{\text{ctr}}$ then $\Sigma . B \in \text{box}_{ABC}^{\text{ctr}}$.

• If $\Sigma \in \text{box}_{ABC}$ then $[\Sigma] , [\Sigma] \in \text{box}_{ABC}^{\text{noctr}}$.

• If $\Omega$ is an operator box of ABC and $\Sigma$ is an $\Omega$-tuple of ABC boxes then $\Omega(\Sigma)$ is an ABC box, provided that the tuple would have belonged to $\text{dom}_\Omega$ if we replaced $\text{box}_{ABC}^{\text{str}}$ and $\text{box}_{ABC}^{\text{dyn}}$ respectively by $\text{box}_{ABC}^{\text{noctr}}$ and $\text{box}_{ABC}^{\text{ctr}}$ in the definition of $\text{dom}_\Omega$. Moreover, if all the boxes in $\Sigma$ belong to $\text{box}_{ABC}^{\text{noctr}}$ then so does $\Omega(\Sigma)$; otherwise $\Omega(\Sigma)$ belongs to $\text{box}_{ABC}^{\text{ctr}}$.

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 $\Sigma$ be an ABC box. The proof proceeds by induction on the number $k$ of applications of various operators used in the process of constructing $\Sigma$ (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 ($k = 0$), $\Sigma$ 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: $\Sigma = \Theta$ or $\Sigma = \emptyset$. Then $\Theta \in \text{box}_{ABC}^{\text{noctr}}$ and so, by the induction hypothesis, $\Theta$ is a quasi-safe static box. Thus, by proposition 4.3(7), $\Sigma$ is a quasi-safe dynamic box.

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

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

Case 4: $\Sigma = \Omega(\Sigma)$. Then, by the induction hypothesis, $\Sigma$ is an $\Omega$-tuple of quasi-safe static and/or dynamic boxes. Moreover, by the definition of ABC boxes, $\Sigma \in \text{dom}_\Omega$ and so, by theorem 5.6, $\Omega(\Sigma)$ is a quasi-safe static or dynamic box. Finally, using proposition 4.6, one can see that $\Sigma \in \text{box}_{ABC}^{\text{noctr}}$ iff $\Sigma$ 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 $\Sigma$ is an $\Omega$-tuple of ABC boxes, for some ABC operator box, and $\Sigma \equiv_\Omega \Theta$, then $\Theta$ is also an $\Omega$-tuple of ABC boxes.

### 6. An algebra of asynchronous box expressions

We consider the algebra of process expressions over the signature:

$$\mathcal{A} \cup \{ \text{sc} a \ | \ a \in \mathcal{A} \} \cup \{ \text{tie} b \ | \ b \in \mathcal{B} \} \cup \{ . b \ | \ b \in \mathcal{B} \} \cup \{ \text{in} , , \} \cup \{ , , , \} ,$$

where $\mathcal{A} = \mathcal{A}_e \cup \{ a b^+ , a b^- , a b^\# \ | \ a \in \mathcal{A}_e \land b \in \mathcal{B} \}$ are the constants; the unary operators $\text{sc} a$, $\text{tie} b$ and $b.$ are used in the postfix mode; $\text{in}$ and $\text{in}$ are two positional unary operators (the position of the argument being given by the dot); and the binary operators $\|$, $\;$, $\square$ and $\otimes$ 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 $\text{expr}^{\text{stc}}$ and $\text{expr}^{\text{dyn}}$. Collectively, we will refer to them as the (asynchronous) box expressions, $\text{expr}$. Their syntax is given by:

\[
\begin{align*}
\text{expr}^{\text{stc}} & \quad E \ ::= \quad \alpha \mid E \ \text{sc} \ a \mid E \ \text{tie} \ b \mid E.b \mid E \ | E \mid E \ \square \ E \\
& \quad E; E \mid E \ \bar{\oplus} E \\

\text{expr}^{\text{dyn}} & \quad D \ ::= \quad \bar{E} \mid \underline{E} \mid D \ \text{sc} \ a \mid D \ \text{tie} \ b \mid D.b \mid D \ | D \\
& \quad D \ \square \ E \mid E \ \square \ D \mid D; E \mid E; D \mid D \ \bar{\oplus} E \mid E \ \bar{\oplus} D
\end{align*}
\]

where $\alpha \in A$, $a \in A$, and $b \in B$. 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 $F$ 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 $F;b\ldots b$ (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 $\bar{E}$ 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 $a b^+; f b^-$ has a static component with a $b$ in it, and may be transformed into an equivalent $a b^+; f b^- . b$.

### 6.1. Denotational semantics

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

\[
\begin{align*}
\text{box}(\alpha) & \overset{df}{=} \Sigma\alpha \\
\text{box}(\bar{E}) & \overset{df}{=} \text{box}(E) \\
\text{box}(\underline{E}) & \overset{df}{=} \text{box}(E) \\
\text{box}(F \ \text{una}) & \overset{df}{=} \text{box}(F) \ \text{una} \\
\text{box}(F_1 \ \bin \ F_2) & \overset{df}{=} \text{box}(F_1) \ \bin \ \text{box}(F_2) .
\end{align*}
\]

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

**Proof:**

Follows from theorem 5.6, by induction on the structure of $F$. \(\square\)

\(^9\)Note that, w.r.t. the original PBC and PNA algebras, the above syntax uses slightly different symbols to denote scoping ($E \ \text{sc} \ a$ instead of $[a : E]$) and iteration ($E @ E'$ instead of $E \ast E'$). However, their meanings remain unchanged.
What is more, it follows directly from the definitions that $\text{box}(F)$ is an ABC box and so, by theorem 5.7, it is quasi-safe.

**Proposition 6.2.** Let $F$ be a box expression. Then $\left[F\right]$ and $\left[F'\right]$ are box expressions satisfying the following.

1. $\text{box}(\left[F\right]) = \left[\text{box}(F)\right]$.  
2. $\text{box}(\left[F'\right]) = \left[\text{box}(F)\right]$.  

**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 $(a \square b)(d; e) \equiv (\pi \square b)(\overline{d}; e)$, as in figure 10.

![Derivation Tree](image)

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

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 $b_1$, OPL and OPR). However, no $b$ may ever cross the boundary imposed by the $\text{tie}\ b$ operator. Indeed, the rule $b_1$ excludes $\text{una}$ from being $\text{tie}\ b$, but not from being $\text{tie}\ b'$, provided that $b \neq b'$. The same rule, with $\text{una} = .b'$, implies that $F.b.b' \equiv F'.b'$, i.e., the commutativity of the buffer stuffing operator. Similarly, with $\text{una} = .b$, the rules $E_1$ and $X_1$ lead to $E.b \equiv E.b$ and $E.b \equiv E.b$, while the combination of the rules OPR and OPL leads to $(F.b)\bin F' \equiv F'\bin(F',b)$, 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.
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 $[F_1] \equiv [F_2]$, $[F_1] = [F_2]$ and $\text{box}(F_1) = \text{box}(F_2)$.
2. If $[F_1] \equiv [F_2]$, then $[F_1] = [F_2]$ and $\text{box}(F_1) = \text{box}(F_2)$.
3. If $[F_1] \equiv [F_2]$, then $[F_1] = [F_2]$ and $\text{box}(F_1) = \text{box}(F_2)$.
4. If $F_1 \equiv \overline{F}_2$, then $[F_1] \equiv [F_2]$, $[F_1] = [F_2]$ and $\text{box}(F_1) = \text{box}(F_2)$.
5. If $F_1 \equiv \overline{F}_2$, then $[F_1] \equiv [F_2]$, $[F_1] = [F_2]$ and $\text{box}(F_1) = \text{box}(F_2)$.

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 $M_{\text{box}(F)}^{\text{str}} = \text{box}(F)$ (resp. $M_{\text{box}(F)}^{\text{str}} = \text{box}(F)$).

Proposition 6.4. Let $F$ be a dynamic box expression.
1. \( F \) is initial iff \( F \equiv \boxed{F} \).

2. \( F \) is final iff \( F \equiv \boxed{F} \).

Proof:
The (\( \iff \)) in part (1) can be shown as follows. From proposition 6.3(1) and the translation rules (5), it follows that \( \text{box}(F) = \text{box}(\boxed{F}) = \boxed{\text{box}(F)} \). Moreover, by theorem 6.1(2), \( \text{box}(\boxed{F}) \) is static and so \( \overline{\text{box}}(F) = \overline{\text{box}}(\boxed{F}) = \boxed{\overline{\text{box}}(F)} \), and so \( F \) is initial. For part (2) the proof is similar.

To prove the (\( \iff \)) implications, we proceed by induction on the structure of \( F \). In the base step, \( F = \boxed{E} \) or \( F = \boxed{\overline{E}} \), 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: \( F = F' \text{ un}a \), for some unary ABC operator \( \text{un}a \). Then, from \( F \) being initial and the translation rules (5) as well as proposition 4.5(4) and \( \overline{\text{box}}(F') \overline{\text{un}}a = \overline{\text{box}}(F') \), we have that \( F' \) is also initial. Hence, by the induction hypothesis, \( F' \equiv \boxed{F'} \). We then obtain, using the rules \( \text{e}1 \) and \( \text{con}1 \), that \( F = F' \text{ un}a \equiv \boxed{F'} \text{ un}a \equiv \boxed{F'} \equiv \boxed{F} \).

Case 2: \( F = F' \text{ bin} F'' \), for some binary ABC operator \( \text{bin} \). 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 \( M^{\text{bin}}_{\text{box}}(F') \text{ un}a = \overline{\text{box}}(F') \text{ un}a = \text{bin}(F'') \). Hence, by lemma A.3(3) and without loss generality, \( M^{\text{bin}}_{\text{box}}(F') = \overline{\text{box}}(F') = \text{bin}(F'') \). Thus \( F'' \) is initial and so, by the induction hypothesis and the rules \( \text{ic}1 \) and \( \text{con}2 \), we obtain \( F = F' \text{ bin} F'' \equiv \boxed{F'} \text{ bin} \boxed{F''} \equiv \boxed{F'} \equiv \boxed{F} \).

A static expression \( E \) is tidy if each occurrence of the unary buffer stuffing operator \( \text{.b} \) is either one of the \( .b \)'s such that \( E = E'.b_1 \cdots b_k \), or it appears within a subexpression of the form \( E'.b_1 \cdots b_k \text{tie} b \). Moreover, a dynamic expression \( F \) is in tidy if \( \boxed{F} \) 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 \( F' = \boxed{\overline{\alpha}} \) or \( F' = F'' \text{ un}a \) or \( F' = F'' \text{ bin} F''' \), where \( k \geq 0 \) and \( \text{un}a \) is an instance of scoping or buffer restriction. Then \( M^{\text{op}}_{\text{box}}(F) = \{b_1, \ldots , b_k\} \) where each \( b_k \) identifies the unique open buffer place labelled by \( b_k \). Moreover, in the last two cases \( M^{\text{op}}_{\overline{\text{box}}}(F''') = \emptyset \), while in the last case \( M^{\text{op}}_{\overline{\text{box}}}(F''') \).

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

\( \square \)

Proposition 6.6. Let \( F_1 \) and \( F_2 \) be box expressions such that \( \boxed{F_1} = \boxed{F_2} \).

1. \( \text{box}(F_1) = \boxed{\text{box}(F_2)} \) iff \( F_1 \equiv F_2 \).

2. \( \boxed{\text{box}(F_1)} = \overline{\boxed{\text{box}(F_2)}} \) iff \( F_1 \equiv \overline{F}_2 \).

3. \( \boxed{\text{box}(F_1)} = \boxed{\boxed{\text{box}(F_2)}} \) iff \( F_1 \equiv F_2 \).
that each expressions can be obtained from the auxiliary transition based semantics, in the same way that the derived

induction 5.1(3), places of all the nets in the last equivalence are empty (and so

Thus, by proposition 4.4, \( \text{box}(F_1') = \text{box}(F_2') \). As a result, by the induction hypothesis, \( F_1^\prime \equiv F_2^\prime \) and so

since this auxiliary operational semantics is defined using the denotational one, it will be relatively easy

labels are multisets of the names of transitions in the denotational semantics of a given box expression.

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 \( \mathcal{T} \) of all transition trees in the boxes derived through the \( \text{box} \) mapping. It is easy to check
that each \( t \in \mathcal{T} \) has always the same label in all the boxes derived through the \( \text{box} \) 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 $U \in U \equiv \text{mult}([T])$. 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 $\rightarrow$ which is the least relation comprising all $(F, U, F') \in \text{aexpr} \times U \times \text{aexpr}$ such that the relations in table 2 hold (note that $F \xrightarrow{U} F'$ means that $(F, U, F')$ belongs to $\rightarrow$). In the definition of $\text{EOP}$ we make no restriction on $U_1$ and $U_2$, but the domain of application of $\text{bin}$ 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.

<table>
<thead>
<tr>
<th>EA1</th>
<th>$\alpha \xrightarrow{{a}} a$</th>
<th>EA2</th>
<th>$ab^+ \xrightarrow{{a^b+b}} ab^+b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA3</td>
<td>$a b^- b \xrightarrow{{a^b}} a b^-$</td>
<td>EA4</td>
<td>$a b^+ b \xrightarrow{{a^b}} a b^+ b$</td>
</tr>
<tr>
<td>EQ1</td>
<td>$F \xrightarrow{\varnothing} F$</td>
<td>EQ2</td>
<td>$F \equiv F', F' \xrightarrow{U} F''', F'' \equiv F'''$</td>
</tr>
<tr>
<td>EBUF</td>
<td>$F \xrightarrow{U} F'$</td>
<td>ETIE</td>
<td>$F \xrightarrow{U} F'$</td>
</tr>
<tr>
<td>ESC</td>
<td>$D \xrightarrow{{t_1, u_1, \ldots, t_k, u_k, z_1, \ldots, z_l}} D'$</td>
<td>$F, b \xrightarrow{U} F', b$</td>
<td></td>
</tr>
<tr>
<td>EOP</td>
<td>$F_1 \xrightarrow{U_1} F'_1 \land F_2 \xrightarrow{U_2} F'_2$</td>
<td>$F_1 \text{ bin } F_2 \xrightarrow{(U_1 \text{ bin } U_1) \cup (U_2 \text{ bin } U_2)} F'_1 \text{ bin } F'_2$</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. Transition based operational semantics for ABC, where $a \in \mathbb{A}$ (except in ESC, where $a \in \mathbb{A}$), $b \in \mathbb{B}$ and $\text{bin}$ 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:**

($\Rightarrow$) Consider the derivation tree $T$ 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 $T$ must refer to instances of the rule EQ1. We can then proceed by induction on the structure of $T$, 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).
(\iffalse)\ Follows from the rules \text{EQ1} and \text{EQ2}, with } F'' = F'''.\quad \square

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.

\textbf{Theorem 6.8.} Let $F$ be a box expression.

1. If $F \xrightarrow{U} F'$, then $F'$ is a box expression such that $\text{bbox}(F)[U]\text{bbox}(F')$ and $[F'] = [F']$.

2. If $\text{bbox}(F)[U] \Sigma$, then there is a box expression $F'$ such that $\text{bbox}(F') = \Sigma$ and $F \xrightarrow{U} F'$.

\textbf{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 \text{EQ1} and \text{EA1-4}, and the result holds due to proposition 5.5 and $\Sigma[\emptyset] \Sigma$, for every box $\Sigma$. 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 \text{EQ2} was applied, we use proposition 5.3(1); for a node where \text{EBUF} was applied, we use the translation rules (5); for a node where \text{ESC} or \text{ETIE} was applied, we use proposition 5.4(1,2) and the translation rules (5); and for a node where \text{EOP} was applied, we use proposition 5.3 and the translation rules (5).

(2) If $U = \emptyset$ then, due to $\Sigma = \text{bbox}(F)$, 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), \text{EA1}, \text{EA2}, \text{EA3} \text{ EA4}, \text{EBUF} \text{ EQ2}, \text{E1} and \text{B1}. In the inductive step ($h > 0$), due to the rules in table 1, one of the following cases holds.

Case 1: $F \equiv G \text{una}$, where \text{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 \text{ESC} and \text{ETIE}.

Case 2: $F \equiv G \text{tie} b_1 b_2 \cdots .b$. Then we first apply proposition 5.4(3) to conclude that $U$ is in fact an enabled step for $\text{bbox}(G \text{tie} b)$, and the result holds by the induction hypothesis, proposition 5.4(2) and the rules \text{ETIE} and \text{EBUF}.

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

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 $\xrightarrow{\{I_1,I_4\}}$ (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.
\[
\begin{align*}
&\frac{s; ((pr^+ \otimes f) \parallel (pr^+ \otimes f)) \parallel (cr^- \otimes f)) \text{ tie } r}{\not\exists; ((pr^+ \otimes f) \parallel (pr^+ \otimes f)) \parallel (cr^- \otimes f)) \text{ tie } r} \\
&\frac{\{t_1\}}{\not\exists; ((pr^+ \otimes f) \parallel (pr^+ \otimes f)) \parallel (cr^- \otimes f)) \text{ tie } r} \\
&\frac{\{t_2, t_3\}}{s; ((pr^+.r \otimes f) \parallel (pr^+.r \otimes f)) \parallel (cr^- \otimes f)) \text{ tie } r} \\
&\quad \quad \Rightarrow s; ((pr^+.r \otimes f) \parallel (pr^+.r \otimes f) \parallel (cr^- \otimes f).r) \text{ tie } r} \\
&\quad \quad \Rightarrow s; ((pr^+.r \otimes f) \parallel (pr^+.r \otimes f) \parallel (cr^- .r \otimes f)) \text{ tie } r} \\
&\quad \quad \Rightarrow s; ((pr^+.r \otimes f) \parallel (pr^+.r \otimes f) \parallel (cr^- .r \otimes f)) \text{ tie } r} \\
&\quad \quad \Rightarrow s; ((pr^+.r \otimes f) \parallel (pr^+.r \otimes f) \parallel (cr^- .r \otimes f)) \text{ tie } r} \\
&\quad \quad \Rightarrow \{t_2, t_4\} \\
&\frac{\{t_5, t_6, t_7\}}{s; ((pr^+.r \otimes f) \parallel (pr^+.r \otimes f) \parallel (cr^- \otimes f)) \text{ tie } r} \\
&\quad \quad \Rightarrow s; ((pr^+.r \otimes f) \parallel (pr^+.r \otimes f) \parallel (cr^- \otimes f)) \text{ tie } r}
\end{align*}
\]

\[
\frac{\not\exists;r \otimes f \quad \{v^p_{\text{pr}} < v^p \} \quad \not\exists;r .r \otimes f \quad \{v^p_{\text{pr}} < v^p \}}{pr^+.r \otimes f \quad \{v^p_{\text{pr}} < v^p \} \quad pr^+.r .r \otimes f \quad \{v^p_{\text{pr}} < v^p \}}
\]

\[
\frac{\not\exists;r \otimes f \quad \{v^p_{\text{pr}} < v^p \} \quad \not\exists;r .r \otimes f \quad \{v^p_{\text{pr}} < v^p \}}{pr^+.r \otimes f \quad \{v^p_{\text{pr}} < v^p \} \quad pr^+.r .r \otimes f \quad \{v^p_{\text{pr}} < v^p \}}
\]

\[
\frac{\not\exists;r \otimes f \quad \{v^p_{\text{pr}} < v^p \} \quad \not\exists;r .r \otimes f \quad \{v^p_{\text{pr}} < v^p \}}{pr^+.r \otimes f \quad \{v^p_{\text{pr}} < v^p \} \quad pr^+.r .r \otimes f \quad \{v^p_{\text{pr}} < v^p \}}
\]

Table 3. Execution scenario I and part of the derivation tree for the move labelled by \(\{t_2, t_4\}\). The identities of various transitions are as follows: \(t_1 = v_1 < v^p \), \(t_2 = v_2 < v^p \), \(t_3 = v_3 < v^p \), \(t_4 = v_4 < v^p \), \(t_5 = v_5 < v^p \), \(t_6 = v_6 < v^p \), \(t_7 = v_7 < v^p \).
Let $D$ be a dynamic box expression. We will use $[D]$ to denote all the box expressions derivable from $D$, i.e., the least set of expressions containing $D$ such that if $D' \in [D]$ and $D' \xrightarrow{U} D''$, for some $U \in \mathbb{U}$, then $D'' \in [D]$. Moreover, $[D]_\equiv$ will denote the equivalence class of $\equiv$ containing $D$. The full transition system of $D$ is $\text{fts}_D \overset{\equiv}{=} (V, L, A, \text{init})$, where $V \overset{\equiv}{=} \{[D']_\equiv \mid D' \in [D]\}$ is the set of states; $L \overset{\equiv}{=} \mathbb{U}$ is the set of arc labels; $A \overset{\equiv}{=} \{([D']_\equiv, U, [D''_\equiv]_\equiv) \in V \times \mathbb{U} \times V \mid D' \xrightarrow{U} D''\}$ is the set of arcs; and $\text{init} \overset{\equiv}{=} [D]_\equiv$ is the initial state. For a static box expression $E$, $\text{fts}_E \overset{\equiv}{=} \text{fts}_F$.

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 $D \xrightarrow{\equiv} D'$ for two equivalent but different expressions $D$ and $D'$, whereas in the domain of boxes, $\Sigma(\emptyset) \Theta$ 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 $F$, $\text{iso}_F \overset{\equiv}{=} \{(\text{box}(F') \mid [F']_\equiv \text{ is a node of } \text{fts}_F\}$ is an isomorphism between the full transition systems $\text{fts}_F$ and $\text{fts}_{\text{box}(F)}$. Moreover, $\text{iso}_F$ preserves the property of being in an initial or final state.\(^{10}\)

**Proof:**
Follows from theorem 6.8 (implying, in particular, that for each $D' \in [D]$, we have $[D'] \overset{\equiv}{=} [D]_\equiv$), propositions 6.6(1) and 6.4, and the translation rules (5). \hfill \Box

**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 $\Gamma$ is a finite multiset in $\mathbb{L} \overset{\equiv}{=} \text{mult}(\text{arc})$, as shown in table 4. (Formally, we define a ternary relation $\longrightarrow$ which is the least relation comprising all $(F, \Gamma, F') \in \text{expr} \times \text{mult}((\text{arc}) \times \text{expr}$ such that the relations in table 4 hold, and $F \xrightarrow{\Gamma} F'$ means that $(F, \Gamma, F')$ 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. \hfill \Box

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 $\text{ltt}_F$, and defined as $\text{fts}_F$ 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.
Table 4. Label based operational semantics for ABC, where $a \in A_r$ (except in LSC, where $a \in A_l$), $b \in B$ and bin stands for any binary ABC operator.

**Theorem 6.11.** For every box expression $F$, $\text{iso}_F \overset{\text{df}}{=} \{ [F']_\equiv \backslash \text{box}(F') \mid [F']_\equiv$ is a node of $\text{lt} F \}$ is an isomorphism between the labelled transition systems $\text{lt} F$ and $\text{lt}_{\text{box}(F)}$. Moreover, $\text{iso}_F$ 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:

$$ (pr^+; ((pr^+ \otimes f) \parallel (c\tau^- \otimes f)) \text{tie}_r; c\tau^- \cdot r)) \text{tie}_r. $$

Notice that if we were able to move the $\cdot r$ from the end of the expression towards $c\tau^-$, resulting in $c\tau^- \cdot r$, then we would have been able to execute $c\tau^-$. However, the B1 rule has been designed so that it is impossible to move $\cdot r$ 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.
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 \([,] \) 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

\[
\begin{align*}
(p^r \cdot ((p^r \otimes f) \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & \\
\equiv & \ (p^r \cdot ((p^r \otimes f) \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & E1, IS1 \\
\{[p]\} & (p^r \cdot ([p] \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & LA1, LOP, LTIE \\
\equiv & \ (p^r \cdot ([p] \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & X1, OPL-R \\
\equiv & \ (p^r \cdot ([p] \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & IS1-2, E1 \\
\equiv & \ (p^r \cdot ([p] \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & IPAR1, ITI1 \\
\{[p]\} & (p^r \cdot (p^r \cdot r \otimes f) \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & LA1, LOP, LTIE \\
\equiv & \ (p^r \cdot (p^r \cdot r \otimes f) \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & IIT2, E1 \\
\{[c]\} & (p^r \cdot (p^r \cdot r \otimes f) \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & LA1, LOP, LTIE \\
\equiv & \ (p^r \cdot (p^r \cdot r \otimes f) \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & OPL-R, E1 \\
\{[c]\} & (p^r \cdot (p^r \cdot r \otimes f) \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & LA3, LOP, LTIE \\
\equiv & \ (p^r \cdot (p^r \cdot r \otimes f) \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & IIT3 \\
\{[c]\} & (p^r \cdot (p^r \cdot r \otimes f) \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & LA1, LOP, LTIE \\
\equiv & \ (p^r \cdot (p^r \cdot r \otimes f) \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & IIT5, IPAR2, X1, IS2 \\
\{[c]\} & (p^r \cdot (p^r \cdot r \otimes f) \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & LA3, LOP, LTIE \\
\equiv & \ (p^r \cdot (p^r \cdot r \otimes f) \parallel (c^r \otimes f)) \uparrow r; c^r) \uparrow r & IS3, X1
\end{align*}
\]

Table 5. Execution scenario II.
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 bisimulations 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 operator was less standard and its handling required some care. Finally, the rules \( E_{\text{OP}} \) in table 2 and \( L_{\text{OP}} \) 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 \( \pi \)-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


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, $\Omega_{\sqcap}, \Omega_{\oplus}$ and $\Omega_1$ will be considered as instances of a more general class of operator boxes. A sequential operator box $\Omega_{\sqcap}$ 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 $v \in T_{\Omega_{\sqcap}}, |v| = |v^*| = 1$ and $\lambda_{\Omega_{\sqcap}}(v) = \varphi_{id}$. That is, $\Omega_{\sqcap}$ 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 $\Omega_{\sqcap}$ is the set $\text{dom}_{\Omega_{\sqcap}}$ comprising all $\Omega_{\sqcap}$-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 $\Omega_{\sqcap}$ 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 $\Omega_{\varphi}$, shown on the right of figure 11, is parameterised by an interface function $\varphi : \text{mult}(A_\varphi) \setminus \{\emptyset\} \rightarrow A_\varphi$, and has the domain of application $\text{dom}_{\Omega_{\varphi}} = \text{abox}_{\text{stic}} \cup \text{abox}_{\text{dyn}}$. To ensure the finiteness of the nets created, we assume that there is no finite set of labels $A \subseteq A_\varphi$ such that $\text{mult}(A) \cap \text{dom}_\varphi$ is infinite; it may be checked that $\varphi_{\text{stic}}$ fulfills this constraint. The role of $\Omega_{\varphi}$ will be to effect the change of synchronous communication interface specified by $\varphi$.

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

- The set of transitions of $\Phi$ is the set of all trees $v_i < t$ 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 $p^b = (p^b_{v_1}, \ldots, p^b_{v_n}) \in S_{\Omega(\Sigma)}$, where each $p^b_{v_i}$ is the unique $b$-labelled place of $\Sigma_i$. 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 $\Omega_\varphi$, the intuition behind a multiset $\Gamma$ in the domain of $\varphi$ 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 $\Gamma$. 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 $\varphi_{id}$, no transition label is changed for them.) Hence, the application of a communication interface operator $\Omega_\varphi$ to a box $\Sigma$ results in a labelled net which is like $\Sigma$ with the only difference that the set of transitions comprises all trees $t = v^\varphi < \{t_1, \ldots, t_l\}$ such that $\{t_1, \ldots, t_l\} \in \text{mult}(T_\Sigma)$ and the multiset $\Lambda \overset{df}{=} \{\lambda_\Sigma(t_1), \ldots, \lambda_\Sigma(t_l)\}$ belongs to the domain of $\varphi$. The label of $t$ is $\varphi(\Lambda)$, and for a place $p$ of $\Omega_\varphi(\Sigma)$, which is also a place of $\Sigma$, the weight function is given by:

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

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 $\Omega$ be any sequential operator box or the parallel composition operator box $\Omega_\parallel$, $\Sigma$ be any $\Omega$-tuple of boxes, and $\Sigma$ be any box.
1. \( M_{\Omega(\Sigma)}^{\text{ctr}} = \emptyset \) iff \( M_{\Sigma_v}^{\text{ctr}} = \emptyset \), for each \( v_i \in T_\Omega \).
   Note: This property does not rely on the fact that \( \Sigma \) belongs to \( \text{dom}_\Omega \), and it would hold for any operator based on net substitution.

2. If \( \Sigma' \) is \( \Omega_\omega(\Sigma) \) or \( \Sigma \) tied \( b \) or \( \Sigma.B \), then \( M_{\Sigma'}^{\text{ctr}} = \emptyset \) iff \( M_{\Sigma'}^{\text{ctr}} = \emptyset \).

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

(2) Follows immediately from the fact that the control places of \( \Sigma \) and their markings remain unchanged. \( \square \)

Lemma A.2. If \( M \) is a clean marking of a box \( \Sigma \) such that \( M(e) + M(x) \geq 1 \), for all \( e \in \Sigma \) and \( x \in \Sigma \), then \( M^{\text{ctr}} \in \{ \Sigma, \Sigma^0 \} \).

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

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

1. If there is a place in \( \mathbb{P}(s'' \text{ in } M) \) which is marked at \( M \) then \( s'' \in \{s, s''\} \).

2. If \( M^{\text{ctr}} = \Sigma \) then \( M^{\text{ctr}} = \mathbb{P}(s) \), and if \( M^{\text{ctr}} = \Sigma^0 \) then \( M^{\text{ctr}} = \mathbb{P}(s') \).

3. If \( M^{\text{ctr}} \geq \mathbb{P}(s'') \) then \( M^{\text{ctr}} = \mathbb{P}(s'') \), and one of the following holds:
   (a) \( s'' = s \neq s' \) and \( M^{\text{ctr}} = \Sigma \).
   (b) \( s'' = s' \neq s \) and \( M^{\text{ctr}} = \Sigma^0 \).
   (c) \( s'' = s = s' \) and \( M^{\text{ctr}} \in \{ \Sigma, \Sigma^0 \} \).

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

(2) Suppose that \( M^{\text{ctr}} = \Sigma \) (the case \( M^{\text{ctr}} = \Sigma^0 \) is symmetric). Then the control places marked in \( \Omega_{sq}(\Sigma) \) are exactly those in whose construction the places from \( \Sigma \) were used. Moreover, \( M(q) = M_{\Sigma}(e) \), for all \( e \in \Sigma \) and \( q \in \mathbb{P}(s,e) \). Hence \( M^{\text{ctr}} = \mathbb{P}(s) \).

(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 \( e \in \Sigma \) and \( q \in \mathbb{P}(s'',e) \). Hence \( M_{\Sigma} \geq \Sigma \), and so \( M_{\Sigma}^{\text{ctr}} = \Sigma \) since \( M_{\Sigma} \) is clean. Moreover, by part (2), \( M^{\text{ctr}} = \mathbb{P}(s'') \).

Suppose now that \( s'' = s = s' \). Then \( 1 \leq M(q) = M_{\Sigma}(e) + M_{\Sigma}(x) \), for all \( e \in \Sigma, x \in \Sigma^0 \) and \( q \in \mathbb{P}(s'', e, x) \). Hence, by lemma A.2, \( M_{\Sigma}^{\text{ctr}} \in \{ \Sigma, \Sigma^0 \} \). Moreover, by part (2), \( M^{\text{ctr}} = \mathbb{P}(s'') \). \( \square \)
Theorem A.4. Let $\Omega$ be any general operator box and $\Sigma \in \text{dom}_{\Omega}$. Then $\Omega(\Sigma)$ is a box with a clean and ac-free marking. Moreover, if all the dynamic boxes (if any) in $\Sigma$ have quasi-safe markings, then the marking of $\Omega(\Sigma)$ is also quasi-safe.

Proof:

The result is straightforward for operators other than the sequential ones, so suppose that $\Omega$ is such an operator. Then $\Omega(\Sigma)$ is ex-restricted since $\mathbb{P}(s) \neq \emptyset$ for every $s \in S_{\Omega}$ (due to the ex-restrictedness of the boxes in $\Sigma$) and, in particular, for $s \in \sigma_{\Omega} \cup \sigma^0_{\Omega}$. Moreover, the $\mathbb{B}$-restrictedness follows directly from definitions, and the control-restrictedness from the same property of the boxes in $\Sigma$, and the fact that $\mathbb{P}(s,p) \neq \emptyset$ whenever $s \in S_{\Omega}$ and $p \in \sigma^0_{\Omega}$ for $v \in s^*$ (or $p \in \sigma^0_{\Omega}$ for $v \in s^*$). That $M_{\Omega(\Sigma)}$ is clean follows from lemma A.3(3), with $\{s\} = \sigma_{\Omega}$ or $\{s\} = \sigma^0_{\Omega}$. 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 $\Sigma$ are static boxes then, by definition, all the control places of $\Omega(\Sigma)$ are unmarked and the property follows immediately, from the control-restrictedness of $\Omega(\Sigma)$.

So, let us assume that $\Sigma_w$ is the unique dynamic box in $\Sigma$ and take a transition $v \in t$ of $\Omega(\Sigma)$. We distinguish two cases:

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

- If $s \notin *w \cup w^*$ then, by the definition of $\Omega(\Sigma)$, for every $q \in \mathbb{P}(s,p)$ we have $M_{\Omega(\Sigma)}(q) = 0$.
- If $w \in s^* \setminus s^*$ then, due to the cleaness of $\Sigma_w$, either $M_{\Sigma_w}^c = \sigma_{\Sigma_w}$ or there is $x \in \sigma_{\Sigma_w}$ such that $M_{\Sigma_w}(x) = 0$. In the former case, for every $q \in \mathbb{P}(s,p)$, we have $M_{\Omega(\Sigma)}(q) = 1$. In the latter case, for every $q \in \mathbb{P}(s,p,x)$, we have $M_{\Omega(\Sigma)}(q) = 0$.
- If $w \in s^* \setminus s^*$ then, due to the cleaness of $\Sigma_w$, either $M_{\Sigma_w}^c = \sigma_{\Sigma_w}$, or there is $e \in \sigma_{\Sigma_w}$ such that $M_{\Sigma_w}(e) = 0$. In the former case, for every $q \in \mathbb{P}(s,p)$, we have $M_{\Omega(\Sigma)}(q) = 1$. In the latter case, for every $q \in \mathbb{P}(s,p,e,x)$, we have $M_{\Omega(\Sigma)}(q) = 0$.

Case 2: $v = w$. Since $\Sigma_w$ is an ac-free labelled net, there is a control place $p$ of $\Sigma_w$ 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 $p \in \mathcal{E}$. Besides, if $\Sigma_w$ 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 $\Omega(\Sigma)$, $p \in \mathcal{E}(w \in t)$ and
$$M_{\Omega(\Sigma)}(p) = M_{\Sigma_w}(p) < 2 \cdot W_{\Sigma_w}(p,t) = 2 \cdot W_{\Omega(\Sigma)}(p,w \in t).$$

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

If $p \in \sigma_{\Sigma_w}$ (the case $p \in \sigma^0_{\Sigma_w}$ is symmetric), let $s$ be the place in $\Omega$ such that $\{s\} = \mathcal{E}$. We then observe that the following hold:

- If $w^* \neq \{s\}$ then, for every $q \in \mathbb{P}(s,p)$ we have
  $$M_{\Omega(\Sigma)}(q) = M_{\Sigma_w}(p) < 2 \cdot W_{\Sigma_w}(p,t) = 2 \cdot W_{\Omega(\Sigma)}(q,w \in t).$$

  Moreover, $M_{\Omega(\Sigma)}(q) = M_{\Sigma_w}(p) \leq 1$ in the quasi-safe case.
• If \( \bullet w = \{ s \} = w^* \) then, due to the cleanness of \( \Sigma_w \), either \( M_{\Sigma_w}^{cr} = \Sigma_w^c \), or there is \( x \in \Sigma_w^c \) such that \( M_{\Sigma_w}(x) = 0 \). In the former case, for every \( q \in \mathbb{P}\langle s, p \rangle \) we have \( M_{\Omega(\Sigma)}(q) = 1 \). In the latter case, for every \( q \in \mathbb{P}\langle s, p, x \rangle \), we have

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

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

\[\Box\]

**Proposition A.5.** Let \( \Omega \) be a sequential operator box, and \( \Sigma \) be an \( \Omega \)-tuple of static boxes.

1. If \( v \in T_{\Omega} \) is such that \( ^*\Omega = \bullet v \) or \( ^*\Omega = \nu^* \), then \( \Omega(\Sigma) = \Omega(\Sigma') \), where \( \Sigma' \) is \( \Sigma \) with \( \Sigma_v \) replaced respectively by \( \Sigma_{\nu} \) or \( \Sigma_v \).

2. If \( v \in T_{\Omega} \) is such that \( ^\circ\Omega = \bullet v \) or \( ^\circ\Omega = \nu^* \), then \( \Omega(\Sigma) = \Omega(\Sigma') \), where \( \Sigma' \) is \( \Sigma \) with \( \Sigma_v \) replaced respectively by \( \Sigma_{\nu} \) or \( \Sigma_v \).

**Proof:**
Follows from the definitions. 

\[\Box\]

**A.2. Structural equivalence**

For a general operator box \( \Omega \) 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 \( \Omega \) be a general operator box, and \( \Sigma \in dom_{\Omega} \).

1. If \( \Sigma \equiv_\Omega \Theta \), then \( \Theta \in dom_{\Omega} \) and \( [\Sigma] = [\Theta] \). Moreover, if all the boxes in \( \Sigma \) are quasi-safe then so are all the boxes in \( \Theta \).

2. \( \equiv_\Omega \) is an equivalence relation on \( dom_{\Omega} \).

3. If \( \Theta \in dom_{\Omega} \) and \( [\Sigma] = [\Theta] \), then \( \Omega(\Sigma) = \Omega(\Theta) \) iff \( \Sigma \equiv_\Omega \Theta \).

**Proof:**
(1) The result is obvious for all unary operator boxes. For the parallel operator box, it follows from the observation that \( \equiv_{\Omega} \equiv_{\Omega}^1 \equiv_{\Omega}^5 \) 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 \( \Omega \), it suffices to observe that the relations \( \equiv_{\Omega}^i \) only act on the markings of the components and preserve the number of static and dynamic boxes in \( \Sigma \). Indeed, in the case of \( \equiv_{\Omega}^5 \) this follows from proposition 4.3(5,6), and in the remaining cases can be shown by case analysis. Take, for instance, the relation \( \equiv_{\Omega}^1 \) with \( (\Sigma_{\nu}, \Sigma_w) = (\Psi_{\nu}, \Psi_w) \) and \( (\Theta_{\nu}, \Theta_w) = (\Psi_{\nu}, \Psi_w) \). Then, by the definition of \( dom_{\Omega} \), \( (\Sigma_{\nu}, \Sigma_w) \in abox_{dyn} \times abox_{dsc} \). Thus, by proposition 4.3(7), we have \( \Psi_{\nu} \in abox_{dsc} \), and so \( (\Theta_{\nu}, \Theta_w) \in abox_{dsc} \times abox_{dyn} \). Hence \( \Theta \in dom_{\Omega} \).

The second part follows from the definition of a quasi-safe box, and the observation that the boxes in \( \Theta \) have clean markings (since \( \Theta \in dom_{\Omega} \)).
(2) The result is again obvious for all unary operator boxes. For the parallel operator box, it is easy to check that \( \equiv_{\Omega} \equiv_5 \equiv_5 \) is reflexive, symmetric and transitive.

Next let us consider any sequential operator \( \Omega \). That \( \equiv_\Omega \) is reflexive follows from \( \text{id}_{\text{abox}} \subseteq (\equiv_{\Omega} \circ \equiv_{\Omega}^4) \). Moreover, it is easy to see that each \( \equiv_{\Omega}^i \) is symmetric and \( (\equiv_{\Omega}^5 \circ \equiv_{\Omega}^i) = (\equiv_{\Omega}^5 \circ \equiv_{\Omega}^5) \), for \( 0 \leq i \leq 4 \). Hence \( \equiv_{\Omega} \) is also symmetric. That it is also transitive on \( \text{dom}_{\Omega} \) 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 \( \text{dom}_{\Omega} \):

\[
(\equiv_{\Omega}^5 \circ \equiv_{\Omega}^j \circ \equiv_{\Omega}^5 \circ \equiv_{\Omega}^k) = (\equiv_{\Omega}^5 \circ \equiv_{\Omega}^5 \circ \equiv_{\Omega}^j \circ \equiv_{\Omega}^k) = (\equiv_{\Omega}^5 \circ \equiv_{\Omega}^j \circ \equiv_{\Omega}^k)
\]

\[
(\equiv_{\Omega}^j \circ \equiv_{\Omega}^k) \subseteq \bigcup_{i=0}^{4} \equiv_{\Omega}^i.
\]

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

\[
(\equiv_{\Omega}^1 \circ \equiv_{\Omega}^0) \subseteq \equiv_{\Omega}^3 \quad (\equiv_{\Omega}^1 \circ \equiv_{\Omega}^4) \subseteq (\equiv_{\Omega}^1 \cup \equiv_{\Omega}^4) \quad (\equiv_{\Omega}^1 \circ \equiv_{\Omega}^2) = \emptyset
\]

(3) Since \( [\Sigma] = [\Theta] \), \( \Sigma \) and \( \Theta \) 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 \( \Sigma \) and \( \Theta \), 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 \( \equiv_{\Omega} \).

Finally, we consider the case of a sequential operator box \( \Omega \). Then \( \Omega(\Sigma) = \Omega(\Theta) \) since \( [\Sigma] = [\Theta] \). Moreover, by the definition of net substitution,

\[
M_{\Omega(\Sigma)}^{\text{op}} = M_{\Omega(\Theta)}^{\text{op}} \iff \sum_{v \in \text{dom}_{\Omega}} M_{\Sigma_v}^{\text{op}} = \sum_{v \in \text{dom}_{\Omega}} M_{\Theta_v}^{\text{op}},
\]

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 \( \equiv_{\Omega} \). Now, the closed buffer places (with their marking) are the same in \( \Sigma \) and \( \Omega(\Sigma) \), as well as in \( \Theta \) and \( \Omega(\Theta) \), and they are not influenced by \( \equiv_{\Omega} \), so that we only need to consider the control places.

We first consider the \( (\iff) \) direction of the property.

If all the boxes in \( \Sigma \) are static, then \( \Sigma \equiv_{\Omega} \Theta \) and we clearly have \( \Omega(\Sigma) = \Omega(\Theta) \). Next we prove that for \( 0 \leq i \leq 4 \), if \( \Sigma \equiv_{\Omega}^i \Theta \) then we have also \( M_{\Omega(\Sigma)}^{\text{ctr}} = M_{\Omega(\Theta)}^{\text{ctr}} \). We consider the following three cases (the proof for \( i \in \{2, 3\} \) is similar to that for \( i = 1 \)):

- \( \Sigma \equiv_{\Omega}^0 \Theta \). Suppose that \( v \) and \( \Psi \) are as in the definition of \( \equiv_{\Omega} \). Without loss of generality, we may assume that \( \Sigma_v = \Psi \) and \( \Theta_v = \Psi \). Then by the cleanness of \( \Sigma_v \) and \( \Theta_v \), we have \( M_{\Sigma_v}^{\text{ctr}} = \Psi_{\Sigma_v} \) and \( M_{\Theta_v}^{\text{ctr}} = \Theta_v^{\text{ctr}} \). Hence, by lemma A.3(2), \( M_{\Omega(\Sigma)}^{\text{ctr}} = \Psi(s) = M_{\Omega(\Theta)}^{\text{ctr}} \), where \( \{s\} = v = v^* \).

This completes the proof.
Proposition A.7. Let \( \Omega \) be an \( n \)-ary sequential operator box and \( \Sigma \in \text{dom}_\Omega \).

1. If \( \Sigma \models \mathcal{H} \) (for \( i \leq n \)), then \( \Theta \in \text{dom}_\Omega \), and

\[
\Omega(\Sigma) \models \left( (v_1 \ll U_1) \cup \ldots \cup (v_n \ll U_n) \right) \Omega(\Theta) .
\]
2. If $\Omega(\Sigma)[U] \Theta$, then there are $\Psi, \Phi \in dom_\Omega$ and steps $U_1, \ldots, U_n$ such that $\Sigma \equiv_\Omega \Psi, \Psi_i [U_i] \Phi_i$ (for $i \leq n$), $\Theta = \Omega(\Phi)$ and

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

Note I: As a consequence, $\text{enabled}(\Omega(\Sigma))$ comprises exactly all sets $(v_1 \ll U_1) \cup \ldots \cup (v_n \ll U_n)$ of transitions such that there is $\Psi \equiv_\Omega \Sigma$ and $U_i \in \text{enabled}(\Psi_i)$ (for $i \leq n$).

Note II: For the sequential operators, at most one $U_i$ is non-empty.

**Proof:**

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

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

- For every internal or closed buffer place $p$ in $\Sigma_i$, we have:

$$M_{\Omega(\Sigma)}(p) = M_{\Sigma_i}(p) \geq \sum_{t \in U_i} W_{\Sigma_i}(p, t) = \sum_{t \in U_i} W_{\Omega(\Sigma)}(p, v_i \ll t) \quad \text{and} \quad \sum_{t \in U_i} W_{\Sigma_i}(t, p) = \sum_{t \in U_i} W_{\Omega(\Sigma)}(v_i \ll t, p).$$

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

$$M_{\Omega(\Sigma)}(b) \geq M_{\Sigma_i}(b) \geq \sum_{t \in U_i} W_{\Sigma_i}(b, t) = \sum_{t \in U_i} W_{\Omega(\Sigma)}(b, v_i \ll t) \quad \text{and} \quad \sum_{t \in U_i} W_{\Sigma_i}(t, b) = \sum_{t \in U_i} W_{\Omega(\Sigma)}(v_i \ll t, b).$$

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

$$M_{\Omega(\Sigma)}(q) = \sum_{p \in P} M_{\Sigma_i}(p) \geq \sum_{p \in P} \sum_{t \in U_i} W_{\Sigma_i}(p, t) = \sum_{t \in U_i} W_{\Omega(\Sigma)}(q, v_i \ll t) \quad \text{and} \quad \sum_{p \in P} \sum_{t \in U_i} W_{\Sigma_i}(t, p) = \sum_{t \in U_i} W_{\Omega(\Sigma)}(v_i \ll t, q).$$

Hence $\Omega(\Sigma)[(v_1 \ll U_1) \cup \ldots \cup (v_n \ll U_n)] \Omega(\Theta)$.

(2) If $\Omega(\Sigma)[U] \Theta$, then there exists $n$ steps $U_1, \ldots, U_n$ such that $U = (v_1 \ll U_1) \cup \ldots \cup (v_n \ll U_n)$. If $U = \emptyset$, then each $U_i$ is empty too, and the property obviously holds, with $\Psi = \Phi = \Sigma$.

This will be the case, in particular, if $\Sigma$ are static boxes. Otherwise, we will assume that $\Sigma_i$ is the only dynamic box in $\Sigma$ and consider two cases:

Case 1: $M^{\text{stp}}_{\Sigma} \notin \{v_1 \ll \Sigma_i, \Sigma_i \}$. By the cleanness of $\Sigma_i$, there is $e \in \Sigma_i$ and $x \in \Sigma_i^0$ 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 \( \Psi \), we have that \( q \in \Psi(v_j < t) \) and \( M_{\Omega(\Sigma)}(q) = M_{\Sigma_i}(q) = 0 \).

If \( p \) is an entry place in \( \Psi \) and \( v_j = \{ s \} \) (or \( p \) is an exit place in \( \Psi \) and \( v_j^* = \{ s \} \)), for every

\[
\begin{align*}
q \in \mathbb{P}(s, p) & \quad \text{if } v_i \neq \{ s \} \neq v_i^*, & q \in \mathbb{P}(s, p, e) & \quad \text{if } v_i = \{ s \} \neq v_i^*, \\
q \in \mathbb{P}(s, p, x) & \quad \text{if } v_i \neq \{ s \} = v_i^*, & q \in \mathbb{P}(s, p, e, x) & \quad \text{if } v_i = \{ s \} = v_i^*,
\end{align*}
\]

we have that \( q \in \Psi(v_j < t) \) and \( M_{\Omega(\Sigma)}(q) = 0 \).

Hence \( U_j = \emptyset \), for all \( j \neq i \).

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

- For every internal or closed buffer place \( p \) in \( S_{\Psi_i} \), we have:

\[
M_{\Psi_i}(p) = M_{\Sigma_i}(p) = M_{\Omega(\Sigma)}(p) \geq \sum_{t \in U_i} W_{\Omega(\Sigma)}(p, v_i < t) = \sum_{t \in U_i} W_{\Psi_i}(p, t) .
\]

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

\[
M_{\Psi_i}(b) = M_{\Omega(\Sigma)}(b) \geq \sum_{t \in U_i} W_{\Omega(\Sigma)}(b, v_i < t) = \sum_{t \in U_i} W_{\Psi_i}(b, t) .
\]

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

\[
M_{\Psi_i}(p) = M_{\Sigma_i}(p) = M_{\Omega(\Sigma)}(q) \geq \sum_{t \in U_i} W_{\Omega(\Sigma)}(q, v_i < t) = \sum_{t \in U_i} W_{\Psi_i}(p, t) ;
\]

and for every \( q \in \mathbb{P}(s, p, x) \) if \( v_i = \{ s \} = v_i^* \), we have:

\[
M_{\Psi_i}(p) = M_{\Sigma_i}(p) = M_{\Omega(\Sigma)}(q) \geq \sum_{t \in U_i} W_{\Omega(\Sigma)}(q, v_i < t) = \sum_{t \in U_i} (W_{\Psi_i}(p, t) + W_{\Psi_i}(x, t)) \geq \sum_{t \in U_i} W_{\Psi_i}(p, t) .
\]

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

Case 2: \( M_{\Sigma_i} = \tau \Sigma_i \) (the case \( M_{\Sigma_i} = \Sigma_i \) is similar). Suppose that \( v_i = \{ s \} \). Then, by lemma A.3(2), \( M_{\Omega(\Sigma)} = \mathbb{P}(s) \). Hence, for every \( (v < t) \in U \), we have the following: \( \Psi \) does not contain any internal places; if \( \Psi \cap \tau \Sigma_v \neq \emptyset \) then \( v = \{ s \} \); and if \( \Psi \cap \tau \Sigma_v \neq \emptyset \) then \( v^* = \{ s \} \).

Suppose now that \( v < t, v' < t' \in U \), where \( v \neq v' \). Then assume that there is \( e \in \Psi \cap \tau \Sigma_v \) and \( e' \in \Psi' \cap \tau \Sigma_v \). We have, for every \( q \in \mathbb{P}(s, e, e') \), that

\[
1 = M_{\Omega(\Sigma)}(q) \geq W_{\Omega(\Sigma)}(q, v < t) + W_{\Omega(\Sigma)}(q, v' < t') = W_{\Sigma_v}(e, t) + W_{\Sigma_v}(e', t') \geq 2 .
\]

Hence we have obtained a contradiction (the situation is similar if \( e \) is replaced by an \( x \in \Psi \cap \tau \Sigma_v \) and/or \( e' \) is replaced by an \( x' \in \Psi' \cap \tau \Sigma_v \)). As a consequence, there is a unique non-empty \( U_j \). We then consider two cases:
Proposition A.8. Let $\Sigma$ be a static or dynamic box, and $\Omega_{\varphi}$ be a communication interface operator box.

• $j = i$. 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 $e \in \ell \cap \circ \Sigma_i$ and $x \in \ell' \cap \Sigma_i^0$ because either $v_i^* = \{s\} \neq \{s\}$ and for $q \in \mathbb{P}\langle s', x \rangle$ we have

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

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

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

Hence the control pre-places of the transitions in $U_i$ are either all in $\circ \Sigma_i$ or all in $\Sigma_i^0$. In the first case, we may not have $t \neq t' \in U_i$ with some $e \in \ell \cap \ell' \cap \circ \Sigma_i$ since then, for every $q \in \mathbb{P}\langle s, e \rangle$, we would have had:

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

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

$$M_{\Sigma_i}(e) = M_{\Omega_i}(q) \geq \sum_{t \in U_i} W_{\Omega_i}(q, v_i < t) = \sum_{t \in U_i} W_{\Sigma_i}(e, t) .$$

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

$$M_{\Sigma_i}(p) = M_{\Omega_i}(p) \geq \sum_{t \in U_i} W_{\Omega_i}(p, v_i < t) = \sum_{t \in U_i} W_{\Sigma_i}(p, t) .$$

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

$$M_{\Psi_i}(b) = M_{\Omega_i}(b) \geq \sum_{t \in U_i} W_{\Omega_i}(b, v_i < t) = \sum_{t \in U_i} W_{\Psi_i}(b, t) .$$

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

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

• $j \neq i$. Then either $n_j = \{s\}$ and we replace $\Sigma$ by $\Psi$ where the only modifications consist in replacing $M_{\Psi_i}^{cr}$ by the empty marking and $M_{\Psi_i}^{cr}$ by $\circ \Psi_j$ (see the definition of $\equiv^0_{\Omega}$), or $v_j^* = \{s\}$ and we replace $\Sigma$ by $\Psi$ where the only modifications consist in replacing $M_{\Psi_i}^{cr}$ by the empty marking and $M_{\Psi_i}^{cr}$ by $\Psi_j^0$ (see the definition of $\equiv^0_{\Omega}$), and we have the same situation as before with $\Sigma \equiv_{\Omega} \Psi$.

□
1. If $\Sigma [U_1 \uplus \ldots \uplus U_k] \Theta$ and each $\lambda_\Sigma(U_i)$ belongs to the domain of $\varphi$, then
\[
\Omega_\varphi(\Sigma) \{v^\varphi \triangleleft U_1, \ldots, v^\varphi \triangleleft U_k\} \Omega_\varphi(\Theta).
\]

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

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

**Proof:**

It can be simply checked applying the definitions. \qed