Box Calculus with Coloured Buffers

Cécile Bui Thanh, Hanna Klaudel, Franck Pommereau

Box Calculus with Coloured Buffers

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 with multiway communication* (MBC), a formalism suitable for modelling compositionally distributed systems using both synchronous and asynchronous communication and a number of control flow operators. MBC is composed of two semantically consistent models: an algebra of low-level Petri nets and an associate algebra of process terms whose constants and operators directly correspond to the Petri net ones.
In this paper, we extend the Petri net algebra of MBC by allowing the communication buffers to carry coloured tokens which can be used for both synchronous and asynchronous communications. In asynchronous ones, coloured tokens can be used through high-level links making easy, for instance, the representation of program variables (*e.g.*, counters), allowing a compact representation of large (possibly infinite) systems. In synchronous ones, coloured tokens may be used as parameters of synchronising actions, allowing a simple and compositional expression of interprocess communication. We also extend the MBC process terms and their associate structured operational semantics. The obtained high-level framework, called *box calculus with data*, BCD, is coherent with respect to MBC through the unfolding operation. Moreover, the resulting algebra of BCD terms is consistent with the net algebra in the sense that a term and the corresponding net generate isomorphic transition systems.
Keywords. Coloured Petri nets, process algebra, synchronous and asynchronous communication, structured operational semantics.

1 Introduction

The framework within the present paper is set, is 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 calculus* (ABC [6]) which introduced asynchronous communication links [9] and then to *asynchronous box calculus with multiway communication* (MBC [5]). The MBC model is composed of an algebra of process terms with a fully compositional translation into labelled Petri nets (called *boxes*). As the main result, it was shown that the semantics of a term obtained by the operational rules (SOS) was equivalent to the step sequence semantics of the associated Petri net.

The operators of the MBC model relevant for this paper are the following: *sequence* $E_1 \fatsemi E_2$ (the execution of the process E_1 is followed by that of E_2); *choice* $E_1 \Box E_2$ (either E_1 or E_2 can be executed); *parallel composition* $E_1 \| E_2$ (E_1 and E_2 can be executed concurrently); *iteration* $E_1 \circledast E_2$ (E_1 can be executed an arbitrary number of times, and then is followed by E_2); *buffer restriction* $E \,\mathsf{tie}\, b$ (the buffer b and the related asynchronous links become private to E) and *scoping* $E \,\mathsf{sc}\, a$ (all multi-way synchronisations involving the actions a or $\hat{a}$ are enforced and the synchronising events may no longer be executed independently). Some of these operations are illustrated in figure 1.

A basic process term is composed of a multiaction (multiset of actions) and a multilink (multiset of links); one of them or both may be empty. For instance, $\{a_1, \widehat{a_2}\}\{b_1^-, b_2^+\}$, where a_1 and $\widehat{a_2}$ are actions used for synchronisation (through the operator of scoping) while b_1^- and b_2^+ are

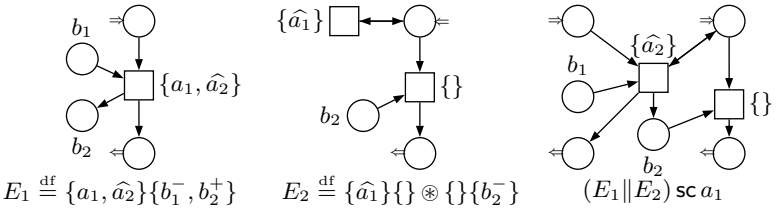

Fig. 1. Example of MBC terms and (unmarked) boxes.

asynchronous links expressing the receiving of a resource from the buffer b_1 and the sending to the buffer b_2. Such a term may be executed if it is in a context with sufficiently many resources in buffers in order to satisfy all the receiving links. Its execution removes a resource from the buffer b_1, produces one in the buffer b_2, and emits the multiaction $\{a_1, \widehat{a_2}\}$. A basic net corresponding to such a term has one transition labelled $\{a_1, \widehat{a_2}\}$ connected to one input and one output place and to two buffer places labelled b_1 and b_2, see figure 1. The receiving link b_1^- is represented by an arc going from the buffer place b_1 to the transition and the sending link b_2^+ is represented by the arc from the transition to the buffer place labelled b_2. Notice that the buffers in MBC may only carry ordinary black tokens $\bullet$.

1.1 An extension with coloured buffer

The motivation of the extension proposed in this paper is the will of introducing in MBC a number of high-level features intended to ease the modelling of complex systems manipulating data such as program variables. Such data may be expressed in MBC but in a very inefficient manner (from the human point of view) because of the low-level aspects of the model. For instance, in order to model a counter variable, one have to handle an MBC buffer for each possible counter value. In our example below, it would lead to produce an infinite net since buffers may carry any value in $\mathbb{N}$. This problem may be solved by considering high-level buffers and some related high-level features. This extension still allows to have a syntactic domain of terms (a process algebra), as well as the associated semantics through both a class of composable Petri nets with their execution rules, and a set of structured operational rules (SOS). It turns out that the obtained class of Petri nets was shown sufficient to model the semantics of the high-level parallel specification language B(PN)^2 [8]. It may also be used in order to express a Petri net semantics of π-calculus [10,7].

In the extension proposed here, the changes concern the syntactical level (terms), the associated SOS rules, as well as the Petri net level (boxes). The main change at the syntactical level concerns the form of atomic terms which are now triples $\alpha\beta\gamma$, where α is a multiset of synchronous high-level actions, β is a multiset of asynchronous high-level links and γ is a set of boolean expressions, called a *guard*. For instance, an atomic term may be $\{a(x)\}\{b^+(y), b^+(y), b^-(2)\}\{x \leq y\}$ denoted $\langle a(x) \mid b^+(y, y), b^-(2) \mid x \leq y\rangle$. We illustrate our extension using an example of a producer-consumer system based on two process terms:

$$\textsc{Prod} \stackrel{\text{df}}{=} \big(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b_p(0) \circledast \langle a(n) \mid b_p^-(n) \mid \top\rangle\big)\,\mathsf{tie}\,b_p\ ,$$

$$\textsc{Cons} \stackrel{\text{df}}{=} \big(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x, x) \mid \top\rangle.b_c(0) \circledast \langle \widehat{a}(m) \mid b_c^-(k) \mid m{-}k \leq 4\rangle\big)\,\mathsf{tie}\,b_c\ .$$

The producer process repetitively generates random resources on a buffer b (through the link $b^+(x)$, where x is a free variable), updates the private counter of produced resources b_p (through the links $b_p^-(n), b_p^+(n{+}1)$) and emits a multiaction $\{a_p\}$. The counter b_p is initialised to 0 which is visible in the term as $.b_p(0)$. The producer process may terminate at any time; it removes then the number of produced resources through the link $b_p^-(n)$, and generates the corresponding multiaction, for

instance $\{a(6)\}$ if $n = 6$. The consumer process repetitively consumes a pair of the same value from the buffer b (through the link $b^-(x,x)$), updates the private counter of consumed resources b_c (initialised to 0) and generates the multiaction $\{a_c\}$. It may terminate if the number of resources left (produced but not yet consumed) is less or equal to 4 (which is checked through the link $b_c^-(k)$ and the guard $m-k \leq 4$, where m is the number of produced resources retrieved through the action $\widehat{a}(m)$). The nets (*boxes*) of these processes are shown in figure 2, where the open buffer places are identified by the label b and the closed buffer places by the labels ${b_p}^\diamond$ and ${b_c}^\diamond$ (they correspond to the buffers b_p and b_c after the application of the buffer restriction $\mathsf{tie}\, b_p$ and $\mathsf{tie}\, b_c$).

The system where the processes Prod and Cons operate in parallel is:

$$\textsc{Par} \stackrel{\text{df}}{=} \textsc{Prod} \parallel \textsc{Cons} \quad .$$

However, this system does not allow yet for synchronous communication between the processes (through the conjugated actions $a(n)$ and $\widehat{a}(m)$). This may be achieved by applying the scoping w.r.t. a, leading to the term:

$$\textsc{Sys} \stackrel{\text{df}}{=} (\textsc{Prod} \parallel \textsc{Cons})\, \mathsf{sc}\, a \quad .$$

The boxes of Par and $\overline{\textsc{Sys}}$ (which is the initial state of Sys) are presented in figure 2. They are all the translation of their process terms (themselves called the *BCD terms* or just *terms*).

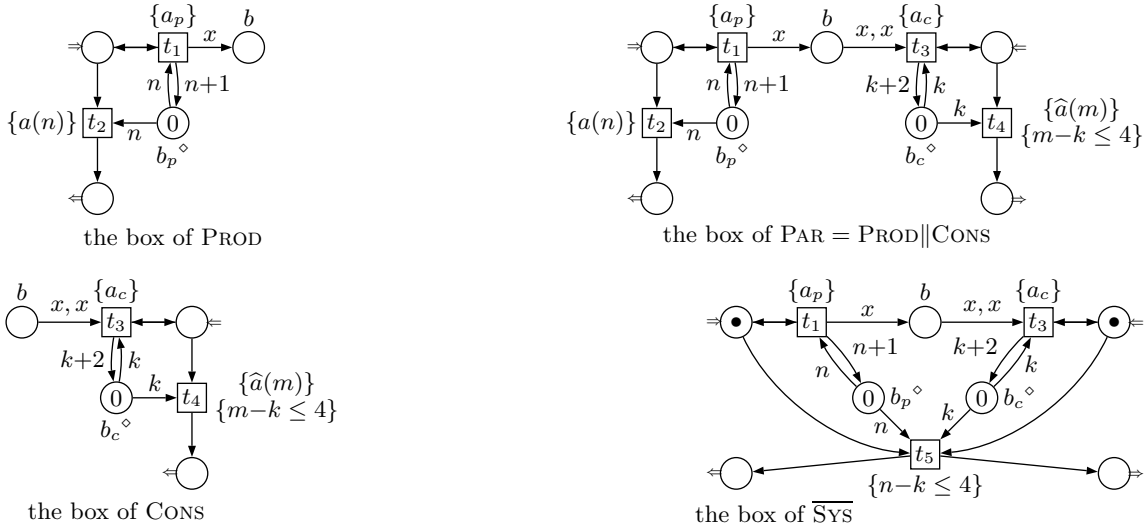

Fig. 2. Example of BCD boxes involved in the producer-consumer example. The buffer places may carry tokens in $\mathbb{N}$ while the other places may only hold $\bullet$.

Notice that, in a box, places have a status (*e.g.*, entry, exit, internal, open buffer or closed buffer) while transitions are labelled by multiactions and guards. The multilinks present in the term are represented in the box as arcs between a transition and the corresponding buffer places, labelled by the arguments of the links. Each binary operator merges the open buffer places having the same label (making asynchronous communication effective). The scoping w.r.t. a applied to the box of Par produces the new transition t_5 (visible in the box of Sys) whose empty multiaction is omitted in the figure, and removes from the net the transitions involving a or $\widehat{a}$ in their labels (t_2 and t_4).

In this paper, empty multiactions, guards which are always true and annotations $\{\bullet\}$ on the arcs are omitted in the figures, as well as disconnected unmarked buffer places. The entry places

are indicated by an incoming ⇒ and the exit places by an outgoing one, the internal places have no annotation and the buffer places are identified by the label.

1.2 Structured operational semantics

The operational semantics of terms is given through SOS rules in Plotkin's style [12]. However, instead of expressing the evolutions through rules modifying the structure of the terms, like $a.E \xrightarrow{a} E$ in CCS [11], 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)terms. This is illustrated in figure 2, where the net on the bottom right represents the initial state of the system specified by Sys, which corresponds to the term $\overline{\textsc{Sys}}$.

There are two kinds of SOS rules: equivalence rules specifying when two distinct terms denote the very same state, *e.g.*,

$$\overline{\textsc{Prod} \parallel \textsc{Cons}} \equiv \overline{\textsc{Prod}} \parallel \overline{\textsc{Cons}}$$

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

$$\overline{\langle a_p | b_p^-(n), b_p^+(n{+}1), b^+(x) | \top \rangle . b_p(0)} \xrightarrow{\{(\{a_p\}, \{n \mapsto 0, x \mapsto 3\})\}} \underline{\langle a_p | b_p^-(n), b_p^+(n{+}1), b^+(x) | \top \rangle . b_p(1) . b(3)}$$

In order to provide more intuition about the way the BCD algebra is used, we give an example of evolution (or execution scenario) of the system $\overline{\textsc{Sys}}$. To do so, we use labelled step sequences as a formal device for capturing concurrent behaviours (a labelled step being a multiset of multiactions associated to bindings, which map values to variables).

Consider the box N of $\overline{\textsc{Sys}}$ shown in figure 2 and the following evolution:

- the producer generates the resources 2, 5 and 2 in a row (this is modelled by three firings of the transition t_1);
- the consumer consumes two resources 2 (transition t_3) and the producer generates simultaneously the resource 4 (transition t_1);
- the producer and consumer terminate (transition t_5).

Such a scenario corresponds to the labelled step sequence

$$\{(\{a_p\}, \{n \mapsto 0, x \mapsto 2\})\}\,\{(\{a_p\}, \{n \mapsto 1, x \mapsto 5\})\}\,\{(\{a_p\}, \{n \mapsto 2, x \mapsto 2\})\}$$
$$\{(\{a_c\}, \{k \mapsto 0, x \mapsto 2\}), (\{a_p\}, \{n \mapsto 3, x \mapsto 4\})\}\,\{(\{\}, \{n \mapsto 4, k \mapsto 2\})\}$$

leading from N to a box N', where N' is N with two tokens 5 and 4 in the buffer place b, one token in each of the exit places, and no token elsewhere. At the BCD term level, the same labelled step sequence leads from $\overline{\textsc{Sys}}$ to $\textsc{Sys}'$, such that the box of $\textsc{Sys}'$ is N', the scenario above is detailed in the section 7using the operational rules.

2 Preliminary definitions

2.1 Values, variables, structures and expressions

Throughout the paper we will use structures involving variables and values which can be represented and used in BCD processes. Thus, we assume that there exists a set $\mathbb{D}$ containing all the *data values* (or just *values*), usually denoted by v; in particular, $\mathbb{D}$ contains the set of natural numbers, the black token $\bullet$, boolean values $\top$ and $\bot$, etc. We also assume the existence of a set $\mathbb{V}$ of *variables*. The set of all the variables appearing in an arbitrary expression e is denoted $\mathsf{var}(e)$.

The structures involving values, variables and expressions are in particular tuples, sets or multisets. A *multiset* over a set X is a function $\mu : X \to \mathbb{N}$. We denote by $\mathsf{mult}(X)$ the set of all finite multisets μ over X, *i.e.*, those satisfying $\sum_{x\in X} \mu(x) < \infty$. We will write $\mu \leq \mu'$ if the domain X of μ is included in that of μ', and if $\mu(x) \leq \mu'(x)$, for all $x \in X$. An element $x \in X$ belongs to μ, denoted $x \in \mu$, if $\mu(x) > 0$. The sum and difference of multisets, and the multiplication by a non-negative integer are respectively denoted by $+$, $-$ and $\cdot$ (the difference will only be applied when the second argument is smaller or equal to the first one). A subset of X may be treated as a multiset over X, by identifying it with its characteristic function, and a singleton set can be identified with its sole element. A finite multiset μ over X may be written as $\sum_{x\in X} \mu(x) \cdot x$ 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 μ 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, we consider several kinds of functions defined on values and variables. All these functions are naturally extended to more complex structures (like expressions, tuples, sets, multisets, multisets of tuples, etc.) by applying them component-wise to each element of the structure. For instance, the image of a set of elements is the set of the images of the elements. Moreover, the application of a function f to a tuple $(x_1, \ldots, x_n)$ is written $f(x_1, \ldots, x_n)$ instead of $f((x_1, \ldots, x_n))$.

The expressions we consider are built over values in $\mathbb{D}$, variables in $\mathbb{V}$ and a set of suitable operators. An expression which always evaluates into a value in $\mathbb{D}$ is called a *data expression* and one which evaluates to a truth value in $\{\bot, \top\}$ is called a *boolean expression*. We will denote the set of all data expressions by $\mathbb{E}$. Moreover, we shall assume that a set (or a multiset) of boolean expressions is also a boolean expression whose truth value corresponds to truth value of the conjunction of its constituting expressions.

A *substitution* is a function $\delta : \mathbb{V} \to \mathbb{D} \cup \mathbb{V}$ extended on the domain $\mathbb{D} \cup \mathbb{V}$ by the identity on $\mathbb{D}$. We denote by Δ the set of all substitutions, and by δ_{id} the identity on $\mathbb{D} \cup \mathbb{V}$.

An expression e may be evaluated if we give first a binding of the variables occurring in e (*i.e.*, belonging to $\mathsf{var}(e)$). Formally, a *binding* σ is a substitution such that $\sigma(x) \in \mathbb{D}$ for all $x \in \mathbb{V}$, which associates a value to each variable. We shall denote $\sigma \stackrel{\mathrm{df}}{=} \{x \mapsto 2\}$ a binding which associates 2 to x if the values associated to the other variables are not important in the context, for instance, for the expression $x + 1$. We denote by $\sigma(e)$ the expression e in which each variable has been substituted by the corresponding value, *e.g.*, in our case $\sigma(x + 1) = 2 + 1$. The evaluation of e with σ is then denoted by $\mathsf{eval}(\sigma(e))$; in our case $\mathsf{eval}(\sigma(x + 1)) = 3$.

A *renaming* ρ is a substitution which is a bijection on $\mathbb{V}$ (and the identity on $\mathbb{D}$), usually denoted by the set of variable changes it makes. For instance, we shall denote $\rho = \{x \mapsto y, y \mapsto z, z \mapsto x\}$ if ρ is such that $\rho(x) = y$, $\rho(y) = z$, $\rho(z) = x$ and $\rho(w) = w$ for all $w \in \mathbb{V} \setminus \{x, y, z\}$.

An *unifier* η for two expressions e_1 and e_2 in $\mathbb{D} \cup \mathbb{V}$ is here a substitution such that $\eta(e_1) = \eta(e_2)$ and:

- if $\{e_1, e_2\} \subseteq \mathbb{V}$ then $\{\eta(e_1), \eta(e_2)\} \subseteq (\mathsf{var}(e_1) \cup \mathsf{var}(e_2))$;
- for all $z \in \mathbb{D} \cup \mathbb{V} \setminus (\mathsf{var}(e_1) \cup \mathsf{var}(e_2))$ we have $\eta(z) = z$.

This corresponds to what is usually called a *most general unifier*, which is unique up to renaming of variables. The unifiers will generally be used in the context of tuples of expressions.

2.2 Distinguished sets

We introduce now some notions and notations needed to formalise the high-level features of our extension.

We assume that there is a finite set $\mathbb{S}$ of *action symbols* representing synchronous interface activities used, in particular, to model handshake communication. Each action symbol $a \in \mathbb{S}$ has the arity $\mathsf{ar}(a)$ corresponding to the number of parameters which are used during a synchronous communication. We also assume that, for every $a \in \mathbb{S}$, there exists an $\widehat{a}$ in $\mathbb{S}$ such that $\widehat{\widehat{a}} = a$ and $\mathsf{ar}(a) = \mathsf{ar}(\widehat{a})$. An *action* is composed of an action symbol and a number of parameters in $\mathbb{V} \cup \mathbb{D}$,

for instance, if $\mathsf{ar}(a) = 2$, then $a(x, 1)$ and $\widehat{a}(2, y)$ are examples of actions. Moreover, if $\mathsf{ar}(a') = 0$, we shall write a' instead of $a'()$, omitting the useless parentheses. The set of all possible actions with parameters is $\mathbb{A} \stackrel{\text{df}}{=} \{a(\widetilde{x}) \mid a \in \mathbb{S}, \widetilde{x} \in (\mathbb{D} \cup \mathbb{V})^{\mathsf{ar}(a)}\}$.

We consider also a finite set $\mathbb{B}$ of *buffer symbols* (or *buffers*) intended for asynchronous inter-process communications. Each buffer $b \in \mathbb{B}$ is assigned a non-empty type $\mathsf{type}(b) \subseteq \mathbb{D}$, representing all the values it can hold.[1] Communications through a buffer b are represented with *asynchronous links* (or just *links*) of the form $b^+(e)$ for sending or $b^-(e)$ for receiving (with $e \in \mathbb{E}$). Thus, we denote by $\mathbb{L} \stackrel{\text{df}}{=} \{b^+(e), b^-(e) \mid b \in \mathbb{B}, e \in \mathbb{E}\}$ the set of all possible links. Moreover, in process terms, we will use the notations $b(v)$, with $v \in \mathsf{type}(b)$, to denote the presence of a token v in the buffer b.

Finally, we assume there exists a set of *statuses of places* defined as $\mathbb{P} \stackrel{\text{df}}{=} \{\mathsf{e}, \mathsf{i}, \mathsf{x}\} \uplus \mathbb{B} \uplus \mathbb{B}^\diamond$, where $\mathbb{B}^\diamond \stackrel{\text{df}}{=} \{b^\diamond \mid b \in \mathbb{B}\}$. Each element of $\mathbb{B}^\diamond$ has a type such that for all $b \in \mathbb{B}$, $\mathsf{type}(b^\diamond) \stackrel{\text{df}}{=} \mathsf{type}(b)$. A place status e denotes an *entry place* while i denotes an *internal place* and x an *exit place*; together, the entry, internal and exit places are called *control places*. A status $b \in \mathbb{B}$ corresponds to an *open buffer place* and $b^\diamond \in \mathbb{B}^\diamond$ to a *closed buffer place* (a buffer place after an application of the buffer restriction operation).

2.3 Synchronous communication

We need a device to express, *e.g.*, when concurrent events may synchronise. We use for that purpose *interface functions* $\varphi\colon \mathsf{mult}(\mathsf{mult}(\mathbb{A})) \to \mathsf{mult}(\mathbb{A}) \times \Delta$, interpreted as a subset of $\mathsf{mult}(\mathsf{mult}(\mathbb{A})) \times \mathsf{mult}(\mathbb{A}) \times \Delta$. For $(\Gamma, \alpha, \delta) \in \varphi$, we define $\varphi(\Gamma) \stackrel{\text{df}}{=} \delta(\alpha)$.

An example of such a function is $\varphi_{\text{id}} \stackrel{\text{df}}{=} \{(\{\alpha\}, \alpha, \delta_{\text{id}}) \mid \alpha \in \mathsf{mult}(\mathbb{A})\}$. This function is used when no synchronous interface change has to be performed.

The interface function $\varphi_{\mathsf{sy}\,a}$ is used in order to specify when concurrent events, labelled by multi-actions $\alpha_1, \ldots, \alpha_k$, can synchronise with respect to $a \in \mathbb{S}$. It is possible if there exists a $(\{\alpha_1, \ldots, \alpha_k\}, \alpha, \delta)$ in $\varphi_{\mathsf{sy}\,a}$ and leads to create a new event labelled $\varphi(\{\alpha_1, \ldots, \alpha_k\})$. Formally, $\varphi_{\mathsf{sy}\,a}$ is defined as the smallest set containing φ_{id} and such that, if $\{(\Gamma_1, \alpha_1 + \{a(\widetilde{x_1})\}, \delta_1)$ and $(\Gamma_2, \alpha_2 + \{\widehat{a}(\widetilde{x_2})\}, \delta_2)\}$ belong to $\varphi_{\mathsf{sy}\,a}$ are such that

- there exists an unifier η for $\delta_1(\widetilde{x_1})$ and $\delta_2(\widetilde{x_2})$, and
- $\mathsf{var}(\Gamma_1) \cap \mathsf{var}(\Gamma_2) = \mathsf{var}(\alpha_1) \cap \mathsf{var}(\alpha_2) = \mathsf{var}(\widetilde{x_1}) \cap \mathsf{var}(\widetilde{x_2}) = \varnothing$,

then $(\Gamma_1 + \Gamma_2, \alpha_1 + \alpha_2, \eta \circ \delta_1 \circ \delta_2)$ also belongs to $\varphi_{\mathsf{sy}\,a}$. Notice that, by construction, we always have $\delta_1 \circ \delta_2 = \delta_2 \circ \delta_1$ because the variable changes performed by δ_i (for $i \in \{1, 2\}$) only concern elements of $\mathsf{var}(\Gamma_i) \cup \mathsf{var}(\alpha_i) \cup \mathsf{var}(\widetilde{x_i})$; thus, for $j \neq i$ we have $\delta_i(x) \neq x \Rightarrow \delta_j(x) = x$ for all $x \in \mathbb{V}$.

For instance, $(\{\{a_1(x), a_1(3), a_2(x)\}, \{\widehat{a_1}(y), a_3\}, \{\widehat{a_1}(5)\}\}, \{a_2(x), a_3\}, \{x \mapsto 5, y \mapsto 3\}) \in \varphi_{\mathsf{sy}\,a_1}$ means that the multiactions $\{a_1(x), a_1(3), a_2(x)\}$, $\{\widehat{a_1}(y), a_3\}$ and $\{\widehat{a_1}(5)\}$ can synchronise in a three-way communication,[2] leading to the multiaction $\{a_2(x), a_3\}$ on which the substitution $\{x \mapsto 5, y \mapsto 3\}$ has to be applied in order to give the same parameters to the matching actions. The fact that $\{\{a_1(x), \widehat{a_1}(y)\}, \{a_2(z)\}\} \notin \mathsf{dom}\,\varphi_{\mathsf{sy}\,a_1}$ means that multiactions $\{a_1(x), \widehat{a_1}(y)\}$ and $\{a_2(z)\}$ cannot synchronise together. Such a function will be used to enforce CCS-like synchronisations, but with no limitation on the number of simultaneously performed synchronisations.

Another useful interface function is $\varphi_{\mathsf{sc}\,a}$, for $a \in \mathbb{S}$, which allows to setup all synchronous communications w.r.t. a but forbids the execution of events labelled by multiactions still involving actions with a or $\widehat{a}$. Formally, $\varphi_{\mathsf{sc}\,a} \stackrel{\text{df}}{=} \{(\Gamma, \alpha, \delta) \in \varphi_{\mathsf{sy}\,a} \mid \forall \widetilde{x} \in (\mathbb{D} \cup \mathbb{V})^{\mathsf{ar}(a)}, a(\widetilde{x}) \notin \alpha \text{ and } \widehat{a}(\widetilde{x}) \notin \alpha\}$. This interface function is an extension of that used in MBC to the context of actions with parameters. One can check that if $\mathsf{ar}(a) = 0$ for all $a \in \mathbb{S}$, $\varphi_{\mathsf{sc}\,a}$ coincides with what has been considered in MBC [5].

[1] In the example given in the introduction, we assumed that $\mathsf{type}(b) = \mathsf{type}({b_p}^\diamond) = \mathsf{type}({b_c}^\diamond) = \mathbb{N}$.

[2] $a_1(x)$ synchronising with $\widehat{a_1}(5)$ and $a_1(3)$ synchronising with $\widehat{a_1}(y)$, from what we can deduce the substitution

3 Labelled nets

A *(marked) labelled net* is, in the present framework, a tuple $N \stackrel{\text{df}}{=} (S, T, \lambda, M)$ such that:

- S and T are disjoint sets of respectively *places* and *transitions*;
- λ is a *labelling* for places, transitions and arcs (elements of $(S \times T) \cup (T \times S)$). For a place $s \in S$ we have $\lambda(s) \in \mathbb{P}$. For a transition $t \in T$, $\lambda(t)$ is an interface function φ or a pair $\alpha(t)\gamma(t)$ where $\alpha(t)$ is a *multiaction* in $\mathsf{mult}(\mathbb{A})$ and $\gamma(t)$ is a *guard*, *i.e.*, a boolean expression. The labelling of an arc is a multiset of data expressions representing the tokens which may flow on the arc, an empty multiset denoting the absence of arc;
- each place $s \in S$ has a *type* $\mathsf{type}(s)$, representing the tokens it can hold, which is defined as follows: if $\lambda(s) = b \in \mathbb{B}$ or $\lambda(s) = b^\diamond \in \mathbb{B}^\diamond$ then $\mathsf{type}(s) \stackrel{\text{df}}{=} \mathsf{type}(b)$; otherwise, $\mathsf{type}(s) \stackrel{\text{df}}{=} \{\bullet\}$;
- M is a *marking* function which associates to each place $s \in S$ a multiset of values which belongs to $\mathsf{mult}(\mathsf{type}(s))$ and represents the tokens held by s.

Moreover, N is *finite* if both S and T are finite sets and it is *simple* if all its arcs are labelled by $\{\bullet\}$.

A binding σ is enabling for a transition $t \in T$ if the guard of t evaluates to true, *i.e.*, $\mathsf{eval}(\sigma(\gamma(t))) = \top$, and if σ respects the types of the places, *i.e.*, for each $s \in S$, $\mathsf{eval}(\sigma(s,t)) \in \mathsf{mult}(\mathsf{type}(s))$ and $\mathsf{eval}(\sigma(t,s)) \in \mathsf{mult}(\mathsf{type}(s))$.

For a node $r \in S \cup T$, we define ${}^\bullet r \stackrel{\text{df}}{=} \{r' \in S \cup T \mid \lambda(r', r) \neq \varnothing\}$ and, similarly, $r^\bullet \stackrel{\text{df}}{=} \{r' \in S \cup T \mid \lambda(r, r') \neq \varnothing\}$. We denote by M^{c} the marking M restricted to the control places and by N^{c} the net N in which all the buffer places together with their adjacent arcs have been removed. Moreover, M_N^{e} denotes the entry (or initial) marking of N (which only has one token $\bullet$ in each entry place), while M_N^{x} denotes its exit (or final) marking (one token in each exit place).

A marking M of N is:

- *clean* if $M^{\mathsf{c}} \geq M^{\mathsf{e}} \Rightarrow M^{\mathsf{c}} = M^{\mathsf{e}}$ and $M^{\mathsf{c}} \geq M^{\mathsf{x}} \Rightarrow M^{\mathsf{c}} = M^{\mathsf{x}}$.
- *ac-free* if, for every transition t, there is a control place $s \in {}^\bullet t$ such that $M(s) < 2 \cdot \mathsf{eval}(\sigma(\lambda(s,t)))$ for every binding σ which enables t, meaning that the marking of the control places does not allow *auto-concurrency*.

To avoid ambiguity, we will sometimes decorate the various components of N with the index N; *e.g.*, T_N will denote the set of transitions of N.

3.1 Transition rule and step sequences

Let $N = (S, T, \lambda, M)$ be a labelled net. A *bounded transition* is a pair (t, σ) where t is a transition and σ is an enabling binding of t. A finite step sequence semantics of a labelled net N captures the potential concurrency in the behaviour of the system modelled by N. A finite multiset of bounded transitions $U \stackrel{\text{df}}{=} \{(t_1, \sigma_1), \ldots, (t_k, \sigma_k)\}$ is *enabled* by N if, for every place $s \in S$,

$$M(s) \geq \sum_{(t,\sigma) \in U} U(t,\sigma) \cdot \mathsf{eval}(\sigma(\lambda(s,t))) \quad .$$

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

$$M'(s) \stackrel{\text{df}}{=} M(s) - \sum_{(t,\sigma) \in U} U(t,\sigma) \cdot \mathsf{eval}(\sigma(\lambda(s,t))) + \sum_{(t,\sigma) \in U} U(t,\sigma) \cdot \mathsf{eval}(\sigma(\lambda(t,s))) \quad .$$

We will denote this by $N[U\rangle N'$, where N' is N with the marking changed to M'. Labelled steps may be obtained then through the formula[3]

$$\alpha(U) \stackrel{\text{df}}{=} \sum_{(t,\sigma)\in U} U(t,\sigma)\cdot(\alpha(t),\sigma) \quad .$$

This allows one to translate various behavioural notions defined in terms of multisets of transitions into notions based on multisets of transition labels (*labelled steps*). Although we will use the same term "step" to refer both to a transition step and to a labelled step, 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 and the same enabling binding.

A *step sequence* of N is a (possibly empty) sequence of steps, $\omega = U_1 \cdots U_k$, such that there are nets $N_1, \ldots, N_k$ satisfying $N[U_1\rangle N_1[U_2\rangle N_2 \cdots [U_k\rangle N_k$. We will denote this by $N[\omega\rangle N_k$ or $N_k \in [N\rangle$, and call N_k *derivable* from N and its marking M_{N_k} *reachable* from M_N (with the convention that $N[\omega\rangle N$ if $k = 0$, *i.e.*, if ω is the empty step sequence). The definition of a *labelled step sequence* of N is similar.

3.2 Equality of labelled nets

Labelled nets have an important property that the variables appearing around a transition (*i.e.*, in its label and in the annotations of the adjacent arcs) have only a local meaning. That means in particular, that one may rename them consistently without changing the behaviour of the net.

More formally, if $N = (S, T, \lambda, M)$ is a labelled net and $t \in T$, we define the *area* of t as the set of all the annotations on t and its adjacent arcs, *i.e.*, $\{\lambda(t)\} \cup \{\lambda(s,t), \lambda(t,s) \mid s \in S\}$. We denote by $\mathsf{var}(t)$ the set of all the variables appearing in the area of t, *i.e.*,

$$\mathsf{var}(t) \stackrel{\text{df}}{=} \mathsf{var}(\lambda(t)) \cup \{\mathsf{var}(\lambda(s,t)) \mid s \in S\} \cup \{\mathsf{var}(\lambda(t,s)) \mid s \in S\} \quad .$$

Let σ be an enabling binding of t such that $(t,\sigma) \in \mathsf{enabled}(N)$ and let ρ be a renaming. By the definition of the enabling, we may always apply ρ to the area of t and the only consequence will be that now $(t, \sigma \circ \rho)$ will belong to $\mathsf{enabled}(N_\rho)$ where N_ρ is N in which the area of t have been renamed with ρ.

This property has two important consequences. (1) We may always rename the areas of transitions in such a way that the sets of their variables are pairwise disjoint; for technical reasons we will consider only such labelled nets, which are called *area disjoint*. (2) We consider as equal the labelled nets which are isomorphic up to renaming of their nodes (places and transitions) and up to the renaming of variables in the areas of their transitions.

4 An algebra of boxes

Our target model is a class of labelled nets called *boxes*. In order 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 boxes.

A *box* is a labelled net N such that no transition is labelled by an interface function, and N itself is:

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

[3] The guards are not considered since they all evaluate to true.

A box N is *static* (resp. *dynamic*) if $M_N^{\mathsf{c}} = \varnothing$ (resp. $M_N^{\mathsf{c}} \neq \varnothing$) and all the markings reachable from M_N^{c}, M_N^{e} or M_N^{x} in the net N^{c} are both clean and ac-free. We denote $\mathcal{B}_s$ and $\mathcal{B}_d$ the set of static and dynamic boxes, respectively.

The basic building blocks, from which other static and dynamic boxes will be constructed, are the boxes $N_{\alpha\beta\gamma}$, for $\alpha \in \mathsf{mult}(\mathbb{A})$, $\beta \in \mathsf{mult}(\mathbb{L})$ and γ a guard (a boolean expressions). Each $N_{\alpha\beta\gamma}$ 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 $b \in \mathbb{B}$. It has only one transition $v^{\alpha\beta\gamma}$ labelled by $\alpha\gamma$, which has one incoming arc labelled $\{\bullet\}$ from e and one outgoing arc to x with the same label. The other arcs correspond to the links in β and we have:

$$\lambda_{N_{\alpha\beta\gamma}}(v^{\alpha\beta\gamma}, s_b) \stackrel{\text{df}}{=} \sum_{e\in\mathbb{E}} \beta(b^+(e)) \cdot e \qquad \text{and} \qquad \lambda_{N_{\alpha\beta\gamma}}(s_b, v^{\alpha\beta\gamma}) \stackrel{\text{df}}{=} \sum_{e\in\mathbb{E}} \beta(b^-(e)) \cdot e \ ,$$

for each $b \in \mathbb{B}$. The marking of $N_{\alpha\beta\gamma}$ is empty. Two examples of such boxes are given in figure 3.

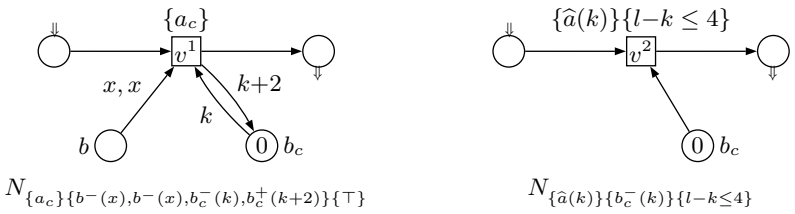

Fig. 3. Two examples of basic boxes, where the identities of the transitions are defined as $v^1 \stackrel{\text{df}}{=} v^{\{a_c\}\{b^-(x),b^-(x),b_c^-(k),b_c^+(k+2)\}\{\top\}}$ and $v^2 \stackrel{\text{df}}{=} v^{\{\widehat{a}(k)\}\{b_c^-(k)\}\{l-k\leq 4\}}$.

We will use the following *marking operators*, which modify the marking of a box N, where $b(v)$ represent a token v in the open buffer place b such that $v \in \mathsf{type}(b)$ and B is a multiset of such $b(v)$'s:

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

4.1 Operator boxes

An *operator box* Ω is an unmarked, finite, simple, ex-restricted and control-restricted labelled net with only control places (hence it is not $\mathbb{B}$-restricted) and such that every transition v is labelled by an interface function. For every operator box Ω, we will assume that its transitions $v_1, \ldots, v_n$ are implicitly ordered, and then each n-tuple of nets (or terms later on) $\mathbf{N} = (N_1, \ldots, N_n)$, will be referred to as an *Ω-tuple* (or, simply, a *tuple*); we will also use N_{v_i} to denote N_i, for $i \leq n$. The notation $\mathbf{N}$ will be used in the net substitution operation, denoted by $\Omega(\mathbf{N})$, and defined in the next section.

We will consider for BCD four groups of operator boxes (directly inherited from MBC) as described below and depicted in the figure 4.

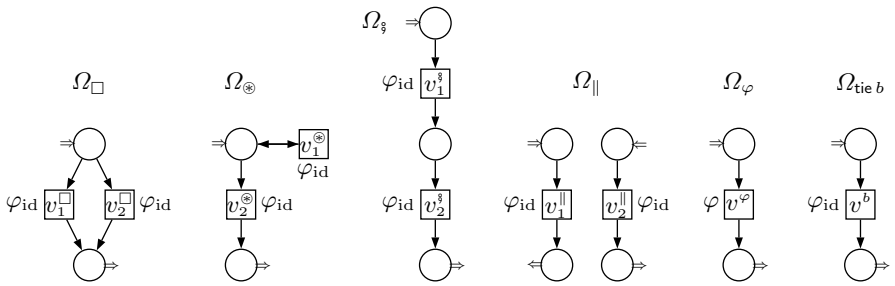

Fig. 4. The operator boxes of BCD.

Sequential operators. A *sequential* operator box Ω_{sq} is an operator box such that: no place is disconnected; there is exactly one entry place, and exactly one exit place; and, for every transition $v \in T_{\Omega_{sq}}$, $|{}^\bullet v| = |v^\bullet| = 1$ and $\lambda_{\Omega_{sq}}(v) = \varphi_{\mathrm{id}}$. That is, Ω_{sq} can be thought of as a *finite automaton* in which each transition will be substituted by a potentially complex box by the net substitution operation. The domain of application of Ω_{sq} is the set $\mathsf{dom}_{\Omega_{sq}}$ comprising all Ω_{sq}-tuples of static and dynamic boxes such that at most one box is dynamic.

Examples of sequential operator boxes are choice $\Omega_{\Box}$, iteration $\Omega_{\circledast}$ and sequence $\Omega_{\fatsemi}$ depicted in figure 4. They are all binary, with the domain of application $\mathsf{dom}_{\Omega_{\Box}} = \mathsf{dom}_{\Omega_{\circledast}} = \mathsf{dom}_{\Omega_{\fatsemi}} \stackrel{\text{df}}{=} (\mathcal{B}_s)^2 \cup (\mathcal{B}_d \times \mathcal{B}_s) \cup (\mathcal{B}_s \times \mathcal{B}_d)$. We will denote: $N_1 \,\Box\, N_2 \stackrel{\text{df}}{=} \Omega_{\Box}(N_1, N_2)$, $N_1 \circledast N_2 \stackrel{\text{df}}{=} \Omega_{\circledast}(N_1, N_2)$ and $N_1 \fatsemi N_2 \stackrel{\text{df}}{=} \Omega_{\fatsemi}(N_1, N_2)$.

Parallel composition $\Omega_{\|}$. This is also a binary operator box (see figure 4), but with the domain of application $\mathsf{dom}_{\Omega_{\|}} \stackrel{\text{df}}{=} (\mathcal{B}_s)^2 \cup (\mathcal{B}_d)^2$, so that its two operands may evolve concurrently. We will denote $N_1 \| N_2 \stackrel{\text{df}}{=} \Omega_{\|}(N_1, N_2)$.

Communication interface operators. A unary *communication interface* operator box Ω_{φ}, shown in figure 4, is parameterised by an interface function φ, and has the domain of application $\mathsf{dom}_{\Omega_{\varphi}} \stackrel{\text{df}}{=} \mathcal{B}_s \cup \mathcal{B}_d$. The role of Ω_{φ} will be to effect the change of synchronous communication interface specified by φ.

An example of such operators is the scoping, which is parameterised by an action $a \in \mathbb{S}$. We will denote $N\,\mathsf{sc}\,a \stackrel{\text{df}}{=} \Omega_{\mathsf{sc}\,a}(N)$.

Buffer restriction $\Omega_{\mathsf{tie}\,b}$. Parameterised by a buffer $b \in \mathbb{B}$, this unary operator also has the domain of application $\mathsf{dom}_{\Omega_{\mathsf{tie}\,b}} \stackrel{\text{df}}{=} \mathcal{B}_s \cup \mathcal{B}_d$. Buffer restriction will hide the b-labelled open buffer place of the box it is applied to. We will denote $N\,\mathsf{tie}\,b \stackrel{\text{df}}{=} \Omega_{\mathsf{tie}\,b}(N)$.

4.2 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 terms. 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 $\mathbb{I}$ of *basic* transition identities and a corresponding set of basic labelled trees with a single node labelled with an element of $\mathbb{I}$. All the transitions in the operator boxes are assumed to be of that kind, as well as the $v^{\alpha\beta\gamma}$ used for the basic boxes. To express more complex (unordered) finite trees used as transition identities in boxes obtained through net substitution, we will use the following linear notations:

- $v \triangleleft \mathsf{T}$, where $v \in \mathbb{I}$ is a basic transition identity and T is a finite set of finite labelled trees, denotes a tree where the trees of the set T are appended to a root labelled with v;

– $v \lhd \mathtt{t}$ denotes the tree $v \lhd \{\mathtt{t}\}$.

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. 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 Ω be a sequential or the parallel operator with transitions $v_1, \ldots, v_n$, and $\mathbf{N} = (N_1, \ldots, N_n) = (N_{v_1}, \ldots, N_{v_n})$ be a tuple of boxes in dom_Ω. Then $\Omega(\mathbf{N}) = N'$ whose components are defined as follows.

The set of transitions of N' is the set of all trees $v_i \lhd t$ (with $t \in T_{N_i}$ and $i \in \{1, \ldots, n\}$). The label of each $v_i \lhd t$ is that of t. Each internal or closed buffer place $p \in S_{N_i}$ belongs to $S_{N'}$, its label and marking are unchanged and for every transition $w \lhd t$, the label of the arc between p and $w \lhd t$ is given by:

$$\lambda_{N'}(p, w \lhd t) \stackrel{\text{df}}{=} \begin{cases} \lambda_{N_i}(p,t) & \text{if } w = v_i, \\ \varnothing & \text{otherwise,} \end{cases}$$

and similarly for $\lambda_{N'}(w \lhd t, p)$.

For every place $s \in S_\Omega$ with ${}^\bullet s = \{u_1, \ldots, u_k\}$ and $s^\bullet = \{w_1, \ldots, w_m\}$, we construct in $S_{N'}$ all the places of the form $p \stackrel{\text{df}}{=} (x_1, \ldots, x_k, e_1, \ldots, e_m)$, where each x_i is an exit place of N_{u_i}, and each e_j is an entry place of N_{w_j}. The label of p is that of s, its marking is the sum of the markings of x_1, …, x_k, e_1, …, e_m, and for every transition $w \lhd t$, the arcs label are given by:

$$\lambda_{N'}(p, w \lhd t) \stackrel{\text{df}}{=} \begin{cases} \lambda_{N_w}(x_i,t) + \lambda_{N_w}(e_j,t) & \text{if } w \in {}^\bullet s \cap s^\bullet \text{ and } w = u_i = w_j, \\ \lambda_{N_w}(x_i,t) & \text{if } w \in {}^\bullet s \setminus s^\bullet \text{ and } w = u_i, \\ \lambda_{N_w}(e_j,t) & \text{if } w \in s^\bullet \setminus {}^\bullet s \text{ and } w = w_j, \\ \varnothing & \text{otherwise,} \end{cases}$$

and similarly for $\lambda_{N'}(w \lhd t, p)$.

For every $b \in \mathbb{B}$, there is a unique b-labelled place $p^b \stackrel{\text{df}}{=} (p^b_{v_1}, \ldots, p^b_{v_n})$ in $S_{N'}$, where each $p^b_{v_i}$ is the unique b-labelled place of N_i. The marking of p^b is the sum of the markings of the $p^b_{v_i}$'s, and for each transition $w \lhd t$, the arcs labels are given by $\lambda_{N'}(p^b, w \lhd t) \stackrel{\text{df}}{=} \lambda_{N_w}(p^b_w, t)$, and similarly for $\lambda_{N'}(w \lhd t, p^b)$.

Communication interface operators. For a communication interface operator Ω_φ, the intuition behind a triple (Γ, α, δ) in φ is that some interface change can be applied to any finite multiset of transitions whose synchronous labels match the Γ. More precisely, such transitions can be synchronised to yield a new transition having the synchronous label $\varphi(\Gamma)$, performing the unification of actions parameters. (Note that, since sequential operators as well as the parallel one, use the interface function φ_{id}, no transition label is changed for them.) Hence, the application of a communication interface operator Ω_φ to an area disjoint box N results in a labelled net which is like N with the only difference that the set of transitions comprises all trees $t \stackrel{\text{df}}{=} v^\varphi \lhd \{t_1, \ldots, t_l\}$ such that $\{t_1, \ldots, t_l\} \in \mathsf{mult}(T_N)$ and there exists $(\Gamma, \alpha, \delta) \in \varphi$ where $\Gamma \stackrel{\text{df}}{=} \{\alpha_N(t_1), \ldots, \alpha_N(t_l)\}$. The label of t is $\varphi(\Gamma)\{\delta(\gamma_N(t_i)) \mid 1 \le i \le l\}$, and for a place p of $\Omega_\varphi(N)$, the arcs are labelled by $\lambda_{\Omega_\varphi(N)}(p,t) \stackrel{\text{df}}{=} \sum_{i=1}^{l} \delta(\lambda_N(p, t_i))$, and similarly for $\lambda_{\Omega_\varphi(N)}(t,p)$.

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

5 An algebra of terms

We consider an algebra of process terms over the signature:

$$\mathcal{C} \cup \{\overline{(\cdot)}, \underline{(\cdot)}\} \cup \{\,\|\,,\,\fatsemi\,,\,\Box\,,\,\circledast\} \cup \{\mathsf{sc}\,a \mid a \in \mathbb{S}\} \cup \{.b(v), \mathsf{tie}\,b \mid b \in \mathbb{B}, v \in \mathsf{type}(b)\}$$

where $\mathcal{C} \stackrel{\text{df}}{=} \{\alpha\beta\gamma \mid \alpha \in \mathsf{mult}(\mathbb{A}), \beta \in \mathsf{mult}(\mathbb{L})$ and γ is a guard$\}$ are the constants; the binary operators $\|$, $\fatsemi$, $\Box$ and $\circledast$ will be used in the infix mode; the unary operators $\mathsf{sc}\,a$, $\mathsf{tie}\,b$ and $.b(v)$ will be used in the postfix mode; and $\overline{(\cdot)}$ and $\underline{(\cdot)}$ are two positional unary operators (the position of the argument being given by the dot).

There are two classes of process terms corresponding to the static and dynamic boxes, viz. the *static* and *dynamic* terms, denoted respectively by $\mathcal{T}_s$ and $\mathcal{T}_d$. Collectively, we will refer to them as the *terms*, $\mathcal{T}$. Their syntax is given by:

$$\begin{array}{lll}
\mathcal{T}_s & E ::= & \alpha\beta\gamma \mid E\,\mathsf{sc}\,a \mid E\,\mathsf{tie}\,b \mid E.b(v) \mid \ E\|E \ \mid E \Box E \mid \ E \fatsemi E \ \mid E \circledast E \\
\mathcal{T}_d & D ::= & \overline{E} \ \mid \ \underline{E} \ \mid D\,\mathsf{sc}\,a \mid D\,\mathsf{tie}\,b \mid D.b(v) \mid \ D\|D \ \mid D \Box E \mid E \Box D \\
 & & D \fatsemi E \mid E \fatsemi D \mid D \circledast E \mid E \circledast D
\end{array}$$

where $\alpha\beta\gamma \in \mathcal{C}$, $a \in \mathbb{S}$, $b \in \mathbb{B}$ and $v \in \mathsf{type}(b)$.

Essentially, a term 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 $.b(v)$'s). Thus, a term $\overline{E}$ 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. An atomic term encodes as many executions as there are distinct enabling bindings for it. A binding σ is enabling for a term $\alpha\beta\gamma$ if $\mathsf{eval}(\sigma(\gamma)) = \top$ and if for each $b^-(e)$ and $b^+(e)$ in β, $\mathsf{eval}(\sigma(e)) \in \mathsf{type}(b)$.

We will use F to denote any static or dynamic term. We also use the notations $\lfloor F \rfloor$ and $\llbracket F \rrbracket$ yielding static terms, where $\lfloor F \rfloor$ is F with all occurrences of $\overline{(\cdot)}$ and $\underline{(\cdot)}$ removed, and $\llbracket F \rrbracket$ is $\lfloor F \rfloor$ with all occurrences of $.b(v)$ removed. Note that we do not need terms of the form $F.B$ since $F.\{b(v_1), \ldots, b(v_k)\}$ would be equivalent to $F.b(v_1) \cdots .b(v_k)$ (but such terms can be used as a convenient shorthand). Note that the $.b(v)$ notation is needed for static as well as for dynamic terms because the dormant part of a dynamic term may still have $.b(v)$'s which are later needed in the active part. For instance, $D \stackrel{\text{df}}{=} \langle a_1 \mid b^+(1) \mid \top\rangle.b(1) \fatsemi \overline{\langle a_2 \mid b^-(x) \mid \top\rangle}$ has a static component with a $.b(1)$ in it, and may be transformed into an equivalent $D' \stackrel{\text{df}}{=} \langle a_1 \mid b^+(1) \mid \top\rangle \fatsemi \overline{\langle a_2 \mid b^-(x) \mid \top\rangle.b(1)}$, see section 5.2.

5.1 Denotational semantics

The denotational semantics of terms is given by a mapping $\mathsf{box} : \mathcal{T} \to \mathcal{B}_s \cup \mathcal{B}_d$, defined homomorphically by induction on their structure, following the syntax of terms. Below, $\alpha\beta\gamma \in \mathcal{C}$, $a \in \mathbb{A}$, $b \in \mathbb{B}$, $v \in \mathbb{D}$, una stands for any unary operator ($\mathsf{sc}\,a$, $\mathsf{tie}\,b$ or $.b(v)$), and bin for any binary operator ($\|$, $\Box$, $\fatsemi$ or $\circledast$).

$$\mathsf{box}(\alpha\beta\gamma) \stackrel{\text{df}}{=} N_{\alpha\beta\gamma} \qquad \mathsf{box}(\overline{E}) \stackrel{\text{df}}{=} \overline{\mathsf{box}(E)} \qquad \mathsf{box}(\underline{E}) \stackrel{\text{df}}{=} \underline{\mathsf{box}(E)}$$
$$\mathsf{box}(F\,\mathsf{una}) \stackrel{\text{df}}{=} \mathsf{box}(F)\,\mathsf{una} \qquad \mathsf{box}(F_1\,\mathsf{bin}\,F_2) \stackrel{\text{df}}{=} \mathsf{box}(F_1)\,\mathsf{bin}\,\mathsf{box}(F_2)\ .$$

5.2 Structural similarity relation

We define the *structural similarity* relation on terms, denoted by $\equiv$, as the least equivalence relation on terms such that all the equations in table 1 are satisfied. Most of these rules are exactly those of MBC.

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

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

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

similarity relation is closed in the domain of terms, in the sense that, if a term matches one of the sides of any rule then, the other side defines a legal term.

In developing the operational semantics of the box algebra, we first introduce operational rules based on transitions of the nets providing the denotational semantics of terms. Then we will introduce the label based rules, together with the derived consistency results.

5.3 Transition and label 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 $\lambda(t)$.

We consider first the transition based operational semantics. It has moves of the form $F \xrightarrow{U} F'$ such that F and F' are terms and $U \in \mathbb{U} \stackrel{\text{df}}{=} \mathsf{mult}(\mathbb{T} \times \mathfrak{B})$, where $\mathfrak{B}$ is the set of bindings. The idea here is that U is a valid step for the boxes associated with F and F', *i.e.*, that $\mathsf{box}(F)[U\rangle\,\mathsf{box}(F')$.

Formally, we define a ternary relation $\longrightarrow$ which is the least relation comprising all $(F, U, F') \in \mathcal{T} \times \mathbb{U} \times \mathcal{T}$ such that the relations in table 2 hold. Notice that we use $F \xrightarrow{U} F'$ to denote $(F, U, F') \in \longrightarrow$. In the definition of EOP we make no restriction on U_1 and U_2 but the domain of application of 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.

For the label based operational semantics we retain first the structural similarity relation $\equiv$ on terms without any change. Next, we define moves of the form $F \xrightarrow{\Gamma} F'$, where F and F' are

$$\text{EA}\quad \overline{\alpha\beta\gamma.B} \xrightarrow{\{(v^{\alpha\beta\gamma},\sigma)\}} \underline{\alpha\beta\gamma.B'} \quad \text{if} \begin{cases} \sigma \text{ is an enabling binding for } \alpha\beta\gamma, \\ B = \sum_{b^-(e)\in\mathbb{L}} \beta(b^-(e)).b(\mathsf{eval}(\sigma(e))), \\ B' = \sum_{b^+(e)\in\mathbb{L}} \beta(b^+(e)).b(\mathsf{eval}(\sigma(e))). \end{cases}$$

$$\text{EQ1}\quad F \xrightarrow{\{\}} F \qquad \text{EQ2}\quad \frac{F \equiv F', F' \xrightarrow{U} F'', F'' \equiv F'''}{F \xrightarrow{U} F'''}$$

$$\text{EBUF}\quad \frac{F \xrightarrow{U} F'}{F.b(v) \xrightarrow{U} F'.b(v)} \qquad \text{ETIE}\quad \frac{F \xrightarrow{\{(t_1,\sigma_1),\cdots,(t_k,\sigma_k)\}} F'}{F\,\mathsf{tie}\,b \xrightarrow{\{(v^{\mathsf{tie}\,b}\lhd t_1,\sigma_1),\cdots,(v^{\mathsf{tie}\,b}\lhd t_k,\sigma_k)\}} F'\,\mathsf{tie}\,b}$$

$$\text{EOP}\quad \frac{F_1 \xrightarrow{\{(t_1,\sigma_1),\cdots,(t_k,\sigma_k)\}} F_1',\ F_2 \xrightarrow{\{(t'_1,\sigma'_1),\cdots,(t'_l,\sigma'_l)\}} F_2'}{F_1\,\mathsf{bin}\,F_2 \xrightarrow{\{(v^{\mathsf{bin}}\lhd t_1,\sigma_1),\cdots,(v^{\mathsf{bin}}\lhd t_k,\sigma_k)\}\cup\{(v^{\mathsf{bin}}\lhd t'_1,\sigma'_1),\cdots,(v^{\mathsf{bin}}\lhd t'_l,\sigma'_l)\}} F_1'\,\mathsf{bin}\,F_2'}$$

$$\text{ESC}\quad \frac{F \xrightarrow{U_1\uplus\ldots\uplus U_k} F'}{F\,\mathsf{sc}\,a \xrightarrow{\{(v^{\mathsf{sc}\,a}\lhd U_1^T,\sigma_1),\ldots,(v^{\mathsf{sc}\,a}\lhd U_k^T,\sigma_k)\}} F'\,\mathsf{sc}\,a} \text{ if } C_i \text{ for } 1 \le i \le k$$

Table 2. Transition based operational semantics, where $a \in \mathbb{S}$, $b \in \mathbb{B}$, $v \in \mathsf{type}(b)$, $C_i \stackrel{\text{df}}{=} \left(\sum_{(t,\sigma)\in U_i} U_i(t,\sigma)\cdot\sigma(\alpha(t)) \in \mathsf{dom}(\varphi_{\mathsf{sc}\,a})\right) \wedge \left(U_i^T = \sum_{(t,\sigma)\in U_i} U_i(t,\sigma)\cdot t\right) \wedge \left(\forall(t,\sigma)\in U_i, \sigma = \sigma_i\right)$ and bin stands for any binary operator.

terms as before, and $\Gamma \in \mathsf{mult}(\mathsf{mult}(\mathbb{A}) \times \mathfrak{B})$, these new rules are directly obtained from the rules of table 2 and are given in the table 3.

6 Unfolding and main result

In this section, we define a translation from the high-level domain of BCD to the low-level domain of MBC. Such a translation is called an *unfolding* and will be denoted by unf.

We introduce first the low-level sets of actions, buffers and links, used in the definition of terms and boxes. The set of low-level actions is $\mathbb{A}_\ell \stackrel{\text{df}}{=} \{a(\widetilde{d}) \mid a \in \mathbb{S}, \widetilde{d} \in \mathbb{D}^{\mathsf{ar}(a)}\}$, that of low-level buffers is $\mathbb{B}_\ell \stackrel{\text{df}}{=} \{b_d \mid d \in \mathbb{B}, d \in \mathsf{type}(b)\}$, that of low-level closed buffers is $\mathbb{B}_\ell{}^\diamond \stackrel{\text{df}}{=} \{b_d{}^\diamond \in \mathbb{B}^\diamond, d \in \mathsf{type}(b)\}$ and that of low-level links is $\mathbb{L}_\ell \stackrel{\text{df}}{=} \{b_d^+, b_d^- \mid b \in \mathbb{B}, d \in \mathsf{type}(b)\}$. A low-level interface function φ is a mapping $\mathsf{mult}(\mathsf{mult}(\mathbb{A}_\ell)) \to \mathsf{mult}(\mathbb{A}_\ell)$, identified with its graph included in $\mathsf{mult}(\mathsf{mult}(\mathbb{A}_\ell)) \times \mathsf{mult}(\mathbb{A}_\ell)$.

Low-level boxes sensibly differ from high-level ones. Formally, they are tuples of the form (S, T, λ, W, M), where the places in S are not typed and may only contain tokens $\bullet$, the transitions are labelled by multisets of low-level actions, the arcs are defined through a weight function W representing the number of tokens flowing on the arc rather than values, the marking function returns the number of tokens held by places instead of a multiset of values and the guards do not exist anymore.

The unfolding of a BCD box $N = (S, T, \lambda, M)$ is $\mathsf{unf}(N) \stackrel{\text{df}}{=} (S', T', \lambda', W, M')$, defined has follows:

- $S' \stackrel{\text{df}}{=} \{s_d \in S \mid s \in S, d \in \mathsf{type}(s)\}$,
- $T' \stackrel{\text{df}}{=} \{t_\sigma \mid t \in T, \sigma \text{ is an enabling binding of } t\}$,
- for all $s_d \in S'$, $\lambda'(s_d) \stackrel{\text{df}}{=} \begin{cases} b_d & \text{if } \lambda(s) = b \in \mathbb{B}, \\ b_d^\diamond & \text{if } \lambda(s) = b^\diamond \in \mathbb{B}^\diamond, \\ \lambda(s) & \text{otherwise,} \end{cases}$

$$\text{LA}\quad \overline{\alpha\beta\gamma.B} \xrightarrow{\{(\alpha,\sigma)\}} \underline{\alpha\beta\gamma.B'} \quad \text{if} \begin{cases} \sigma \text{ is an enabling binding for } \alpha\beta\gamma, \\ B = \sum_{b^-(e)\in\mathbb{L}} \beta(b^-(e)).b(\mathsf{eval}(\sigma(e))), \\ B' = \sum_{b^+(e)\in\mathbb{L}} \beta(b^+(e)).b(\mathsf{eval}(\sigma(e))). \end{cases}$$

$$\text{LQ1}\quad F \xrightarrow{\{\}} F \qquad \text{LQ2}\quad \frac{F \equiv F', F' \xrightarrow{\Gamma} F'', F'' \equiv F'''}{F \xrightarrow{\Gamma} F'''}$$

$$\text{LBUF}\quad \frac{F \xrightarrow{\Gamma} F'}{F.b(v) \xrightarrow{\Gamma} F'.b(v)}$$

$$\text{LTIE}\quad \frac{F \xrightarrow{\Gamma} F'}{F\,\mathsf{tie}\,b \xrightarrow{\Gamma} F'\,\mathsf{tie}\,b} \qquad \text{LOP}\quad \frac{F_1 \xrightarrow{\Gamma_1} F_1',\ F_2 \xrightarrow{\Gamma_2} F_2'}{F_1\,\mathsf{bin}\,F_2 \xrightarrow{\Gamma_1+\Gamma_2} F_1'\,\mathsf{bin}\,F_2'}$$

$$\text{LSC}\quad \frac{F \xrightarrow{\Gamma_1+\ldots+\Gamma_k} F'}{F\,\mathsf{sc}\,a \xrightarrow{(\varphi_{\mathsf{sc}\,a}(\Gamma_1^\alpha),\sigma_1)+\ldots+(\varphi_{\mathsf{sc}\,a}(\Gamma_k^\alpha),\sigma_k)} F'\,\mathsf{sc}\,a} \quad \text{if } C_i \text{ for } 1 \le i \le k$$

Table 3. Label based operational semantics for BCD, where $a \in \mathbb{A}$, $b \in \mathbb{B}$, $v \in type(b)$, $C_i \stackrel{\text{df}}{=} \left(\Gamma_i^\alpha = \sum_{(\alpha,\sigma)\in\Gamma_i} \Gamma_i(\alpha,\sigma)\cdot\sigma(\alpha)\right) \wedge \left(\Gamma_i^\alpha \in \mathsf{dom}(\varphi_{\mathsf{sc}\,a})\right) \wedge \left(\forall(\alpha,\sigma)\in\Gamma_i, \sigma = \sigma_i\right)$ and bin stands for any binary BCD operator.

- for all $t_\sigma \in T'$, $\lambda'(t_\sigma) \stackrel{\text{df}}{=} \sigma(\alpha(t))$,
- for all $s_d \in S'$ and $t_\sigma \in T'$, $W(s_d, t_\sigma) \stackrel{\text{df}}{=} \mathsf{eval}(\sigma(\lambda(s,t)))(d)$ and $W(t_\sigma, s_d) \stackrel{\text{df}}{=} \mathsf{eval}(\sigma(\lambda(t,s)))(d)$,
- for all $s_d \in S'$, $M'(s_d) \stackrel{\text{df}}{=} M(s)(d)$, *i.e.*, the place s_d contains as many tokens as the value d appears in the place s.

An example of the unfolding of a box is provided in the figure 5.

Moreover, we define $\mathsf{unf}(a) \stackrel{\text{df}}{=} \{a(\widetilde{d}) \mid \widetilde{d} \in \mathbb{D}^{\mathsf{ar}(a)}\} \subseteq \mathbb{A}_\ell$ and $\mathsf{unf}(b) \stackrel{\text{df}}{=} \{b_d \mid d \in \mathsf{type}(b)\} \subseteq \mathbb{B}_\ell$.

The steps of a high-level box can be unfolded as follows: $\mathsf{unf}(U) \stackrel{\text{df}}{=} \sum_{(t,\sigma)\in U} U(t,\sigma)\cdot t_\sigma$ for any step U. The unfolding of labelled steps is similar: $\mathsf{unf}(\Gamma) \stackrel{\text{df}}{=} \sum_{(\alpha,\sigma)\in\Gamma} \Gamma(\alpha,\sigma)\cdot\sigma(\alpha)$ for any labelled step Γ.

The low-level version of terms has the following syntax, which is actually the syntax of MBC terms:

$$\begin{aligned} \mathcal{T}_{s\ell} \quad & E ::= \alpha_\ell\beta_\ell \mid E\,\mathsf{sc}\,a_\ell \mid E\,\mathsf{tie}\,b_\ell \mid \ E.b_\ell \ \mid E\|E \mid E \,\square\, E \mid \ E \fatsemi E \ \mid E \circledast E \\ \mathcal{T}_{d\ell} \quad & D ::= \ \overline{E} \ \mid \ \underline{E} \ \mid D\,\mathsf{sc}\,a_\ell \mid D\,\mathsf{tie}\,b_\ell \mid D.b_\ell \mid \ D\|D \ \mid D \,\square\, E \mid E \,\square\, D \\ & \qquad D \fatsemi E \mid \ E \fatsemi D \ \mid D \circledast E \mid \ E \circledast D \end{aligned}$$

where $\alpha_\ell \in \mathsf{mult}(\mathbb{A}_\ell)$, $\beta_\ell \in \mathsf{mult}(\mathbb{L}_\ell)$, $a_\ell \in \mathbb{A}_\ell$ and $b_\ell \in \mathbb{B}_\ell$.

The unfolding of BCD terms is defined by induction on their syntax; the unfolding of a basic term is the choice between all the low-level terms it encodes:

$$\mathsf{unf}(\alpha\beta\gamma) \stackrel{\text{df}}{=} \bigsqcap_{\sigma\in\mathfrak{B}_{\alpha\beta\gamma}} \alpha_\sigma\beta_\sigma$$

where $\mathfrak{B}_{\alpha\beta\gamma}$ is the set of all the bindings enabling $\alpha\beta\gamma$, $\alpha_\sigma \stackrel{\text{df}}{=} \sum_{A\in\alpha} \alpha(A)\cdot\sigma(A)$ and $\beta_\sigma \stackrel{\text{df}}{=} \sum_{b^*(e)\in\beta, *\in\{+,-\}} \beta(b^*(e))\cdot b^*_{\mathsf{eval}(\sigma(e))}$. Then, for the terms F, F_1 and F_2, $a \in \mathbb{S}$, $b \in \mathbb{B}$, $d \in \mathsf{type}(b)$

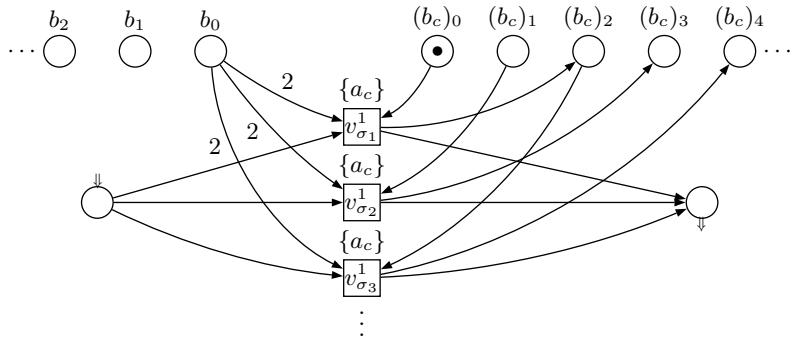

Fig. 5. A fragment of $\mathsf{unf}(N_{\{a_c\}\{b^-(x),b^-(x),b_c^-(k),b_c^+(k+2)\}\{\top\}})$, where $\sigma_1 = \{x \mapsto 0, k \mapsto 0\}$, $\sigma_2 = \{x \mapsto 0, k \mapsto 1\}$, and $\sigma_3 = \{x \mapsto 0, k \mapsto 2\}$. Notice that transition $v^1_{\sigma_2}$ is dead because place $(b_c)_1$ can never be marked.

and any binary operator bin, we define:

$$\mathsf{unf}(F\,\mathsf{sc}\,a) \stackrel{\text{df}}{=} \mathsf{unf}(F)\,\mathsf{sc}\,\mathsf{unf}(a) \qquad \mathsf{unf}(F\,\mathsf{tie}\,b) \stackrel{\text{df}}{=} \mathsf{unf}(F)\,\mathsf{tie}\,\mathsf{unf}(b)$$
$$\mathsf{unf}(F.b(d)) \stackrel{\text{df}}{=} \mathsf{unf}(F).b_d \qquad \mathsf{unf}(F_1\,\mathsf{bin}\,F_2) \stackrel{\text{df}}{=} \mathsf{unf}(F_1)\,\mathsf{bin}\,\mathsf{unf}(F_2)$$
$$\mathsf{unf}(\overline{F}) \stackrel{\text{df}}{=} \overline{\mathsf{unf}(F)} \qquad \mathsf{unf}(\underline{F}) \stackrel{\text{df}}{=} \underline{\mathsf{unf}(F)}$$

Notice that since $\mathsf{unf}(a)$ is generally a set of low-level actions, $\mathsf{sc}\,\mathsf{unf}(a)$ is in fact a shorthand for successive applications of scoping with respect to all elements of $\mathsf{unf}(a)$ (which may be applied in any order because of the commutativity of scoping). A similar remark concerns $\mathsf{unf}(b)$.

6.1 Relationship with the MBC model

The low-level objects defined above are almost those defined by MBC with the following syntactical differences:

- the set $\mathbb{A}_\ell$ above is denoted $\mathbb{A}$ in MBC, moreover, the conjugated action of $a(1)$ is denoted $\widehat{a}(1)$ instead of $\widehat{a(1)}$ in MBC;
- the symbols ${b_d}^\diamond$, for $d \in \mathbb{D}$, used for low-level closed buffers are all denoted $\flat$ in MBC.

The BCD and MBC frameworks are very similar: both define a domain of terms with an operational semantics and a denotational semantics. As a consequence, most semantical objects defined in BCD are high-level versions of those defined in MBC. In particular, the function box for the denotational semantics and the relations $\longrightarrow$ and $\equiv$ for the operational semantics already exist in MBC. Moreover, BCD defines the same operators as MBC except that they operate on high-level boxes, actions and buffers. The same concerns the step sequence semantics which is defined the same way in BCD and MBC.

The definition of the net substitution is exactly the same in BCD and MBC in the case of sequential and parallel operators. In the case of buffer restriction $\Omega_{\mathsf{tie}\,b}$, the only difference is that in BCD, a label is changed from b to $b^\diamond$ while in MBC, it is changed to $\flat$ (which makes no difference for the behaviour). In the case of communication interface operators, BCD uses high-level interface functions while, MBC uses low-level ones.

Similarly, the definitions of the operational semantics in BCD and MBC are very closely related, with however some small differences: $b(v)$ in the BCD rules OPL and OPR becomes b_v in MBC (and analogously for the rules in table 2 and 3); the BCD rule REN has no equivalent in MBC because

there are no variables; the BCD rule EA allows to execute a basic term under an enabling binding, which corresponds in MBC to an execution (through the rules EA and EOP which is necessary in order to select a term in the choice of terms produced by the unfolding) of the low-level term corresponding to this binding. The rules ESC are also similar even if the BCD version turns out to be really more complicated than the MBC one. Nevertheless, the principle remains the same: in BCD the execution of each transition based step U_i allows the execution of a transition $v^{\mathsf{sc}\,a} \triangleleft U_i$, which corresponds exactly to what the rule ESC in MBC allows to infer. Thus, several applications of this rule in MBC, one for each U_i, can simulate an application of the rule ESC in BCD; exactly as one scoping w.r.t. $a \in \mathbb{A}$ corresponds to many scoping w.r.t. the symbols in $\mathsf{unf}(a)$.

As a consequence, in what follows, we will use exactly the same notations for the high- and low-level versions of operations, semantical functions, etc. For instance, we shall denote $F_1 \| F_2$ the parallel composition regardless of whether F_1 and F_2 are high- or low-level terms.

6.2 Consistency results

The main result of the paper is the consistency of the model w.r.t. the unfolding which states that for any term, the semantics of its unfolding coincides with the unfolding of its semantics. Since a term has a denotational and an operational semantics, our consistency result is twofold: the first theorem below concerns the consistency of the box semantics while the second one concerns the consistency of the operational semantics.

As shown in [5], the box and the SOS semantics of a MBC term are equivalent in arguably the strongest sense (they generate isomorphic transition systems). As a direct consequence of this and our consistency results, we can infer that the box and SOS semantics of a MBC term are equivalent as well. More results derived from these two theorems are given in the section 6.3.

Lemma 1. *Let N, N_1 and N_2 be high-level boxes,* bin *be ant binary operator, $a \in \mathbb{S}$ and $b \in \mathbb{B}$. We have:*

1. $\mathsf{unf}(N_1 \,\mathsf{bin}\, N_2) = \mathsf{unf}(N_1)\,\mathsf{bin}\,\mathsf{unf}(N_2)$.
2. $\mathsf{unf}(N\,\mathsf{tie}\,b) = \mathsf{unf}(N)\,\mathsf{tie}\,\mathsf{unf}(b)$.
3. $\mathsf{unf}(N\,\mathsf{sc}\,a) = \mathsf{unf}(N)\,\mathsf{sc}\,\mathsf{unf}(a)$.

These equalities are considered up to renaming of places and transitions.

Proof. (1) Let $N_{left} \stackrel{\text{df}}{=} \mathsf{unf}(N_1 \,\mathsf{bin}\, N_2)$ and $N_{right} \stackrel{\text{df}}{=} \mathsf{unf}(N_1)\,\mathsf{bin}\,\mathsf{unf}(N_1)$. For each transition $(t_{v_i} \triangleleft t)_\sigma$ in N_{left}, there is a corresponding transition $t_{v_i} \triangleleft t_\sigma$ in N_{right} and conversely. Similarly, the control places are of the form $(x_1, \ldots, x_k, e_1, \ldots, e_k)_\bullet$ in N_{left}, corresponding to the places $(x_{1\bullet}, \ldots, x_{k\bullet}, e_{1\bullet}, \ldots, e_{k\bullet})$ in N_{right}. Concerning the buffer places, each $(p^b_{v_1}, \ldots, p^b_{v_n})_d$ in N_{right} corresponds to a place $((p^b_{v_1})_d, \ldots, (p^b_{v_n})_d)$ in N_{left} (with additional parentheses for a better reading). One can check that the arcs are created consistently in both cases, as well as the marking.

(2) This case is very similar to the previous one. In particular, the buffer places related to b are called $(b^\diamond)_v$ when the buffer restriction is applied first, while they are called $(b_v)^\diamond$ when the unfolding is applied first.

(3) The result follows from the observation that a high-level interface function encodes exactly the low-level ones which can be obtained through any binding. More precisely, for any binding σ and any $a \in \mathbb{S}$, the high-level interface function $\varphi_{\mathsf{sc}\,a}$ is such that:

$$\bigcup_{(\Gamma,\alpha,\delta)\in\varphi_{\mathsf{sc}\,a}} \sigma \circ \delta(\Gamma, \alpha) = \bigcup_{a_\ell \in \mathsf{unf}(a)} \varphi_{\mathsf{sc}\,a_\ell}$$

where each $\varphi_{\mathsf{sc}\,a_\ell}$ is a low-level interface function used to perform one scoping involved in $\mathsf{sc}\,\mathsf{unf}(a)$. On the one hand, the binding σ is necessary in order to eliminate the variables in (Γ, α), but $\sigma \circ \delta(\Gamma, \alpha)$ already belongs to one $\varphi_{\mathsf{sc}\,a_\ell}$. On the other hand, for each (Γ, α) in any $\varphi_{\mathsf{sc}\,a_\ell}$, we also already have $(\Gamma, \alpha, \delta_{\mathrm{id}})$ in $\varphi_{\mathsf{sc}\,a}$. □

Theorem 1. *Let N and N' be two boxes, U a step and Γ a labelled step. Then:*

1. $N[U\rangle N' \iff \mathsf{unf}(N)[\mathsf{unf}(U)\rangle\, \mathsf{unf}(N')$.
2. $N[\Gamma\rangle N' \iff \mathsf{unf}(N)[\mathsf{unf}(\Gamma)\rangle\, \mathsf{unf}(N')$.

Proof. (1) By definition of the unfolding because each high-level transition t which is enabled with a binding σ appears in the unfolding as t_σ which is also enabled; and conversely.

(2) Follows from point (1), by definition of labelled steps. □

Theorem 2. *Let F and F' be two terms, U a step and Γ a labelled step. Then:*

1. $\mathsf{unf}(\mathsf{box}(F)) = \mathsf{box}(\mathsf{unf}(F))$.
2. $F \equiv F' \iff \mathsf{unf}(F) \equiv \mathsf{unf}(F')$.
3. $F \xrightarrow{U} F' \iff \mathsf{unf}(F) \xrightarrow{\mathsf{unf}(U)} \mathsf{unf}(F')$.
4. $F \xrightarrow{\Gamma} F' \iff \mathsf{unf}(F) \xrightarrow{\mathsf{unf}(\Gamma)} \mathsf{unf}(F')$.

Proof. (1) By induction on the syntax of terms, using the lemma 1.

(2) Assume that $F \equiv F'$. For each rule in the table 1 used to derive this equivalence, there exists an equivalent rule in MBC. The only exception is the rule REN which allows one to rename the variables in a term, but renaming variables has no impact on the unfolding which just uses other equivalent enabling bindings.

(3) The results comes from the direct correspondence between the BCD and MBC operational rules. In particular, a bounded transitions (t, σ) in U corresponds to a t_σ in $\mathsf{unf}(U)$. The case of the high-level basic action is different because it unfolds into a choice of low-level basic actions, so the rule EA in the high-level corresponds to the rules EA and EOP in the low-level (the involved operator being the choice).

(4) Follows from point (3), by definition of labelled steps. □

6.3 Other results

From the consistency results, we may derive several interesting properties of the BCD model, directly inherited from MBC.

Corollary 1. *Let N be a box and $N[U\rangle N'$.*

1. *If N is static, then $U = \{\}$ and $N = N'$.*
2. *If Σ is dynamic, then U is a set of bounded transitions.*

Corollary 2. *Let N be a box and B, B' in $\mathsf{mult}(\{b(v) \mid b \in \mathbb{B}, v \in \mathsf{type}(b)\})$.*

1. *N is static iff $N.B$ is static, and N is dynamic iff $N.B$ is dynamic.*
2. *$\overline{N}$ is dynamic iff N is static iff $\underline{N}$ is dynamic.*
3. *$N.\varnothing = N$, $N.B.B' = N.(B + B')$, $\overline{N}.B = \overline{N.B}$ and $\underline{N}.B = \underline{N.B}$.*
4. *If N is static or dynamic, then $\lfloor N \rfloor$ and $[\![N]\!]$ are static.*
5. *If N is static, then $\lfloor N \rfloor = N$.*
6. *$[\![[\![N]\!]]\!] = [\![\lfloor N \rfloor]\!] = \lfloor [\![N]\!] \rfloor = [\![N]\!]$.*
7. *$\lfloor N \rfloor.B = \lfloor N.B \rfloor$, $\lfloor \overline{N} \rfloor = \lfloor \underline{N} \rfloor = \lfloor \lfloor N \rfloor \rfloor = \lfloor N \rfloor$ and $[\![N.B]\!] = [\![\overline{N}]\!] = [\![\underline{N}]\!] = [\![N]\!]$.*

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

Corollary 3. *Let Ω be any sequential operator box or the parallel composition operator box $\Omega_\parallel$, $\mathbf{N}$ be any Ω-tuple of boxes, and N be any box.*

1. $M^{\mathsf{c}}_{\Omega(\mathbf{N})} = \varnothing$ *iff* $M^{\mathsf{c}}_{N_{v_i}} = \varnothing$, *for each* $v_i \in T_\Omega$.
2. *If* N' *is* $\Omega_\varphi(N)$ *or* $N\,\mathsf{tie}\,b$ *or* $N.B$, *then* $M^{\mathsf{c}}_{N'} = \varnothing$ *iff* $M^{\mathsf{c}}_N = \varnothing$.

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

A marking M of a box N *quasi-safe* if, for every transition t, there is a control place $s \in {}^\bullet t$ such that $M(s) \leq \{\bullet\}$; note that this implies ac-freeness.

Corollary 4. *Let* Ω *be any operator box and* $\mathbf{N} \in \mathsf{dom}_\Omega$. *Then* $\Omega(\mathbf{N})$ *is a box with a clean and ac-free marking. Moreover, if all the dynamic boxes (if any) in* $\mathbf{N}$ *have quasi-safe markings, then the marking of* $\Omega(\mathbf{N})$ *is also quasi-safe.*

Corollary 5. *Let* Ω *be a sequential operator box, and* $\mathbf{N}$ *be an* Ω*-tuple of static boxes.*

1. *If* $v \in T_\Omega$ *is such that the entry place of* Ω *is* ${}^\bullet v$ *or* $v^\bullet$, *then* $\overline{\Omega(\mathbf{N})} = \Omega(\mathbf{N}')$, *where* $\mathbf{N}'$ *is* $\mathbf{N}$ *with* N_v *replaced respectively by* $\overline{N_v}$ *or* $\underline{N_v}$.
2. *If* $v \in T_\Omega$ *is such that the exit place of* Ω *is* ${}^\bullet v$ *or* $v^\bullet$, *then* $\underline{\Omega(\mathbf{N})} = \Omega(\mathbf{N}')$, *where* $\mathbf{N}'$ *is* $\mathbf{N}$ *with* N_v *replaced respectively by* $\overline{N_v}$ *or* $\underline{N_v}$.

The way static and dynamic boxes are composed in BCD guarantees that the result is a static or dynamic box when the domain of application of the operators is respected.

Corollary 6. *Let* Ω *be an operator box of ABC and* $\mathbf{N} \in \mathsf{dom}_\Omega$. *Then every net derivable from* $\Omega(\mathbf{N})$ *is of the form* $\Omega(\mathbf{N}')$, *where* $\mathbf{N}' \in \mathsf{dom}_\Omega$ *and* $[\![\mathbf{N}']\!] = [\![\mathbf{N}]\!]$. *Moreover, if no box in* $\mathbf{N}$ *is dynamic, then every net derivable from* $\overline{\Omega(\mathbf{N})}$ *or* $\underline{\Omega(\mathbf{N})}$ *is of the form* $\Omega(\mathbf{N}')$, *where* $\mathbf{N}' \in \mathsf{dom}_\Omega$ *and* $\lfloor \mathbf{N}' \rfloor = \lfloor \mathbf{N} \rfloor$.

Corollary 7. *Every composite net of BCD is a quasi-safe static or dynamic box. Moreover, it is static iff the marking operators* $\overline{(\cdot)}$ *or* $\underline{(\cdot)}$ *are not used, unless in the scope of the* $\lfloor\cdot\rfloor$ *or* $[\![\cdot]\!]$ *operators.*

Corollary 8. *Let* F *be a box.*

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

Corollary 9. *Let* F_1 *and* F_2 *be two terms.*

1. *If* $F_1 \equiv F_2$ *then* $\lfloor F_1 \rfloor \equiv \lfloor F_2 \rfloor$, $[\![F_1]\!] = [\![F_2]\!]$ *and* $\mathsf{box}(F_1) = \mathsf{box}(F_2)$.
2. *If* $[\![F_1]\!] = [\![F_2]\!]$, *then* $\mathsf{box}(F_1) = \mathsf{box}(F_2)$ *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 $F_1 \stackrel{\text{df}}{=} \langle a \mid \varnothing \mid \top \rangle\, \mathsf{sc}\, a$ and $F_2 \stackrel{\text{df}}{=} \langle \widehat{a} \mid \varnothing \mid \top \rangle\, \mathsf{sc}\, a$ for which $F_1 \not\equiv F_2$ but $\mathsf{box}(F_1) = \mathsf{box}(F_2)$ (no transition is left in the nets by the scoping operation).

Corollary 10. *Let* F *be a term.*

1. $\mathsf{box}(F) = \overline{\lfloor \mathsf{box}(F) \rfloor}$ *iff* $F \equiv \overline{\lfloor F \rfloor}$.
2. $\mathsf{box}(F) = \underline{\lfloor \mathsf{box}(F) \rfloor}$ *iff* $F \equiv \underline{\lfloor F \rfloor}$.

Corollary 11. *Let* F *and* F' *be two terms. Then,* $F \xrightarrow{\{\}} F'$ *iff* $F \equiv F'$.

7 Execution scenarios

We give here the execution of the scenario presented in introduction, using the rules of the transition based and label based operational semantics.

$\overline{\textsc{Sys}}$

$\equiv$ E1, IPAR1, IIT1

$\Big(\overline{(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b_p(0)}\circledast\langle a(n) \mid b_p^-(n)|\top\rangle)\,\mathsf{tie}\,b_p \;\|$
$\overline{(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle.b_c(0)}\circledast\langle\widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle)\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$

$\xrightarrow{\{(t_1,\{n\mapsto 0,x\mapsto 2\})\}}$ EA, ETIE, EOP, ESC

$\Big(\underline{(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b_p(1).b(2)}\circledast\langle a(n) \mid b_p^-(n)|\top\rangle)\,\mathsf{tie}\,b_p \;\|$
$\overline{(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle.b_c(0)}\circledast\langle\widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle)\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$

$\xrightarrow{\{(t_1,\{n\mapsto 0,x\mapsto 5\})\}}$ IIT2, EA, ETIE, EOP, ESC

$\Big(\underline{(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b_p(2).b(2).b(5)}\circledast\langle a(n) \mid b_p^-(n)|\top\rangle)\,\mathsf{tie}\,b_p \;\|$
$\overline{(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle.b_c(0)}\circledast\langle\widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle)\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$

$\xrightarrow{\{(t_1,\{n\mapsto 0,x\mapsto 2\})\}}$ IIT2, EA, ETIE, EOP, ESC

$\Big(\underline{(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b_p(3).b(2).b(5).b(2)}\circledast\langle a(n) \mid b_p^-(n)|\top\rangle)\,\mathsf{tie}\,b_p \;\|$
$\overline{(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle.b_c(0)}\circledast\langle\widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle)\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$

$\xrightarrow{\{(t_3,\{k\mapsto 0,x\mapsto 2\}),(t_1,\{n\mapsto 3,x\mapsto 4\})\}}$ IIT2, EA, ETIE, EOP, ESC

$\Big(\underline{(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b_p(4).b(5).b(4)}\circledast\langle a(n) \mid b_p^-(n)|\top\rangle)\,\mathsf{tie}\,b_p \;\|$
$\underline{(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle.b_c(2)}\circledast\langle\widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle)\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$

$\equiv$ IIT3, B1

$\Big(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b(5).b(4)\circledast\overline{\langle a(n) \mid b_p^-(n)|\top\rangle).b_p(4)}\,\mathsf{tie}\,b_p \;\|$
$(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle\circledast\overline{\langle\widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle.b_c(2)})\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$

$\xrightarrow{\{(t_5,\{n\mapsto 4,k\mapsto 2\})\}}$ EA, EOP, ETIE, ESC

$\Big(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b(5).b(4)\circledast\underline{\langle a(n) \mid b_p^-(n)|\top\rangle)}\,\mathsf{tie}\,b_p \;\|$
$(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle\circledast\underline{\langle\widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle})\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$

$\equiv$ $\underline{\textsc{Sys}}$ IIT5, IPAR2, X1

Fig. 6. Execution scenario using transition based SOS rules.

$\overline{\text{SYS}}$

$\equiv$ E1, IPAR1, IIT1

$$\Big(\overline{(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b_p(0)} \circledast \langle a(n) \mid b_p^-(n) | \top\rangle)\,\mathsf{tie}\,b_p \;\|$$
$$\overline{(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle.b_c(0)} \circledast \langle \widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle)\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$$

$\xrightarrow{\{(\{a_p\},\{n\mapsto 0,x\mapsto 2\})\}}$ LA, LOP, LTIE, LSC

$$\Big(\underline{(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b_p(1).b(2)} \circledast \langle a(n) \mid b_p^-(n) | \top\rangle)\,\mathsf{tie}\,b_p \;\|$$
$$\overline{(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle.b_c(0)} \circledast \langle \widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle)\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$$

$\xrightarrow{\{(\{a_p\},\{n\mapsto 1,x\mapsto 5\})\}}$ IIT2, LA, LOP, LTIE, LSC

$$\Big(\underline{(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b_p(2).b(2).b(5)} \circledast \langle a(n) \mid b_p^-(n) | \top\rangle)\,\mathsf{tie}\,b_p \;\|$$
$$\overline{(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle.b_c(0)} \circledast \langle \widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle)\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$$

$\xrightarrow{\{(\{a_p\},\{n\mapsto 2,x\mapsto 2\})\}}$ IIT2, LA, LOP, LTIE, LSC

$$\Big(\underline{(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b_p(3).b(2).b(5).b(2)} \circledast \langle a(n) \mid b_p^-(n) | \top\rangle)\,\mathsf{tie}\,b_p \;\|$$
$$\overline{(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle.b_c(0)} \circledast \langle \widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle)\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$$

$\xrightarrow{\{(\{a_c\},\{k\mapsto 0,x\mapsto 2\}),(\{a_p\},\{n\mapsto 3,x\mapsto 4\})\}}$ IIT2, LA, LOP, LTIE, LSC

$$\Big(\underline{(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b_p(4).b(5).b(4)} \circledast \langle a(n) \mid b_p^-(n) | \top\rangle)\,\mathsf{tie}\,b_p \;\|$$
$$\underline{(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle.b_c(2)} \circledast \langle \widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle)\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$$

$\equiv$ IIT3, B1

$$\Big(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b(5).b(4) \circledast \overline{\langle a(n) \mid b_p^-(n) | \top\rangle).b_p(4)}\,\mathsf{tie}\,b_p \;\|$$
$$(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle \circledast \overline{\langle \widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle.b_c(2)})\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$$

$\xrightarrow{\{(\{\},\{n\mapsto 4,k\mapsto 2\})\}}$ LA, LOP, LTIE, LSC

$$\Big(\langle a_p \mid b_p^-(n), b_p^+(n{+}1), b^+(x) \mid \top\rangle.b(5).b(4) \circledast \underline{\langle a(n) \mid b_p^-(n) | \top\rangle)}\,\mathsf{tie}\,b_p \;\|$$
$$(\langle a_c \mid b_c^-(k), b_c^+(k{+}2), b^-(x,x) \mid \top\rangle \circledast \underline{\langle \widehat{a}(m) \mid b_c^-(k) \mid m{-}k \le 4\rangle})\,\mathsf{tie}\,b_c\Big)\,\mathsf{sc}\,a$$

$\equiv$ $\underline{\text{SYS}}$ IIT5, IPAR2, X1

Fig. 7. Execution scenario using label based SOS rules.

8 Conclusion

We presented here how the compositional framework of MBC may be efficiently extended in order to cope with systems manipulating large data types (even infinite as show in the example). The main change consisted in using buffers capable to carry coloured tokens. This feature was exploited by both, asynchronous communications (which were possible through high-level links) and synchronous ones (which were able to transmit high-level values through the parameters of actions). This extension provides a real progress form a practical point of view since it allows to represent in a very compact way systems with potentially infinite data types.

Actually, independent papers [8,7] showed that the obtained class of composable Petri nets is powerful enough to give an elegant compositional semantics to parallel specification languages, like $B(PN)^2$, or formalisms, like π-calculus.

From the theoretical point of view, the obtained framework, called BCD, was shown to be a coherent high-level counter-part of MBC in the sense that a BCD term and its unfolding (which is an MBC term) have the same behaviour and similarly for a BCD box and its unfolding. Moreover, for a BCD term, the semantics in terms of transition systems generated, on the one hand, by the associated operational rules and, on the other hand, by the transition rule of the associated Petri net, are isomorphic.

References

1. E.Best, R.Devillers and J.Hall. *The Petri Box Calculus: a New Causal Algebra with Multilabel Communication.* Advances in Petri Nets 1992, LNCS 609, 1992.
2. E.Best, R.Devillers and M.Koutny. *A Unified Model for Nets and Process Algebras.* Handbook of Process Algebra, Elsevier, 2001.
3. E.Best, R.Devillers and M.Koutny. *Petri Net Algebra.* EATCS Monographs on TCS, Springer, 2001.
4. E.Best and R.P.Hopkins. *B(PN)^2 – a Basic Petri Net Programming Notation.* PARLE'93, LNCS 694, 1993.
5. C.Bui Thanh, H.Klaudel and F.Pommereau. *Asynchronous Box Calculus with Multi-way Communication.* LACL Tech. Report, Univ. Paris 12, 2002. Available on ⟨`http://www.univ-paris12.fr/lacl`⟩.
6. R.Devillers, H.Klaudel, M.Koutny and F.Pommereau. *An Algebra of Non-safe Petri Boxes.* AMAST'02, LNCS 2422, Springer, 2002.
7. R.Devillers, H.Klaudel and M.Koutny. *Compositional High-Level Petri Net Semantics of π-calculus.* Manuscript.
8. H.Klaudel *Parameterized M-expression semantics of parallel procedures.* DAPSYS'00, Kluwer Academic Publishers, 2000.
9. H.Klaudel and F.Pommereau. *Asynchronous links in the PBC and M-nets.* ASIAN'99, LNCS 1742, Springer, 1999.
10. R.Milner, R.Parrow and J.Walker. *A calculus of mobile processes, Parts I and II.* Information and Computation, 100(1), 1992.
11. R.Milner *Communication and Concurrency.* Prentice Hall, 1989.
12. G.D.Plotkin. *A Structural Approach to Operational Semantics.* Tech. Report FN-19, Computer Science Department, Univ. of Aarhus, 1981.