Asynchronous Box Calculus with multi-way communication
Cécile Bui Thanh, Hanna Klaudel, Franck Pommereau

To cite this version:
Cécile Bui Thanh, Hanna Klaudel, Franck Pommereau. Asynchronous Box Calculus with multi-way communication. [Research Report] LACL, Université Paris-Est/Créteil. 2008. hal-02310227

HAL Id: hal-02310227
https://hal.science/hal-02310227
Submitted on 10 Oct 2019

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 with Multi-way Communication

Bui Thanh, C. and Klaudel, H. and Pommereau, F.

Technical Report TR-2002-16

Laboratory of Algorithms, Complexity and Logic
University of Paris XII, Val-de-Marne
61, avenue du Général de Gaulle
F-94010 CRÉTEIL Cedex – FRANCE
Tel: +33 (0)1 45 17 16 47
Fax: +33 (0)1 45 17 66 01
Asynchronous Box Calculus with Multi-way Communication

Cécile Bui Thanh, Hanna Klaudel, and Franck Pommereau

LACL, Université Paris 12
61, avenue du général de Gaulle
94010 Créteil, France
{bui,klaudel,pommereau}@univ-paris12.fr

Abstract. The starting point of this paper is the Asynchronous Box Calculus (or ABC), a formalism suitable for modelling compositionally distributed systems using both synchronous and asynchronous communication (through actions and links respectively). ABC is composed of two semantically consistent models: an algebra of Petri nets and an associated algebra of process expressions whose constants and operators directly corresponds to the Petri net ones. In this paper, we extend the Petri net algebra of ABC by allowing the use of multisets of actions as well as that of multisets of links. The ABC process expressions and their associated structured operational semantics (SOS) are extended in the same way. The obtained framework, called MBC (for Multi-way communication Box Calculus), allows to express processes which could not be modelled with ABC; for instance, processes with multi-way synchronisation. As it was the case for ABC, the resulting algebra of expressions is consistent with the net algebra in the sense that an expression and the corresponding net generate isomorphic transition systems.

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

1 Introduction

This paper is set in the framework of the Petri Box Calculus (PBC [1]), which has been designed with the aim of allowing a compositional Petri net semantics of nondeterministic and concurrent programming languages [4]. It was later extended into a more generic Petri Net Algebra (PNA [2, 3]), and afterwards to an Asynchronous Box Algebra (ABC [5]) which introduced asynchronous communication links [6]. The ABC model is composed of an algebra of process expressions (box expressions) with a fully compositional translation into labelled Petri nets (boxes). Synchronous communications are modelled by actions and asynchronous communications by links. A basic ABC event can perform at most one synchronous communication and one asynchronous link. This may be seen as a limitation because it makes impossible, for instance, the multi-way synchronisations to be built compositionally. In this paper, we remove this restriction and introduce multisets of actions (called multiactions) for synchronous communications and multisets of links (called multilinks) for asynchronous ones. The resulting model is called Multi-way communication Box Calculus (MBC for short).

We will now introduce informally the aspects of this model which are needed for our extension.

1.1 An algebra of nets and process expressions

The variant of the ABC model relevant for this paper considers the following operators: sequence $E_1; E_2$ (the execution of the process $E_1$ is followed by that of $E_2$); choice $E_1 \sqcup E_2$ (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 $E_1^\ast$ ($E_1$ can be executed an arbitrary number of times and is followed by $E_2$); and buffer restriction $E \langle b \rangle$ (the buffer $b$ and the related asynchronous links become private to the expression $E$). The ABC operator of scoping $E \text{sc} a$ enforces all multi-way synchronisations (instead of binary
ones) between the sets of multiactions involving \(a\) or \(\widehat{a}\), but forbidding any independent execution of such actions.

A basic process expression is composed of a multiaction and a multilink where one or both of them may be empty. An example of a basic expression is be, for instance, \(\{a_r, \widehat{a}_c\}\{b_a^-, b_d^+\}\), where \(a_r\) and \(\widehat{a}_c\) are synchronous actions used for scoping while \(b_a^-\) and \(b_d^+\) are asynchronous links expressing the receiving of a resource from the buffer \(b_a\) and a sending to the buffer \(b_d\). Such an expression may be executed if it is in a context with sufficiently many resources in buffers in order to satisfy all the receiving links (in our case, we should have at least one resource in the buffer \(b_a\)). Its execution removes then a resource from the buffer \(b_a\), produces one in the buffer \(b_d\), and emits the multiaction \(\{a_r, \widehat{a}_c\}\). A basic net corresponding to such an expression has one transition labelled \(\{a_r, \widehat{a}_c\}\) connected to one input and one output place and to two buffer places labelled \(b_a\) and \(b_d\). The receiving link \(b_a^-\) is represented by an arc going from the buffer place \(b_a\) to the transition and the sending link \(b_d^+\) is represented by an arc from the transition to the buffer place labelled \(b_d\).

We illustrate these constructs using an example based on three process expressions:

\[
\begin{align*}
\text{EMPLOYEE} & \equiv \{(\widehat{a}_r, a_r)\} \otimes \{\widehat{a}_c\}\{b_a^-, b_d^+\}\text{tie}b_d, \\
\text{BOSS} & \equiv \{(a_r, \widehat{a}_y)\} : \{\{b_d^+\}\} \setminus \{a_r, \widehat{a}_n\}\{\} \otimes \{a_f\}\{\}, \\
\text{FRIEND} & \equiv \{a_r\}\{\} \otimes \{a_c\}\{\}
\end{align*}
\]

modelling an employee, his boss and his friend. The employee repetitively requests his boss for a pay rise, which is modelled by the synchronous action \(\widehat{a}_r\) in the first part of the iteration. At the same time, the employee receives encouragements from his friend, which is also modelled by the action \(\widehat{a}_r\). Notice that this first step needs a three-way communication, which was impossible to model compositionally in ABC. The boss may answer the request by yes or no which is modelled by the actions \(\widehat{a}_y\) and \(\widehat{a}_n\), respectively. If he accepts, then an amend to the contract is sent to the employee, which is modelled by the sending link \(b_d^+\). If not, the employee continues asking until the boss agrees or terminates, in which case the system is deadlocked. The employee receiving an amend to the contract (modelled by the receiving link \(b_a^-\)) keeps it with his personal documents (modelled by the receiving link \(b_a^-\)) and is congratulated by his friend (through the synchronous action \(\widehat{a}_c\)). Since the buffer \(b_d\) is used to keep private documents, it is encapsulated into EMPLOYEE thanks to the buffer restriction \(\text{tie}b_d\).

Assuming that the binary operators associate to the left, the system where these processes operate in parallel is:

\[\text{PAR} \equiv \text{FRIEND} \parallel \text{EMPLOYEE} \parallel \text{BOSS}.\]

However, this system does not allow yet for synchronous communication between the processes (through the synchronous communication actions \(a_r\) and \(\widehat{a}_r\), \(a_c\) and \(\widehat{a}_c\)). This may be achieved by applying the scoping w.r.t. \(a_r\) and \(a_c\), leading to the expression:

\[\text{PAYRISE} \equiv (\text{FRIEND} \parallel \text{EMPLOYEE} \parallel \text{BOSS}) \text{ sc } a_r \text{ sc } a_c.\]

The nets (called the boxes) corresponding to EMPLOYEE, BOSS and FRIEND are represented on the top of figure 1. The boxes of PAR and PAYRISE are presented in the middle of the same figure. They are all the translation of their process expressions (themselves called the box expressions).

Notice that, in a box, the places are labelled by their status (\(e\) for entry, \(x\) for exit, \(i\) for internal places, \(b\) for restricted buffer places and the name of the buffer for each unrestricted buffer place) while the transitions are labelled by multiactions. The multilinks present in the expression are represented in the box as arcs between a transition and the corresponding buffer places. Each binary operator merges the unrestricted buffer places having the same label making asynchronous communication effective. For instance, the two buffer places labelled by \(b_d\) coming from EMPLOYEE and BOSS are merged when the parallel composition is applied, producing a box with only one \(b_d\)-labelled place. The application of the buffer restriction w.r.t. \(b_d\) changes the status of buffer place
Fig. 1. Boxes used in the example of the pay rise request. By convention, annotations inside the transitions represent the corresponding multi-actions, and the one outside next to the transitions (like in the box of PayRise) represent their names (identities).
$b_d$ to $\triangleright$ denoting that it is now a private place which cannot be merged anymore. The scoping w.r.t. $a_r$ and $a_c$ applied to the box of $\text{Par}$ produces the new transitions labelled $\overline{a_y}$ and $\overline{a_u}$ (visible in the box of $\text{PayRise}$) and removes from the net all the transitions involving $a_r$, $\overline{a_r}$, $a_c$ and $\overline{a_c}$ in their labels.

In this paper, synchronous actions are always denoted by the letter $a$ (possibly with some subscripts), asynchronous links are denoted by the letter $b$ (also with subscripts). Moreover, empty multiactions are omitted in the figures, as well as disconnected unmarked buffer places.

### 1.2 Structured operational semantics

The operational semantics of box expressions is given through SOS rules in Plotkin’s style [7]. However, instead of expressing the evolutions through rules modifying the structure of the expressions, like $a.E \xrightarrow{a} E$ in CCS, the idea here is to represent the current state of the evolution using overbars and underbars, corresponding respectively to the initial and final states of (sub)expressions. This is illustrated by the bottom left net of the figure 1, which represents the initial state of the system specified by $\text{PayRise}$, i.e., that corresponding to the box expression $\overline{\text{PayRise}}$.

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

$$\text{Friend} \| \text{Employee} \| \text{Boss} \equiv \text{Friend} \| \text{Employee} \| \text{Boss} \equiv \text{Friend} \| \text{Employee} \| \text{Boss},$$

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

$$\{a_r, a_y\} \triangleright \{b_u^+\} \xrightarrow{(\overline{a_r}, \overline{a_y})} \{b_u^+\}.$$

### 1.3 Execution examples

In order to provide more intuition about the way the MBC algebra is used, we describe in this section two system evolutions (or execution scenarios) of the system $\text{PayRise}$ presented in the previous section. We use labelled step sequences as a formal device for capturing concurrent behaviours, and we assume that the system starts from its implicit initial state. Thus, for example, we consider $\overline{\text{PayRise}}$ rather than $\text{PayRise}$.

**Scenario I.** Consider the box $\Sigma_0$ of $\overline{\text{PayRise}}$ shown in figure 1, and the following evolution:

- the employee, encouraged by his friend, requests for his pay rise but the boss refuses (this is modelled by the transition $t_1$);
- the employee, still encouraged, asks again and now the boss accepts (transition $t_2$);
- the boss sends the amend to the contract (transition $t_3$, which puts a token in the buffer place labelled by $b_u$);
- the employee receives the amend, puts it into his private documents and is also congratulated by his friend (transition $t_4$); simultaneously, the boss terminates (transition $t_5$).

Such a scenario corresponds to the step sequence $\{t_1\} \{t_2\} \{t_3\} \{t_4, t_5\}$ leading from $\Sigma_0$ to a net $\Sigma_1$, where $\Sigma_1$ is $\Sigma_0$ with one token in the buffer place $\triangleright$, one token in each of the $x$-labelled places, and no token elsewhere. This is denoted by:

$$\Sigma_0 \{\{t_1\}\} \{\{t_2\}\} \{\{t_3\}\} \{\{t_4, t_5\}\} \Sigma_1,$$

In terms of labelled step sequences, the scenario is represented by

$$\Sigma_0 \{\{\overline{a_u}\}\} \{\{\overline{a_y}\}\} \{\{\}\} \{\{a_f\}, \{\}\}\} \Sigma_1.$$

Since each $x$-labelled place holds exactly one token, we will say that $\Sigma_1$ is in a final marking (or final state).

---

1 A labelled step is a multiset of multiactions, while a step is here a set of transitions.
Scenario II. Consider now the following execution of $\Sigma_0$:

- the employee asks for his pay rise but the boss refuses (transition $t_1$);
- the boss terminates (transition $t_5$);

This scenario corresponds to $\Sigma_0[\{\{\overline{a}_n}\}\{\{a_f\}\}] \Sigma_2$, where $\Sigma_2$ is the box represented in figure 1. Notice that $\Sigma_2$ is not in a final marking. Moreover, the transition $t_4$, which comes from the synchronisation of the employee and his friend on action $a_e$ is now dead (it will never have a token in the $b_a$-labelled place in order to fire).

Notice that the same labelled step sequences may be obtained at the box expression level by the application of the operational rules (SOS) to $\text{PAYRISE}$.

1.4 About this paper

Multi-way communication is introduced through multi-actions (multisets of actions replacing the single actions of ABC) and multilinks (multisets of links while ABC has single links). Since one atomic event in MBC can perform arbitrarily many communications, the three kinds of constant expressions and basic boxes in ABC are replaced by infinite families of basic expressions and basic boxes representing all the possibilities. The structured operational semantics is modified in order to take all these changes into account; in particular, the rules for basic moves are replaced with a single general rule. All the proofs were carefully revised and it turned out that in all cases, they could be adapted to the new framework in a quite systematic and straightforward way. This is the reason why they are not included in the present paper.

To start with, section 2 describes the class of labelled nets on which MBC is based. In section 3, we introduce the net algebra part of MBC; in particular, we define a number of operators, either directly, or by using an auxiliary net substitution meta-operator. Section 4 investigates the relationship between the structure and the behaviour of composite nets. In section 5, we introduce an algebra of process expressions which forms the second part of the MBC. 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 in terms of their labels. We also present there our main result that a box expression and the corresponding box generate isomorphic transition systems.

2 Preliminaries

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

2.1 Multisets

A multiset over a set $X$ is a function $\mu : X \rightarrow \mathbb{N}$. 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 $\sum_{x \in X} \mu(x) \cdot x$ or $\sum_{x \in X} \mu(x) \cdot \{x\}$, as well as in extended set notation, e.g., $\{a_1, a_1, a_2\}$ denotes a multiset $\mu$ such that $\mu(a_1) = 2$, $\mu(a_2) = 1$ and $\mu(x) = 0$ for all $x \in X \setminus \{a_1, a_2\}$.

In this paper, in particular for steps or in the annotations of transitions, we may denote by $\{\}$ instead of $\emptyset$ empty multisets of actions or links.
2.2 Actions, buffers and synchronisation

We assume that there is a set $\mathcal{A}$ of actions representing synchronous interface activities used, in particular, to model handshake communication. We will also assume that, for every $a \in \mathcal{A}$, $\widehat{a}$ is an action in $\mathcal{A}$ such that $\widehat{\widehat{a}} = a$. In addition to the set of atomic actions, there is a finite set $\mathcal{B}$ of buffer symbols (or buffers) for asynchronous interprocess communications.\(^2\) In the following, we will use the notation $m\mathcal{B} \overset{df}{=} \text{mult}(\mathcal{B})$. Communications through a buffer $b \in \mathcal{B}$ are represented with asynchronous links of the form $b^+$ for sending or $b^-$ for receiving. Thus, we denote by $\mathcal{L} \overset{df}{=} \{b^+, b^- \mid b \in \mathcal{B}\}$ the set of all possible links.

Since an atomic action can perform simultaneously several synchronous and asynchronous communications, it will have the form $\alpha \beta$ where $\alpha \in m\mathcal{A} \overset{df}{=} \text{mult}(\mathcal{A})$ is a multiaction, representing its synchronous communications (actions), and $\beta \in m\mathcal{L} \overset{df}{=} \text{mult}(\mathcal{L})$ is a multilink, representing its asynchronous communications (links).

We need a device to express formally that, e.g., concurrent actions should synchronise. To this end, we will use partial functions $\varphi : \text{mult}(m\mathcal{A}) \to m\mathcal{A}$, called interface functions, and interpreted as a subset of $\text{mult}(m\mathcal{A}) \times m\mathcal{A}$. For $(\Gamma, \alpha) \in \varphi$, we define $\varphi(\Gamma) = \alpha$.

An example of such a function is the identity $\varphi_{\text{id}} \overset{df}{=} \{(\{\alpha\}, \alpha) \mid \alpha \in m\mathcal{A}\}$. This function is used when no synchronous interface change has to be performed.

The interface function $\varphi_{\text{sy}}$ is used in order to specify when concurrent events labelled by multiactions $\alpha_1, \ldots, \alpha_k$ can synchronise w.r.t $a \in \mathcal{A}$. Formally, $\varphi_{\text{sy}}$ is defined as the smallest set containing $\varphi_{\text{id}}$ and such that if $(\Gamma_1, \alpha_1 + \{a\})$ and $(\Gamma_2, \alpha_2 + \{a\})$ belong to $\varphi_{\text{sy}}$, then $(\Gamma_1 + \Gamma_2, \alpha_1 + \alpha_2)$ belongs to $\varphi_{\text{sy}}$. For instance, we have $\varphi_{\text{sy}}(\{(a_1, a_2), \{\widehat{a}_1, a_3\}, \{\widehat{a}_4\}\}) = \{a_2, a_3\}$, which means that the multiactions $\{a_1, a_1, a_2\}$, $\{\widehat{a}_1, a_3\}$, and $\{\widehat{a}_4\}$ may lead to a multi-way synchronisation represented by the multiaction $\{a_2, a_3\}$, while the fact that $\varphi_{\text{sy}}(\{(a_1, a_1), \{a_2\}\})$ is not defined means that multiactions $\{a_1, \widehat{a}_1\}$ and $\{a_2\}$ cannot synchronise together.

Another useful interface function is $\varphi_{\text{ctr}}$, for $a \in \mathcal{A}$, which allows to setup all synchronous communications w.r.t. $a$ but forbids the execution of events labelled by multiactions still involving $a$. Formally: $\varphi_{\text{ctr}} \overset{df}{=} \{(\Gamma, \alpha) \in \varphi_{\text{sy}} \mid a, \widehat{a} \notin \alpha\}$. Such a function will be used to enforce CCS-like synchronisations, but with no limitation on the number of simultaneously performed synchronisations.

2.3 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 places and transitions, respectively.
- $W$ 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 $\{e, i, x, \triangleright\} \cup \mathcal{B}$, for every place $s \in S$; and $\lambda(t)$ is an interface function $\varphi$ or a multiaction in $m\mathcal{A}$, 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. Collectively, these places are called control (flow) places. If the labelling is $\triangleright$ then $s$ is a closed buffer place, and if it is a buffer symbol $b \in \mathcal{B}$, then $s$ is an open buffer place. The set of all entry (resp. exit) places will be denoted by $^e\Sigma$ (resp. $^\triangleright\Sigma$). We shall also use $M^{\text{ctr}}$ and $M^{\text{opb}}$ to denote $M$ restricted to the control places and to the open buffer places, respectively; finally, $\Sigma^{\text{ctr}}$ is $\Sigma$ with all its buffer places and adjacent arcs removed (intuitively, this also amounts to putting an infinite marking on each closed or open buffer place of $\Sigma$).

\(^2\) The finiteness of $\mathcal{B}$ is not essential, but it will allow us to consider only finite nets. We also assume that $e, x, i, \triangleright \notin \mathcal{B}$. 
We adopt the standard rules about representing nets as directed graphs; moreover, double-headed arrows will represent self-loops. 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 disconnected unmarked buffer places.

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

### 2.4 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 $\text{enabled}(\Sigma)$ 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

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

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

$$\lambda(U) \overset{\text{df}}{=} \sum_{t \in U} U(t) \cdot \lambda(t) \in \text{mult}(m\Lambda).$$

In particular, we will denote $\Sigma[U]\lambda\Sigma'$ whenever there is a multiset of transitions $U$ such that $\Sigma[U]\Sigma'$ and $\lambda = \lambda(U)$. 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 meant. It may happen that two different transition steps correspond to the same labelled step, when different transitions have the same label.

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 $\Sigma[U_1] \Sigma_1[U_2] \Sigma_2 \cdots [U_k] \Sigma_k$. 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.

### 2.5 Clean, ac-free and quasi-safe markings

A marking $M$ of $\Sigma$ is:

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

  If $M^{\text{ctr}} = ^{\circ} \Sigma$, we will say that $\Sigma$ is in an initial state (or marking), and if $M^{\text{ctr}} = ^{\circ} \Sigma$, we will say that $\Sigma$ is in a final state (or marking).

- **ac-free** if, for every transition $t$, there is a control place $s \in ^{\bullet} t$ such that $M(s) < 2 \cdot W(s, t)$, meaning that the marking of the control places does not allow auto-concurrency.

- **quasi-safe** if, for every transition $t$, there is a control place $s \in ^{\bullet} t$ such that $M(s) \leq 1$; note that this implies ac-freeness.
3 An 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 [2]), which allows one to substitute transitions in an operator box by possibly complex asynchronous boxes.

3.1 Asynchronous boxes

An asynchronous box (or, shortly, a box) is a labelled net \( \Sigma \) such that each transition is labelled by a multiaction in \( \mathfrak{mA} \), and \( \Sigma \) itself is:
- \textit{ex-restricted}: there is at least one entry place and at least one exit place;
- \( \mathfrak{B} \)-restricted: for every \( b \in \mathfrak{B} \), there is exactly one \( b \)-labelled place;
- \textit{control-restricted}: for every transition \( t \) there is at least one control place in \( \bullet t \), and at least one control place in \( t \bullet \).

A box \( \Sigma \) is \textit{static} (resp. \textit{dynamic}) if \( M_\Sigma^{\text{str}} = \emptyset \) (resp. \( M_\Sigma^{\text{str}} \neq \emptyset \)) and all the markings reachable from \( M_\Sigma^{\text{str}}, \circ \Sigma \) or \( \Sigma^0 \) in the box \( \Sigma^{\text{str}} \) are both clean and ac-free. The asynchronous boxes, static boxes and dynamic boxes will respectively be denoted by \texttt{abox}, \texttt{abox}\texttt{stc} and \texttt{abox}\texttt{dyn}. In what follows, we will only consider finite boxes and operator boxes.

The basic building blocks, from which other static and dynamic boxes of the MBC will be constructed, are the boxes \( \Sigma_\alpha \beta \), for \( \alpha \in \mathfrak{mA} \) and \( \beta \in \mathfrak{mL} \). Each \( \Sigma_\alpha \beta \) is defined as follows. Its set of places is composed of one entry place \( e \), one exit place \( x \), and one place \( s_b \) labelled \( b \) for each \( b \in \mathfrak{B} \). It has only one transition \( v^{\alpha \beta} \) labelled by \( \alpha \), which has one incoming arc of weight 1 from \( e \) and one outgoing arc of weight 1 to \( x \). The other arcs correspond to the links in \( \beta \) and we have:
\[
W_{\Sigma_\alpha \beta}(s_b, v^{\alpha \beta}) \overset{\alpha}{\Rightarrow} b^{-} \quad \text{and} \quad W_{\Sigma_\alpha \beta}(v^{\alpha \beta}, s_b) \overset{\alpha}{\Rightarrow} b^{+},
\]
for each \( b \in \mathfrak{B} \). The marking of \( \Sigma_\alpha \beta \) is empty. Several examples of such boxes are given in figure 2.

**Proposition 1** Let \( \Sigma \) be a box and \( \Sigma(U) \Sigma' \).

1. If \( \Sigma \) is static, then \( U = \{ \} \) and \( \Sigma = \Sigma' \).
2. If \( \Sigma \) is dynamic, then \( U \) is a set of transitions \(^3\) and \( \Sigma' \) is a dynamic box. \( \square \)

Hence being a dynamic box is invariant over any possible evolution.

We will use the following \textit{marking operators}, which modify the marking of a box \( \Sigma \), where \( b \in \mathfrak{B} \) and \( B \in \mathfrak{mB} \):
- \( \Sigma.B \) adds \( B(b) \) tokens to the \( b \)-labelled open buffer place of \( \Sigma \), for each \( b \in \mathfrak{B} \); in particular, if \( b \in \mathfrak{B} \), then \( \Sigma.b \overset{\alpha}{\Rightarrow} \Sigma.\{b\} \) adds one token to the \( b \)-labelled place of \( \Sigma \). This operation will be called \textit{buffer stuffing}.
- \( \Sigma \) (resp. \( \Sigma' \)) is \( \Sigma \) with one additional token in each entry (resp. exit) place, i.e., \( M_{\Sigma'} \overset{\text{at}}{=}= M_\Sigma + \circ \Sigma \) (resp. \( M_{\Sigma} \overset{\text{at}}{=}= M_\Sigma + \Sigma^0 \)).
- \( \{ \Sigma \} \) 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 2** Let \( \Sigma \) be a box and \( B, B' \in \mathfrak{mB} \).

1. \( \Sigma \) is static iff \( \Sigma.B \) is static, and \( \Sigma \) is dynamic iff \( \Sigma.B \) is dynamic.
2. \( \Sigma \) is dynamic iff \( \Sigma \) is static iff \( \Sigma \) is dynamic.
3. \( \Sigma.\emptyset = \Sigma \), \( \Sigma.B.B' = \Sigma.(B + B') \), \( \Sigma.B = \Sigma.B \) and \( \Sigma.B = \Sigma.B \).
4. If \( \Sigma \) is static or dynamic, then \( \{ \Sigma \} \) and \( \{ \Sigma \} \) are static boxes.
5. If \( \Sigma \) is static, then \( \{ \Sigma \} = \Sigma \).
6. \( \{ \{ \Sigma \} \} = \{ \{ \Sigma \} \} = \{ \{ \Sigma \} \} = \{ \{ \Sigma \} \} \).
7. \( \{ \Sigma \}.B = \{ \{ \Sigma \} \}, \{ \{ \Sigma \} \} = \{ \{ \Sigma \} \} = \{ \Sigma \} \) and \( \{ \{ \Sigma \} \} = \{ \{ \Sigma \} \} = \{ \Sigma \} \).

\(^3\) But if \( \Sigma(G) \Sigma' \), then the labelled step \( G \) may be a true multiset of multiactions.
3.2 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 (based on transitions and on labels), we introduce two kinds of transition systems.

The **full transition system** of a dynamic box $\Sigma$ is $\text{fts}_\Sigma \overset{df}{=} (V, L, \text{init})$ where $V \overset{df}{=} |\Sigma|$ is the set of states; $L \overset{df}{=} 2^T$ is the set of arc labels; $A \overset{df}{=} \{(\Sigma', U, \Sigma'') \in V \times L \times V \mid \Sigma'|U, \Sigma''\}$ is the set of arcs; and $\text{init} \overset{df}{=} \Sigma$ is the initial state. For a static box $\Sigma$, $\text{fts}_\Sigma \overset{df}{=} \text{fts}_{\Sigma,\Sigma}^\text{init}$.

The **labelled transition system** of a static or dynamic box $\Sigma$, denoted by $\text{lts}_\Sigma$, is defined as $\text{fts}_\Sigma$ with each arc label $U$ changed to $\lambda_{\Sigma}(U)$. Note that many different arcs between two states in $\text{fts}_\Sigma$ may lead to a single arc in $\text{fts}_\Sigma$.

As usual, if $(V, L, \text{init})$ and $(V', L', A', \text{init}')$ are two transition systems, 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 5, we construct binary relations which are isomorphisms between transition systems of boxes and the corresponding box expressions.
3.3 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 $\Sigma_v$, 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.

We will consider four groups of operator boxes for MBC as described below.

Sequential operators. A sequential operator box $\Omega_{sq}$ is an operator box such that: no place is disconnected; there is exactly one $e$-labelled place, and exactly one $x$-labelled place; and, for every transition $v \in T_{\Omega_{sq}}$, $|v| = |v^*| = 1$ and $\lambda_{\Omega_{sq}}(v) = \varphi_{id}$. That is, $\Omega_{sq}$ can be thought of as a finite automaton in which each transition will be substituted by a potentially complex box by the net substitution operation. We assume that all the transitions have basic transition identities, and that two distinct operator boxes have disjoint sets of nodes. The domain of application of $\Omega_{sq}$ is the set $\text{dom}_{\Omega_{sq}}$ comprising all $\Omega_{sq}$-tuples of static and dynamic boxes such that at most one box is dynamic.

Examples of sequential operator boxes are choice $\Omega_{\sqcap}$, iteration $\Omega_\oplus$, and sequence $\Omega_\lhd$, depicted in figure 3. They are all binary, with the domain of application $\text{dom}_\Omega = \text{dom}_{\Omega_{sq}} = \text{dom}_\Omega, \overset{df}{=} (\text{abox}^{stc})^2 \cup (\text{abox}^{dyn} \times \text{abox}^{stc}) \cup (\text{abox}^{stc} \times \text{abox}^{dyn})$. We will denote: $\Sigma_1 \sqcap \Sigma_2 \overset{df}{=} \Omega_{\sqcap}(\Sigma_1, \Sigma_2)$, $\Sigma_1 \oplus \Sigma_2 \overset{df}{=} \Omega_\oplus(\Sigma_1, \Sigma_2)$ and $\Sigma_1 ; \Sigma_2 \overset{df}{=} \Omega_\lhd(\Sigma_1, \Sigma_2)$.

Parallel composition $\Omega_\parallel$. This is also a binary operator box (see figure 3), but with the domain of application $\text{dom}_\Omega, \overset{df}{=} (\text{abox}^{stc})^2 \cup (\text{abox}^{dyn})^2$, so that its two operands may evolve concurrently. We will denote $\Sigma_1 \parallel \Sigma_2 \overset{df}{=} \Omega_\parallel(\Sigma_1, \Sigma_2)$.

Communication interface operators. A unary communication interface operator box $\Omega_\varphi$, shown in figure 3, is parameterised by an interface function $\varphi : \text{mult}(mA) \setminus \{\emptyset\} \to mA$, and has the domain of application $\text{dom}_{\Omega_\varphi} \overset{df}{=} \text{abox}^{stc} \cup \text{abox}^{dyn}$. The role of $\Omega_\varphi$ will be to effect the change of synchronous communication interface specified by $\varphi$. It has the domain of application $\text{dom}_{\Omega_\varphi} \overset{df}{=} \text{abox}^{stc} \cup \text{abox}^{dyn}$.

Examples of such operators are the scoping operators, which are parameterised by an action $a \in A$. We will denote $\Sigma \text{sc} a \overset{df}{=} \Omega_{\text{sc} a}(\Sigma)$.

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

3.4 Net substitution

Throughout the rest of the paper, the identities of transitions in asynchronous boxes will play a key 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.

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 in figure 2 and 3 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:
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. With this assumption, in the following we shall construct new places by grouping the 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.

**Sequential and parallel operators.** Let \( \Omega \) be a sequential or the parallel operator with transitions \( v_1, \ldots, v_n \), and \( \Sigma = (\Sigma_1, \ldots, \Sigma_n) = (\Sigma_1', \ldots, \Sigma_n') \) be a tuple of boxes in \( \text{dom}_\Omega \). Then \( \Omega(\Sigma) = \Phi \) whose components are defined as follows.

The set of transitions of \( \Phi \) is the set of all trees \( v_i \triangleleft t \) (with \( t \in T_{\Sigma_i} \) and \( i \in \{1, \ldots, n\} \)). The label of each \( v_i \triangleleft t \) 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 \( w \triangleleft t \), the weight function is given by:

\[
W_\Phi(p, w \triangleleft t) \triangleq \begin{cases} W_{\Sigma_i}(p, t) & \text{if } w = v_i \\ 0 & \text{otherwise} \end{cases},
\]
and similarly for \( W_\Phi(w \triangleleft t, p) \).

For every place \( s \in S_\Phi \) with \( s = \{u_1, \ldots, u_m\} \) and \( s^* = \{w_1, \ldots, w_m\} \), we construct in \( S_\Phi \) all the places of the form \( p \triangleq (x_1, \ldots, x_k, e_1, \ldots, e_m) \), where each \( x_i \) is an exit place of \( \Sigma_{u_i} \), and each \( e_j \) 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 \( w \triangleleft t \), the weight function is given by:

\[
W_\Phi(p, w \triangleleft t) \triangleq \begin{cases} W_{\Sigma_v}(x_i, t) + W_{\Sigma_w}(e_j, t) & \text{if } w \in s \cap s^* \text{ and } w = u_i = w_j \\ W_{\Sigma_v}(x_i, t) & \text{if } w \in s \cap s^* \text{ and } w = u_i \\ W_{\Sigma_w}(e_j, t) & \text{if } w \in s \cap s^* \text{ and } w = w_j \\ 0 & \text{otherwise} \end{cases},
\]
and similarly for \( W_\Phi(w \triangleleft t, p) \). We will denote by \( \mathbb{P}(s) \) the set of all places \( p \) as defined above for a given \( s \), and by \( \mathbb{P}(s, q) \) (resp. \( \mathbb{P}(s, q, q') \)) the sets of all those \( p \in \mathbb{P}(s) \) in which \( q \) (resp. \( q \) and \( q' \)) is present.

For every \( b \in \mathbb{B} \), there is a unique \( b \)-labelled place \( p^b \triangleq (p^b_{v_1}, \ldots, p^b_{v_n}) \in S_\Omega(\Sigma) \), 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, and for every transition \( w \triangleleft t \), the weight function is given by:

\[
W_{\Omega(\Sigma)}(p^b, w \triangleleft t) \triangleq W_{\Sigma_w}(p^b_w, t),
\]
and similarly for \( W_{\Omega(\Sigma)}(w \triangleleft t, p^b) \).

**Communication interface operators.** 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 multiset of transitions whose labels match the argument, i.e., the non-empty multiset of multiactions \( \Gamma \). 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 \( t \triangleq v_\varphi \triangleleft \{t_1, \ldots, t_l\} \) such that \( \{t_1, \ldots, t_l\} \in \text{mult}(T_{\Sigma}) \) and the multiset \( \Lambda \triangleq \{\lambda_\Sigma(t_1), \ldots, \lambda_\Sigma(t_l)\} \) belongs to the domain of \( \varphi \). The label of \( t \) is \( \varphi(\Lambda) \), and for a place \( p \) of \( \Omega_\varphi(\Sigma) \), the weight function is given by:

\[
W_{\Omega_\varphi(\Sigma)}(p, t) \triangleq \sum_{i=1}^{l} W_{\Sigma}(p, t_i),
\]
and similarly for \( W_{\Omega \Sigma}(t, p) \).

**Buffer restriction.** An application of the buffer restriction operator \( \Omega_{\text{tie}b} \) 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 \( \text{tie}b \), the label of the only \( b \)-labelled place is changed to \( b \), and a new unmarked disconnected \( b \)-labelled place is added to \( S_{\Omega_{\text{tie}b} \Sigma} \).

### 3.5 Consistency in the box domain

For a constructed net, the property of having an empty control marking is directly linked to the same property about the arguments.

**Proposition 3** 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 \( M_{\Sigma v_i}^{\text{ctr}} = \emptyset \), for each \( v_i \in T_{\Omega} \).
2. If \( \Sigma' \) is \( \Omega_{\text{tie}b}(\Sigma) \) or \( \Sigma \text{tie}b \) or \( \Sigma.B \), then \( M_{\Sigma'}^{\text{ctr}} = \emptyset \) iff \( M_{\Sigma}^{\text{ctr}} = \emptyset \).

Moreover, the operation of net substitution always returns a syntactically valid object provided that it is applied to operands belonging to the correct domain. This is shown by the two lemmata and the theorem below.

**Lemma 4** 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 \( M^{\text{ctr}} \in \{ \Sigma, \Sigma^* \} \).

**Lemma 5** Let \( \Omega_{\Sigma}(\Sigma) \) be a legal application of a sequential operator box \( \Omega_{\text{seq}} \), and \( z \in T_{\Omega_{\Sigma}} \) be such that \( \Sigma = \Sigma_z \) is a dynamic box. Moreover, let \( s'' \) be a place in \( \Omega_{\Sigma}, M \equiv M_{\Omega_{\Sigma}(\Sigma)}, \Sigma = \{s\} \) and \( z = \{s'\} \).

1. If there is a place in \( P(s'') \) which is marked at \( M \) then \( s'' \in \{s, s'\} \).
2. If \( M_{\Sigma}^{\text{ctr}} = \Sigma^* \) then \( M^{\text{ctr}} = P(s) \), and if \( M_{\Sigma}^{\text{ctr}} = \Sigma^0 \) then \( M^{\text{ctr}} = P(s') \).
3. If \( M^{\text{ctr}} \geq P(s'') \) then \( M^{\text{ctr}} = P(s'') \), and one of the following holds:
   (a) \( s'' = s \neq s' \) and \( M_{\Sigma}^{\text{ctr}} = \Sigma^* \).
   (b) \( s'' = s' \neq s \) and \( M_{\Sigma}^{\text{ctr}} = \Sigma^0 \).
   (c) \( s'' = s = s' \) and \( M_{\Sigma}^{\text{ctr}} \in \{ \Sigma^*, \Sigma^0 \} \).

**Theorem 6.** Let \( \Omega \) be any 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.
4.1 Static properties of boxes

An important result from the point of view of developing an algebra of box and box expressions is given next.

Proposition 7 Let Ω be a sequential operator box, and Σ be an Ω-tuple of static boxes.

1. If \( v \in T_Ω \) is such that \( ^sΩ = ^vΩ or ^sΩ = v^s \), then \( Ω(Σ) = Ω(Σ') \), where \( Σ' \) is \( Σ \) with \( Σ_v \) replaced respectively by \( Σ_v \) or \( Σ_w \).

2. If \( v \in T_Ω \) is such that \( Ω^v = Ω^w \) or \( Ω^v = v^s \), then \( Ω(Σ) = Ω(Σ') \), where \( Σ' \) is \( Σ \) with \( Σ_v \) replaced respectively by \( Σ_v \) or \( Σ_w \).

4.2 Structural equivalence

We intend here to capture situations where different applications of a same operator box lead to the same labelled net. Let \( Ω \) be an operator box and \( (Ω, \Theta) \) be \( Ω \)-tuples of boxes. We start by defining five auxiliary relations \( \equiv_i^Ω \), in the following way.

- \( \equiv_0^Ω \) if there is a transition \( v \in T_Ω \) and a box \( Ψ \) such that \( ^vΩ = v^s \), \( \{Σ_v, Θ_v\} = \{Ψ_v, Θ_v\} \) and \( Σ_u = Θ_u \), for all \( u \in T_Ω \setminus \{v\} \).

- \( \equiv_1^Ω \) if there are two transitions \( v \neq w \in T_Ω \) and two boxes \( Ψ_v \) and \( Ψ_w \) such that \( ^vΩ = ^wΩ \), \( \{Σ_v, Σ_w\}, (Θ_v, Θ_w) \} = \{Ψ_v, Ψ_w\} \) and \( Σ_u = Θ_u \), for all \( u \in T_Ω \setminus \{v, w\} \).

- \( \equiv_2^Ω \) if there are two transitions \( v \neq w \in T_Ω \) and two boxes \( Ψ_v \) and \( Ψ_w \) such that \( v^s = w^s \), \( \{Σ_v, Σ_w\}, (Θ_v, Θ_w) \} = \{Ψ_v, Ψ_w\} \) and \( Σ_u = Θ_u \), for all \( u \in T_Ω \setminus \{v, w\} \).

- \( \equiv_3^Ω \) if, for each \( v \in T_Ω \), there is a box \( Ψ_v \) and two multisets \( B_v, B'_v \) over \( Ω \), such that \( \sum_{v \in T_Ω} B_v = \sum_{v \in T_Ω} B'_v \), and \( Σ_u = Ψ_u, B_v \) and \( Θ_u = Ψ_u, B'_v \), for all \( v \in T_Ω \).

Then we define \( \equiv_Ω = \equiv_1^Ω \circ \bigcup_{i \in I} \equiv_i^Ω \), where \( I = \{0, 1, 2, 3, 5\} \) and \( \equiv_5^Ω \) is \( id_{\text{abox}} \). Note that \( \equiv_{Ω_1} = \equiv_{Ω_1}^4 \) and \( \equiv_{Ω_1} = id_{\text{abox}} \), for every unary operator box \( Ω \).

Proposition 8 Let \( Ω \) be an operator box, and \( Σ \in \text{dom}_Ω \).

1. If \( Σ \equiv Ω \), then \( Θ \in \text{dom}_Ω \) and \( [Σ] = [Θ] \).

2. \( \equiv_Ω \) is an equivalence relation on \( \text{dom}_Ω \).

3. If \( Θ \in \text{dom}_Ω \) and \( [Σ] = [Θ] \), then \( Ω(Σ) = Ω(Θ) \) iff \( Σ \equiv_Ω Θ \).

4.3 Dynamic properties of composite nets

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

Proposition 9 Let \( Σ = (Σ_1, Σ_2) \in \text{dom}_Ω \). Then \( \text{enabled}(Ω(Σ)) \) comprises exactly all sets of transitions \( U = (ν_i \downarrow U_1) ∪ (ν_i \downarrow U_2) \) such that there is a pair of boxes \( Θ \) satisfying \( Θ \equiv_Ω Σ \) and \( U_i \in \text{enabled}(Θ_i) \), for \( i \in \{1, 2\} \). Moreover, \( Ω(Σ) \big| U \) \( Ω(Φ) \), where \( Θ_i \downarrow U_i \downarrow Φ_i \), for \( i \in \{1, 2\} \).
Proposition 10 Let $\Omega$ be a sequential operator box and $\Sigma \in \text{dom}_\Omega$.

1. If $\Sigma_i[U_i] \Theta_i$ (for $i \leq n$), then $\Theta \in \text{dom}_\Omega$, and
   $$\Omega(\Sigma)[(v_1 \downarrow U_1) \cup \ldots \cup (v_n \downarrow U_n)] \Omega(\Theta).$$

2. If $\Omega(\Sigma)[U] \Theta$, then there are $\Psi, \Phi \in \text{dom}_\Omega$ 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
   $$U = (v_1 \downarrow U_1) \cup \ldots \cup (v_n \downarrow U_n).$$

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

Notice that for the sequential operators, at most one $U_i$ is non-empty. Otherwise, the property is very similar to that which holds for the parallel operator box.

Proposition 11 Let $\Sigma$ be a box, $b \in \mathbb{B}$, $B \in \text{mult}(\{b\})$ and $B' \in \mathbb{M}_\mathbb{B}$.

1. $\text{enabled}(\Sigma \downarrow b)$ comprises exactly all sets of transitions $U = v^{\uparrow b} \downarrow V$ such that $V \in \text{enabled}(\Sigma)$.
   Moreover, $\Sigma \downarrow b[U] \Phi \downarrow b$, where $\Sigma[V] \Phi$.

2. $\text{enabled}((\Sigma \downarrow b).B)$ comprises exactly all sets of transitions $U \in \text{enabled}(\Sigma \downarrow b)$.
   Moreover, $(\Sigma \downarrow b).B \downarrow U \Phi \downarrow b$, where $\Sigma \downarrow b[U] \Phi$.

3. $\text{enabled}(\Sigma)$ is a subset of $\text{enabled}(\Sigma.\downarrow b').$
   Moreover, if $\Sigma[U] \Phi$ then $\Sigma.\downarrow b'[U] \Phi.\downarrow b'$.

That the converse of the last property does not hold may be illustrated by a simple counter-example: the dynamic box $\Sigma_{\{1\} \downarrow \downarrow 1}$ only allows the empty step, while $\Sigma_{\{1\} \downarrow \downarrow 1} \downarrow \downarrow 1 \in \text{enabled}(\Sigma)$ and each $\lambda\Sigma(U_i)$ belongs to the domain of $\varphi$.

Proposition 12 Let $\Sigma$ be a static or dynamic box, and $\Omega_\varphi$ be a communication interface operator box.

1. If $\Sigma[U_1 \cup \ldots \cup U_k] \Theta$ and each $\lambda\Sigma(U_i)$ belongs to the domain of $\varphi$, then
   $$\Omega_\varphi(\Sigma)[\{v^r \downarrow U_1, \ldots, v^r \downarrow U_k\}] \Omega_\varphi(\Theta).$$

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

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

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

Proposition 13 Let $B \in \mathbb{M}_\mathbb{B}$ and $\Sigma_{\alpha\beta}$ be one of the basic boxes. If $B(b) \geq \beta(b^-)$, for each $b \in \mathbb{B}$, then the only non-empty steps of $\Sigma_{\alpha\beta}.B$ are

$$\Sigma_{\alpha\beta}.B \left[\{v^{\alpha b}\}\right] \Sigma_{\alpha\beta}.B \left[\left(\sum_{b \in B} b - \sum_{b \in B} \beta(b^-) \cdot b + \sum_{b \in B} \beta(b^+) \cdot b\right)\right].$$

otherwise, $\Sigma_{\alpha\beta}.B$ has no non-empty step.

Various important consequences may be derived from the results presented above; in particular that the way static and dynamic boxes are composed in MBC guarantees that the result is a static or dynamic box when the domain of application of the operators is respected.
Proposition 14. Let $\Omega$ be an operator box of MBC and $\Sigma \in \text{dom}_\Omega$. Then every net derivable from $\Omega(\Sigma)$ is of the form $\Omega(\Theta)$, where $\Theta \in \text{dom}_\Omega$ and $[\Theta] = [\Sigma]$. Moreover, if no box in $\Sigma$ is dynamic, then every net derivable from $\Omega(\Sigma)$ or $\Omega(\Sigma)$ is of the form $\Omega(\Theta)$, where $\Theta \in \text{dom}_\Omega$ and $[\Theta] = [\Sigma]$. □

Theorem 15. Every composite net of MBC is a quasi-safe static or dynamic box. Moreover, it is static iff the marking operators $(\cdot), (\cdot)$ are not used, unless in the scope of the $[\cdot]$ or $[\cdot]$ operators. □

5 An algebra of asynchronous box expressions

We consider an algebra of process expressions over the signature:

$$C \cup \{ (\cdot), (\cdot) \} \cup \{ ||, ;, \square, \oplus \} \cup \{ sc\ a \mid a \in A \} \cup \{ \text{tie} b, .b \mid b \in B \} ,$$

(1)

where $C \overset{\text{df}}{=} \{ \alpha\beta \mid \alpha \in mA, \beta \in mL \}$ are the constants; the binary operators $||$, $;$, $\sqcup$ and $\oplus$ will be used in the infix mode; the unary operators $sc\ a$, $\text{tie} b$ and $.b$ will be used in the postfix mode; and $(\cdot)$ and $(\cdot)$ are two positional unary operators (the position of the argument being given by the dot).

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

$$\text{aexpr}^{\text{stc}} \quad E ::= \alpha\beta \mid E\ sc\ a \mid E\ \text{tie} b \mid E.b \mid E||E \mid E\ \sqcup E \mid E;E \mid E \oplus E$$

$$\text{aexpr}^{\text{dyn}} \quad D ::= \overline{E} \mid \overline{E} \mid D\ sc\ a \mid D\ \text{tie} b \mid D.b \mid D||D \mid D\ \sqcup D \mid D\ \oplus D \mid D\ \oplus D$$

(2)

where $\alpha\beta \in C$, $a \in A$ and $b \in B$. Moreover, we will use $F$ to denote any static or dynamic expression.

We also use the notations $[F]$ and $[F]$ yielding static expressions, where $[F]$ is $F$ with all occurrences of $(\cdot)$ and $(\cdot)$ removed, and $[F]$ 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 $F.b \cdots b'$ (but such terms can be used as a convenient shorthand).

Essentially, a box expression encodes the structure of a box, together with the current marking of the control places (using overbars and underbars) and of the buffer places (using the $.b$'s). Thus, a box expression $\overline{E}$ represents $E$ in its initial state (in terms of nets, this corresponds to the initially marked box of $E$). Similarly, $\underline{E}$ represents $E$ in its final state. Note that the $b$ notation is needed for static as well as for dynamic box expressions because the dormant part of a dynamic box expression may still have $.b$'s which are later needed in the active part. For instance, $D \overset{\text{df}}{=} \{ a \} \{ b^+ \}.b : \overline{\{ a_f \} \{ b^- \}}$ has a static component with $a \ .b$ in it, and may be transformed into an equivalent $D' \overset{\text{df}}{=} \{ a \} \{ b^+ \} : \overline{\{ a_f \} \{ b^- \}}$ (see section 5.2).

5.1 Denotational semantics

The denotational semantics of box expressions is given in the form of a mapping $\text{box} : \text{aexpr} \to \text{abox}$, defined homomorphically by induction on their structure, following the syntax (2). Below, $\alpha\beta \in C$, $a \in A$, $b \in B$. $\text{una}$ stands for any unary operator ($sc\ a$, $\text{tie} b$ or $.b$), and $\text{bin}$ for any binary operator ($||$, $\sqcup$, $;$ or $\oplus$).

$$\text{box}(\alpha\beta) \overset{\text{df}}{=} \Sigma_{\alpha\beta} \quad \text{box}(\overline{E}) \overset{\text{df}}{=} \overline{\text{box}(E)} \quad \text{box}(\underline{E}) \overset{\text{df}}{=} \underline{\text{box}(E)}$$

$$\text{box}(F \ \text{una}) \overset{\text{df}}{=} \text{box}(F) \ \text{una} \quad \text{box}(F \ \text{bin} F_2) \overset{\text{df}}{=} \text{box}(F_1) \ \text{bin} \ \text{box}(F_2) .$$

(3)
The semantical mapping always returns a box, and the property of corresponding to a static or dynamic box has been captured by the syntax (2).

**Theorem 16.** 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.

5.2 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. These rules directly follow those of the ABC and PNA models.

<table>
<thead>
<tr>
<th>CON1</th>
<th>$F \equiv F'$</th>
<th>$F\text{ una} \equiv F'\text{ una}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENT</td>
<td>$E \equiv E'$</td>
<td>$E \equiv E'$</td>
</tr>
<tr>
<td>OPL</td>
<td>$(F.\text{ b}) \text{ bin } F' \equiv (F \text{ bin } F')\text{ b}$</td>
<td></td>
</tr>
<tr>
<td>E1</td>
<td>$F\text{ una} \equiv E\text{ una}$</td>
<td></td>
</tr>
<tr>
<td>B1</td>
<td>$(F.\text{ b}) \text{ una} \equiv (F \text{ una})\text{ b}$ if una $\neq$ b</td>
<td></td>
</tr>
<tr>
<td>IS1</td>
<td>$E_1; E_2 \equiv E_1; E_2$</td>
<td></td>
</tr>
<tr>
<td>IS2</td>
<td>$E_1; E_2 \equiv E_1; E_2$</td>
<td></td>
</tr>
<tr>
<td>OPR</td>
<td>$F\text{ bin } (F'\text{ b}) \equiv (F \text{ bin } F')\text{ b}$</td>
<td></td>
</tr>
<tr>
<td>X1</td>
<td>$E\text{ una} \equiv F\text{ una}$</td>
<td></td>
</tr>
<tr>
<td>IPAR1</td>
<td>$E_1|E_2 \equiv E_1|E_2$</td>
<td></td>
</tr>
<tr>
<td>IC1L</td>
<td>$E_1\square E_2 \equiv E_1\square E_2$</td>
<td></td>
</tr>
<tr>
<td>IC2L</td>
<td>$E_1\square E_2 \equiv E_1\square E_2$</td>
<td></td>
</tr>
<tr>
<td>IC1R</td>
<td>$E_1\square E_2 \equiv E_1\square E_2$</td>
<td></td>
</tr>
<tr>
<td>IC2R</td>
<td>$E_1\square E_2 \equiv E_1\square E_2$</td>
<td></td>
</tr>
<tr>
<td>HT1</td>
<td>$E_1 \circ E_2 \equiv E_1 \circ E_2$</td>
<td></td>
</tr>
<tr>
<td>HT2</td>
<td>$E_1 \circ E_2 \equiv E_1 \circ E_2$</td>
<td></td>
</tr>
<tr>
<td>HT3</td>
<td>$E_1 \circ E_2 \equiv E_1 \circ E_2$</td>
<td></td>
</tr>
<tr>
<td>HT4</td>
<td>$E_1 \circ E_2 \equiv E_1 \circ E_2$</td>
<td></td>
</tr>
<tr>
<td>HT5</td>
<td>$E_1 \circ E_2 \equiv E_1 \circ E_2$</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1.** Structural similarity relation for MBC, where $b \in B$, una stands for any unary MBC operator and bin stands for any binary MBC operator.

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, as shown below.

**Theorem 17.** Let $F_1$ and $F_2$ be box expressions.

1. If $F_1 \equiv F_2$ then $[F_1] \equiv [F_2]$, $[F_1] = [F_2]$ and box($F_1$) = box($F_2$).
2. If $[F_1] = [F_2]$, then box($F_1$) = box($F_2$) iff $F_1 \equiv F_2$. □
That the precondition $\|F_1\| = \|F_2\|$ is needed in the second part of the last result may be justified by the counter-example $F_1 \equiv \{a\} \text{sc} a$ and $F_2 \equiv \{\tilde{a}\} \text{sc} a$ for which $F_1 \neq F_2$ but $\text{box}(F_1) = \text{box}(F_2)$ (no transition is left in the nets by the scoping operation).

Theorem 18. Let $F$ be a box expression.

1. $\text{box}(F) = \|\text{box}(F)\|$ iff $F \equiv \|F\|$.
2. $\text{box}(F) = \|\text{box}(F)\|$ iff $F \equiv \|F\|$.

In the first case above we say that $F$ is an initial expression, and in the second a final one.

In developing the operational semantics of the box algebra, we first introduce operational rules. They are based on transitions of the nets providing the denotational semantics of box expressions. Based on these, we will formulate our key consistency result. Then we will introduce the label based rules, together with the derived consistency results.

5.3 Transition based operational semantics

Consider the set $\mathbb{T}$ of all transition trees in the boxes derived through the box mapping. It is easy to check that each $t \in \mathbb{T}$ has always the same label in all the boxes derived through the box mapping where it occurs; it will be denoted 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 $U \in \bigcup \equiv \text{mult}(\mathbb{T})$. The idea here is that $U$ is a valid step for the boxes associated with $F$ and $F'$, i.e., that $\text{box}(F) \mid U \text{box}(F')$, as stated by theorem 20.

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. Notice that we use $F \xrightarrow{U} F'$ to denote $(F, U, F') \in \rightarrow$. In the definition of $\rightarrow$, we make no restriction on $U_1$ and $U_2$ but the domain of application of $\text{bin}$ will ensure that this rule will always be used with the correct static/dynamic mixture of boxes. For instance, in the case of the choice operator, one of $U_1$ and $U_2$ is necessarily empty, and in the rule of the $\text{sc} a$ operator, each $U_i$ contains only transitions whose labels can be scoped together.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EA</strong></td>
<td>$\alpha \beta \cdot (\sum_{b \in B} \beta(b^-) \cdot b) \xrightarrow{\gamma \alpha \beta \cdot (\sum_{b \in B} \beta(b^+) \cdot b)}$</td>
</tr>
<tr>
<td><strong>EQ1</strong></td>
<td>$F \xrightarrow{1} F$</td>
</tr>
<tr>
<td><strong>EQ2</strong></td>
<td>$F \equiv F', F'' \xrightarrow{U} F''', F''' \equiv F'''$</td>
</tr>
<tr>
<td><strong>EBUF</strong></td>
<td>$F \xrightarrow{U} F'$ if $U \equiv \text{U} \text{F} F$</td>
</tr>
<tr>
<td><strong>ETIE</strong></td>
<td>$F \xrightarrow{U} F'$ if $\lambda(U) \in \text{dom}(\varphi_{\text{sc} a})$</td>
</tr>
<tr>
<td><strong>ESC</strong></td>
<td>$F \xrightarrow{U_1 + \ldots + U_k} F'$ if $\forall i : \lambda(U_i) \in \text{dom}(\varphi_{\text{sc} a})$</td>
</tr>
<tr>
<td><strong>EOP</strong></td>
<td>$F_1 \xrightarrow{U_1} F_1', F_2 \xrightarrow{U_2} F_2'$</td>
</tr>
</tbody>
</table>

Table 2. Transition based operational semantics for MBC, where $a \in A$, $b \in B$ and bin stands for any binary MBC operator.
\[
\begin{align*}
\text{PAYRISE} &
\equiv (\{a_r\} \circ \{a_c\}) \sqcap (\{\hat{\alpha}, \hat{\alpha}\} \circ \{b_n, b_u\}) \text{tie } b_d \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{E1, IPAR1} \\
\equiv (\{a_r\} \circ \{a_c\}) \sqcap (\{\hat{\alpha}, \hat{\alpha}\} \circ \{b_n, b_u\}) \text{tie } b_d \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{IT1, IC1R} \\
\begin{array}{ll}
\{t_1\} & (\{a_r\} \circ \{a_c\}) \sqcap (\{\hat{\alpha}, \hat{\alpha}\} \circ \{b_n, b_u\}) \text{tie } b_d \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{EA, EOP} \\
\end{array} \\
\equiv (\{a_r\} \circ \{a_c\}) \sqcap (\{\hat{\alpha}, \hat{\alpha}\} \circ \{b_n, b_u\}) \text{tie } b_d \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{ESC, ETIE} \\
\equiv (\{a_r\} \circ \{a_c\}) \sqcap (\{\hat{\alpha}, \hat{\alpha}\} \circ \{b_n, b_u\}) \text{tie } b_d \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{IC2R} \\
\begin{array}{ll}
\{t_2\} & (\{a_r\} \circ \{a_c\}) \sqcap (\{\hat{\alpha}, \hat{\alpha}\} \circ \{b_n, b_u\}) \text{tie } b_d \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{EA, EOP} \\
\end{array} \\
\equiv (\{a_r\} \circ \{a_c\}) \sqcap (\{\hat{\alpha}, \hat{\alpha}\} \circ \{b_n, b_u\}) \text{tie } b_d \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{ESY, ETIE} \\
\begin{array}{ll}
\{t_3\} & (\{a_r\} \circ \{a_c\}) \sqcap (\{\hat{\alpha}, \hat{\alpha}\} \circ \{b_n, b_u\}) \text{tie } b_d \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{IS1} \\
\end{array} \\
\equiv (\{a_r\} \circ \{a_c\}) \sqcap (\{\hat{\alpha}, \hat{\alpha}\} \circ \{b_n, b_u\}) \text{tie } b_d \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{EA} \\
\equiv (\{a_r\} \circ \{a_c\}) \sqcap (\{\hat{\alpha}, \hat{\alpha}\} \circ \{b_n, b_u\}) \text{tie } b_d \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{IC2L} \\
\equiv (\{a_r\} \circ \{a_c\}) \sqcap (\{\hat{\alpha}, \hat{\alpha}\} \circ \{b_n, b_u\}) \text{tie } b_d \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{IT3, OPR} \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{OPL, B1} \\
\begin{array}{ll}
\{t_4, t_5\} & (\{a_r\} \circ \{a_c\}) \sqcap (\{\hat{\alpha}, \hat{\alpha}\} \circ \{b_n, b_u\}) \text{tie } b_d \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{EA, EOP, ESY} \\
\end{array} \\
\equiv \text{PAYRISE} \quad \text{IT5, IC2R} \\
\begin{array}{ll}
\{t_4, t_5\} & (\{a_r\} \circ \{a_c\}) \sqcap (\{\hat{\alpha}, \hat{\alpha}\} \circ \{b_n, b_u\}) \text{tie } b_d \\
& \quad \text{sc } a_r \text{ sc } a_c \quad \text{ERS, ETIE} \\
\end{array} \\
\equiv \text{PAYRISE} \quad \text{IPAR2, X1}
\end{align*}

Table 3. Execution scenario I
We will now derive some properties of the derivation rules. First, an empty move always relates two structurally equivalent box expressions.

**Proposition 19** Let $F$ and $F'$ be two box expressions. Then, $F \overset{\emptyset}{\longrightarrow} F'$ iff $F \equiv F'$. \hfill $\square$

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

**Theorem 20.** Let $F$ be a box expression.

1. If $F \overset{U}{\longrightarrow} F'$, then $F'$ is a box expression such that $\text{box}(F)[U]\text{box}(F')$ and $\{F\} = \{F'\}$.
2. If $\text{box}(F)[U]\Sigma$, then there is a box expression $F'$ such that $\text{box}(F') = \Sigma$ and $F \overset{U}{\longrightarrow} F'$. \hfill $\square$

### 5.4 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 20, we are now in a position to relate transition systems generated by a box expression and the corresponding box.

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 $D' \overset{U}{\longrightarrow} D''$, for some $U \in U$, 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 $V \overset{\equiv}{=} \{[D']_\equiv | D' \in [D]\}$ is the set of states; $L \overset{\equiv}{=} U$ is the set of arc labels; $A \overset{\equiv}{=} \{([D']_\equiv, U,[D'']_\equiv) \in V \times U \times V | D' \overset{U}{\longrightarrow} D''\}$ is the set of arcs; and $\text{init} \overset{\equiv}{=} [D]_\equiv$ is the initial state.

For a static box expression $E$, $\text{fts}_E \overset{\equiv}{=} \text{fts}_{\text{box}(F)}$.

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 $D \overset{\equiv}{=} D'$ for two different expressions $D$ and $D'$, whereas in the domain of boxes, $\Sigma \{\{\}\} \Theta$ 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 21.** For every box expression $F$, $\text{iso}_F \overset{\equiv}{=} \{([F']_\equiv, \text{box}(F')) | [F']_\equiv \text{ is a node of } \text{fts}_F \}$ is an isomorphism between the full transition systems $\text{fts}_E$ and $\text{fts}_{\text{box}(F)}$. Moreover, $\text{iso}_F$ preserves the property of being in an initial or final state.\hfill $\square$

### 5.5 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 $F \overset{\Gamma}{\longrightarrow} F'$, where $F$ and $F'$ are box expressions as before, and $\Gamma \in \text{mult}(\mathcal{A})$, as shown in table 4.

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

**Proposition 22** Let $F$ be a box expression and $\Gamma \in \mathcal{L}$. Then $F \overset{\Gamma}{\longrightarrow} F'$ iff there is $U \in U$ such that $F \overset{U}{\longrightarrow} F'$ and $\lambda(U) = \Gamma$. \hfill $\square$

---

4 In terms of box expression (cf. theorem 18) or box net (cf. section 2.5), not w.r.t. the transition system.
The results concerning transition based operational semantics directly extend to the label based one. Let $F$ be a box expression. In view of proposition 22, the label based operational semantics of $F$ is faithfully captured by the labelled transition system of $F$, denoted by $\text{lts}_F$, and defined as $\text{fts}_F$ with each arc label $U$ changed to $(U)$. The consistency result for the label based operational semantics can then be formulated thus.

**Theorem 23.** For every box expression $F$, $\text{iso}_F \overset{df}{=} \{ ([F']_\equiv, \text{box}(F')) \mid [F']_\equiv$ is a node of $\text{lts}_F \}$ is an isomorphism between the labelled transition systems $\text{lts}_F$ and $\text{lts}_{\text{box}(F)}$. Moreover, $\text{iso}_F$ preserves the property of being in an initial or final state.

### 6 Conclusion

We extended the synchronous and asynchronous capabilities of ABC by allowing the use of multisets of actions as well as that of multisets of links. These modifications has been introduced at the level of ABC process expressions and also at the level of their associated structured operational semantics. The resulting framework, called MBC, comprises an algebra of process expressions and an algebra of nets which are consistent in the sense that an expression and the corresponding net generate isomorphic transition systems. As it was the case with PNA, MBC allows also for expressing compositionally processes with multi-way synchronisation, what was not possible with ABC.

In the process of extending PNA with asynchronous communication, the extension we still need is the recursion. Moreover, the enhancement presented in this paper is crucial when one investigates high-level extensions of such formalisms. Actually, a high-level synchronous action may encode a set of low-level actions, and similarly, a high-level communication link may correspond to a set of low-level links. The work presented here is based on multisets which are a natural generalisation of sets. Our future work will go in both directions.

### References

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PayRise</td>
<td>$\equiv ({a_r} \oplus {a_c}) \parallel ({a_r, a_r} \oplus {{b^+_n, b^+_j}}) \triangle b_d$</td>
</tr>
<tr>
<td></td>
<td>$\parallel ({{a_r, a_r} : {{b^+_n}} \square {a_r, a_r}} \oplus {f})} ) \text { sc a}_r \text { sc a}_c$</td>
</tr>
<tr>
<td>${a_r} \rightarrow {a_r}$</td>
<td>${a_r} \oplus {a_c} \parallel ({a_r, a_r} \oplus {{b^-_n, b^-_j}}) \triangle b_d$</td>
</tr>
<tr>
<td></td>
<td>$\parallel ({{a_r, a_r} : {{b^-_n}} \square {a_r, a_r}} \oplus {a_f})} ) \text { sc a}_r \text { sc a}_c$</td>
</tr>
<tr>
<td>${a_r} \rightarrow {a_r}$</td>
<td>${a_r} \oplus {a_c} \parallel ({a_r, a_r} \oplus {{b^-_n, b^-_j}}) \triangle b_d$</td>
</tr>
<tr>
<td></td>
<td>$\parallel ({{a_r, a_r} : {{b^-_n}} \square {a_r, a_r}} \oplus {a_f})} ) \text { sc a}_r \text { sc a}_c$</td>
</tr>
<tr>
<td>${a_r} \rightarrow {a_r}$</td>
<td>${a_r} \oplus {a_c} \parallel ({a_r, a_r} \oplus {{b^-_n, b^-_j}}) \triangle b_d$</td>
</tr>
<tr>
<td></td>
<td>$\parallel ({{a_r, a_r} : {{b^-_n}} \square {a_r, a_r}} \oplus {a_f})} ) \text { sc a}_r \text { sc a}_c$</td>
</tr>
<tr>
<td>${a_r} \rightarrow {a_r}$</td>
<td>${a_r} \oplus {a_c} \parallel ({a_r, a_r} \oplus {{b^-_n, b^-_j}}) \triangle b_d$</td>
</tr>
<tr>
<td></td>
<td>$\parallel ({{a_r, a_r} : {{b^-_n}} \square {a_r, a_r}} \oplus {a_f})} ) \text { sc a}_r \text { sc a}_c$</td>
</tr>
<tr>
<td>$({})$</td>
<td>${a_r} \oplus {a_c} \parallel ({a_r, a_r} \oplus {{b^-_n, b^-_j}}) \triangle b_d$</td>
</tr>
<tr>
<td></td>
<td>$\parallel ({{a_r, a_r} : {{b^-_n}} \square {a_r, a_r}} \oplus {a_f})} ) \text { sc a}_r \text { sc a}_c$</td>
</tr>
<tr>
<td>${a_r} \rightarrow {a_r}$</td>
<td>${a_r} \oplus {a_c} \parallel ({a_r, a_r} \oplus {{b^-_n, b^-_j}}) \triangle b_d$</td>
</tr>
<tr>
<td></td>
<td>$\parallel ({{a_r, a_r} : {{b^-_n}} \square {a_r, a_r}} \oplus {a_f})} ) \text { sc a}_r \text { sc a}_c$</td>
</tr>
<tr>
<td>$(a_f) \rightarrow (a_f)$</td>
<td>${a_r} \oplus {a_c} \parallel ({a_r, a_r} \oplus {{b^-_n, b^-_j}}) \triangle b_d$</td>
</tr>
<tr>
<td></td>
<td>$\parallel ({{a_r, a_r} : {{b^-_n}} \square {a_r, a_r}} \oplus {a_f})} ) \text { sc a}_r \text { sc a}_c$</td>
</tr>
<tr>
<td>$(a_f) \rightarrow (a_f)$</td>
<td>${a_r} \oplus {a_c} \parallel ({a_r, a_r} \oplus {{b^-_n, b^-_j}}) \triangle b_d$</td>
</tr>
<tr>
<td></td>
<td>$\parallel ({{a_r, a_r} : {{b^-_n}} \square {a_r, a_r}} \oplus {a_f})} ) \text { sc a}_r \text { sc a}_c$</td>
</tr>
<tr>
<td>$\equiv \text{PayRise}$</td>
<td></td>
</tr>
</tbody>
</table>

**Table 5.** Execution scenario I.
\[
\begin{align*}
E1, \text{IPAR1} & \quad \equiv \quad (\{a_r\} \oplus \{a_c\}) \parallel ((\{a_r, a_y\} \oplus \{\bar{a}_c\} \{b_y^+, b_y^-\}) \ldots ) \quad \text{sc} \quad a_r, \quad \text{sc} \quad a_c \\
\quad \equiv \quad (\{a_r\} \oplus \{a_c\}) \parallel ((\{a_r, a_y\} \oplus \{\bar{a}_c\} \{b_y^+, b_y^-\}) \ldots ) \quad \text{sc} \quad a_r, \quad \text{sc} \quad a_c \\
\quad \equiv \quad (\{a_r\} \oplus \{a_c\}) \parallel ((\{a_r, a_y\} \oplus \{\bar{a}_c\} \{b_y^+, b_y^-\}) \ldots ) \quad \text{LA, LOP} \\
\quad \equiv \quad (\{a_r\} \oplus \{a_c\}) \parallel ((\{a_r, a_y\} \oplus \{\bar{a}_c\} \{b_y^+, b_y^-\}) \ldots ) \quad \text{LSC, LTIIE} \\
\quad \equiv \quad (\{a_r\} \oplus \{a_c\}) \parallel ((\{a_r, a_y\} \oplus \{\bar{a}_c\} \{b_y^+, b_y^-\}) \ldots ) \quad \text{IC2R} \\
\quad \equiv \quad (\{a_r\} \oplus \{a_c\}) \parallel ((\{a_r, a_y\} \oplus \{\bar{a}_c\} \{b_y^+, b_y^-\}) \ldots ) \quad \text{IT2, IT3} \\
\quad \equiv \quad (\{a_r\} \oplus \{a_c\}) \parallel ((\{a_r, a_y\} \oplus \{\bar{a}_c\} \{b_y^+, b_y^-\}) \ldots ) \quad \text{LA, LOP, LSC}
\end{align*}
\]

Table 6. Execution scenario II.