An Algebra of Non-safe Petri Boxes

Raymond Devillers, Hanna Klaudel, Maciej Koutny, Franck Pommereau

▶ **To cite this version:**

Raymond Devillers, Hanna Klaudel, Maciej Koutny, Franck Pommereau. An Algebra of Non-safe Petri Boxes. AMAST, 2002, Saint-Gilles-les-Bains, France. pp.192-207, 10.1007/3-540-45719-4_14 . hal-02309988

HAL Id: hal-02309988
https://hal.science/hal-02309988

Submitted on 9 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.

An Algebra of Non-safe Petri Boxes

R. Devillers[1], H. Klaudel[2], M. Koutny[3], and F. Pommereau[2]

[1] Département d'Informatique, Université Libre de Bruxelles,
B-1050 Bruxelles, Belgium. rdevil@ulb.ac.be
[2] Université Paris 12, LACL, 61 avenue du général de Gaulle,
94010 Créteil, France. {klaudel,pommereau}@univ-paris12.fr
[3] Department of Computing Science, University of Newcastle upon Tyne,
NE1 7RU, United Kingdom. Maciej.Koutny@newcastle.ac.uk

Abstract. We define an algebraic framework based on non-safe Petri nets, which allows one to express operations such as iteration, parallel composition, and transition synchronisation. This leads to an algebra of process expressions, whose constants and operators directly correspond to those used in Petri nets, and so we are able to associate nets to process expressions compositionally. The semantics of composite nets is then used to guide the definition of a structured operational semantics of process expressions. The main result is that an expression and the corresponding net generate isomorphic transition systems. We finally discuss a partial order semantics of the two algebras developed in this paper.
Keywords. Petri nets, process algebra, operational semantics.

1 Introduction

To relate process algebras, such as CCS [17] or CSP [13], and Petri nets [20], the approaches proposed in the literature often aim at providing a Petri net semantics to an existing process algebra as, *e.g.*, in [5, 6, 10–12, 18]. Another way is to translate elements from Petri nets into process algebras as, *e.g.*, in [1].

A specific framework we are concerned with here is the *Petri Net Algebra* (PNA [4]) and its precursor, *Petri Box Calculus* (PBC [3]). This framework is composed of an algebra of process expressions (called *box expressions*) together with a fully compositional translation into labelled safe Petri nets (called *boxes*). (Recall that in a safe Petri net no place ever holds more than one token.)

The variant of the safe box algebra relevant to this paper comprises: *sequence* $E; F$ (the execution of E is followed by that of F); *choice* $E \,\square\, F$ (either E or F can be executed); *parallel composition* $E \| F$ (E and F can be executed concurrently); *iteration* $E \circledast F$ (E can be executed an arbitrary number of times, and then be followed by F); and *scoping* $E\,\mathsf{sc}\,a$ (all handshake synchronisations involving pairs of a- and $\widehat{a}$-labelled transitions are enforced). Consider, as an example, three processes modelling a critical section and two user processes: $\textsc{CritSect} \stackrel{\text{df}}{=} ((a_1; r_1) \,\square\, (a_2; r_2)) \circledast f$, $\textsc{User}_1 \stackrel{\text{df}}{=} \widehat{a_1}; \widehat{r_1}$ and $\textsc{User}_2 \stackrel{\text{df}}{=} \widehat{a_2}; \widehat{r_2}$. The atomic actions a_1 and a_2 (together with the matching $\widehat{a_1}$ and $\widehat{a_2}$) model getting the access to a shared resource, r_1 and r_2 (together with $\widehat{r_1}$ and $\widehat{r_2}$) model its release, and f models a final action. The box expression where these three

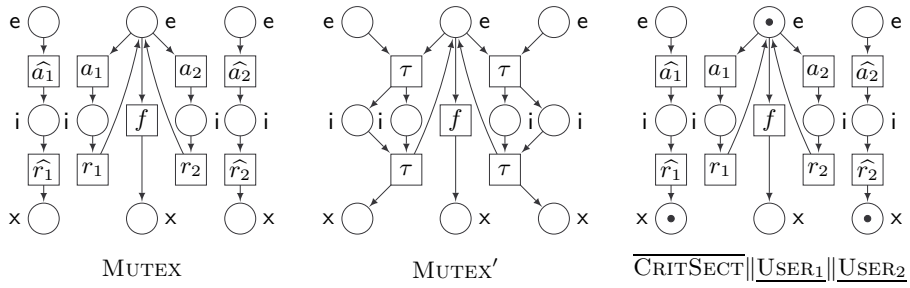

Fig. 1. Boxes and corresponding box expressions.

processes operate in parallel is $\text{MUTEX} \stackrel{\text{df}}{=} \text{CRITSECT}\|\text{USER}_1\|\text{USER}_2$ (operators like parallel composition associate to the right), and the corresponding box is shown on the left of figure 1. In a box, places are labelled by their status (e for entry, x for exit and i for internal) while transitions are labelled by CCS-like communication actions; such as a_1, $\widehat{a_1}$ and τ (as in CCS, τ is an internal action).

Though the box of MUTEX specifies the three constituent processes, it does not allow for interprocess communication. This can be achieved by applying the scoping w.r.t. the synchronisation actions a_i and r_i, which results in $\text{MUTEX}' \stackrel{\text{df}}{=} \text{MUTEX}\,\mathsf{sc}\,a_1\,\mathsf{sc}\,a_2\,\mathsf{sc}\,r_1\,\mathsf{sc}\,r_2$, with the corresponding box shown in figure 1.

The operational semantics of box expressions is given through SOS rules in Plotkin's style [19]. However, instead of rules like $a.E \stackrel{a}{\longrightarrow} E$ in CCS, the current state of an evolution is represented using overbars and underbars, marking respectively the initial and final states of (sub)expressions. E.g., in figure 1, the box on the right represents MUTEX after the two user processes have terminated, and the critical section is still in its initial state. There are two kinds of SOS rules: structural rules specify when distinct expressions denote the same state, *e.g.*, one can deduce that $\overline{\text{CRITSECT}}\|\text{USER}_1\|\text{USER}_2 \equiv \overline{\text{CRITSECT}} \parallel \overline{\text{USER}_1}\|\text{USER}_2 \equiv \overline{\text{CRITSECT}} \parallel \overline{\text{USER}_1} \parallel \overline{\text{USER}_2}$, while evolution rules specify when we may have a state change due to the execution of some of the actions, *e.g.*, one can deduce that $((a_1;r_1)\,\square\,(a_2;r_2)) \circledast \overline{f} \stackrel{\{f\}}{\longrightarrow} ((a_1;r_1)\,\square\,(a_2;r_2)) \circledast \underline{f}$. The two algebras of PNA are fully compatible, in the sense that a box expression and the corresponding box generate isomorphic transition systems. It will be our goal here to retain this property in a more expressive framework based on *non-safe* boxes.

Recently, [7, 15] introduced a novel feature into the above model, aimed at the modelling of asynchronous communication (used, *e.g.*, to model time-dependent or preemptive concurrent systems [15, 16]). Consider the following process expressions modelling a producer and consumer processes (each can perform exactly *one* action, after which it terminates): $\text{PRODONE} \stackrel{\text{df}}{=} pb^+$ and $\text{CONSONE} \stackrel{\text{df}}{=} cb^-$. The pb^+ is an atomic action whose role is to 'produce' a token (resource) in a buffer identified by b; in doing so, it generates the visible label p. The cb^- is an atomic action which 'consumes' a resource from buffer b, generating the visible label c. The boxes of these processes are shown in figure 2, where the buffer places are identified by the b labels.

An intuitive meaning of a b-labelled place is that some transitions can insert

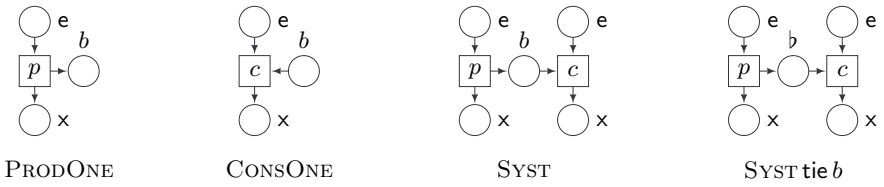

Fig. 2. Asynchronous communication.

tokens into it, while other transitions can later remove them. This gives rise to asynchronous communication as, *e.g.*, in the parallel composition of the producer and consumer processes, $\textsc{Syst} \stackrel{\text{df}}{=} \textsc{ProdOne} \| \textsc{ConsOne}$, and the corresponding box in figure 2, where the two b-labelled buffer places are merged into a single b-labelled place (this place can later be merged with other b-labelled buffer places). An abstraction mechanism for asynchronous communication comes in the form of the *buffer restriction* operator, $\mathsf{tie}\, b$, which changes the b-labelled buffer place into a $\flat$-labelled one. Such a place can no longer be merged with other buffer places, and so $\flat$-labelled places may be viewed as internal places. This is illustrated in figure 2 for $\textsc{Syst}\, \mathsf{tie}\, b$. The resulting model is no longer based on safe Petri nets since the buffer places are in general non-safe.

In this paper, we develop further the above approach, introducing the *Asynchronous Box Calculus* (or ABC) model. In particular, we no longer enforce the safeness of the non-buffer places, which may be undesirable for practical applications since this leads to the imposition of awkward syntactic constraints (see [4]). Safeness was needed in the original PNA to support a simple concurrency semantics of boxes and expressions based on causal partial orders. In this paper, it is replaced by *auto-concurrency freeness* which is a property guaranteeing concurrency semantics in terms of event structures as shown in [14], and so is highly relevant from a theoretical point of view. Moreover, we argue that in the case of ABC without buffer places (but still with non-safe places), one can retain the simple causal partial order semantics.

In short, ABC will comprise an algebra of non-safe boxes, and an algebra of box expressions. The two algebras will be related through a mapping which, for a box expression, returns a corresponding box with an isomorphic transition system. All the proofs can be found in the technical report [8].

Two examples. Consider three process expressions: $\textsc{Prod} \stackrel{\text{df}}{=} \textsc{ProdOne} \circledast f$, $\textsc{ProdPar} \stackrel{\text{df}}{=} (\textsc{ProdOne} \| \textsc{ProdOne}) \circledast f$ and $\textsc{Cons} \stackrel{\text{df}}{=} \textsc{ConsOne} \circledast f$. When operating in parallel, Prod can repeatedly send a token to a b-labelled buffer place, which can then be repeatedly removed by the Cons process.

The first example, $\textsc{Syst}_{\text{I}} \stackrel{\text{df}}{=} s; (\textsc{Prod} \| \textsc{Prod} \| \textsc{Cons})$ on the left of figure 3, models a system composed of two producers and one consumer operating in parallel and preceded by a 'startup' action s. The example $\textsc{Syst}_{\text{II}} \stackrel{\text{df}}{=} (\textsc{ProdPar} \| \textsc{ConsOne})\, \mathsf{tie}\, b$, shown on the right of figure 3, illustrates the encapsulating feature of buffer restriction which makes the buffer place $\flat$-labelled, and so no longer available for future merging. This example also shows that even if we

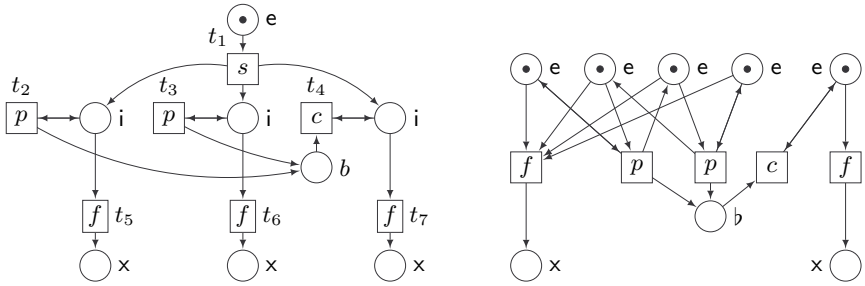

Fig. 3. Boxes for the two execution scenarios (double-headed arrows are self-loops).

disregard the buffer place, the resulting box is still non-safe as executing either of the p-labelled transitions adds a second token to one of the e-labelled places. Hence SYST_II cannot be handled by the preliminary version of ABC presented in [7].

The operational semantics is illustrated using the following two execution scenarios (in each case the system starts from its implicit initial state, *e.g.*, we consider $\overline{\text{SYST}_\text{I}}$ rather than SYST_I). In the first scenario, we consider the net Σ_I shown in figure 3 and the following evolution:

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

This corresponds to $\Sigma_I\,[\{t_1\}\{t_2,t_3\}\{t_2,t_4\}\{t_5,t_6,t_7\}\rangle\,\Sigma'_I$, which is a step sequence such that Σ'_I is Σ_I with two tokens in the buffer place, one token in each of the x-labelled places, and no token elsewhere. Since each x-labelled place has exactly one token, we consider Σ'_I to be in a *final* marking (or state). In terms of labelled step sequences, we have $\Sigma_I\,[\{s\}\{p,p\}\{p,c\}\{f,f,f\}\rangle\,\Sigma'_I$.

In the second scenario, we only consider labelled steps, for the net Σ_{II} in figure 3, and the following evolution:

- the system begins by executing the left p-labelled transition, which puts a token in the buffer place, and another one in the third e-labelled place;
- the consumer takes the token from the buffer place and the right p-labelled transition puts another token in the buffer place;
- the system finishes by executing the two f-labelled transitions.

Such a scenario corresponds to $\Sigma_{II}\,[\{p\}\{p,c\}\{f,f\}\rangle\,\Sigma'_{II}$, which is a labelled step sequence such that Σ'_{II} is Σ_{II} with one token in the buffer place and each of the x-labelled places, and no token elsewhere. Notice that Σ'_{II} is also in a final state.

Basic notations and definitions. Throughout the paper, we use the standard Petri net notions (see, *e.g.*, [20]) and (multi)set notation. In particular, $+$ and $\cdot$ denote multiset addition and multiplication by a natural number, $\mathsf{mult}(X)$ comprises all finite multisets over a set X, and $\leq$ is used to compare multisets. A subset of a set X may be treated as a multiset over X, through its characteristic function, and a singleton set can be identified with its sole element. The specific Petri net framework is sketched below.

We assume that there is a set $\mathbb{A}_\tau$ of (atomic) *actions* representing synchronous interface activities used, in particular, to model handshake communication. Similarly as in CCS, $\mathbb{A}_\tau \stackrel{\text{df}}{=} \mathbb{A} \uplus \{\tau\}$ and, for every $a \in \mathbb{A}$, $\widehat{a}$ is an action in $\mathbb{A}$ such that $\widehat{\widehat{a}} = a$. There is also a finite set $\mathbb{B}$ of *buffer symbols* for asynchronous communication.

A *(marked) labelled net* is here a tuple $\Sigma \stackrel{\text{df}}{=} (S, T, W, \lambda, M)$ such that: S and T are finite disjoint sets of respectively *places* and *transitions*; W is a *weight function* from the set $(S \times T) \cup (T \times S)$ to $\mathbb{N}$; λ is a *labelling* for places and transitions such that for every place $s \in S$, $\lambda(s)$ is a symbol in $\{\mathsf{e}, \mathsf{i}, \mathsf{x}, \flat\} \uplus \mathbb{B}$, and for every transition $t \in T$, $\lambda(t)$ is an action in $\mathbb{A}_\tau$; and M is a *marking*, *i.e.*, a multiset over S.

If the labelling of a place s is e, i or x, then s is an *entry*, *internal* or *exit* place, respectively. If the labelling is $\flat$ then s is a *closed buffer* place, and if it is $b \in \mathbb{B}$, then s is an *open buffer* place. Collectively, the e-, i- and x–labelled places are called *control (flow)* places. Moreover, the set of all entry (resp. exit) places will be denoted by ${}^\circ\Sigma$ (resp. Σ°). To avoid ambiguity, we will sometimes decorate the various components of Σ with the index Σ and, to simplify diagrams, omit isolated unmarked buffer places.

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

For a marking M of a labelled net Σ, we use M^{ctr} to denote M restricted to the control places. Then we say that M is *clean* if ${}^\circ\Sigma \leq M^{ctr} \Rightarrow M^{ctr} = {}^\circ\Sigma$ and $\Sigma^\circ \leq M^{ctr} \Rightarrow M^{ctr} = \Sigma^\circ$. Moreover, M is *ac-free* if, for every $t \in T$, there is a control place $s \in {}^\bullet t$ such that $M(s) < 2 \cdot W_\Sigma(s, t)$, *i.e.*, the marking of s does not allow *auto-concurrency* of t.

2 An algebra of boxes

To model concurrent systems, we use a class of labelled nets called *asynchronous boxes*. To model operations on such nets, we use *operator boxes* (a particular kind of labelled nets where all transitions are intended to be substituted by asynchronous boxes) and the *net substitution* meta-operator (called also refinement [4]), which allows one to realise this substitution.

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

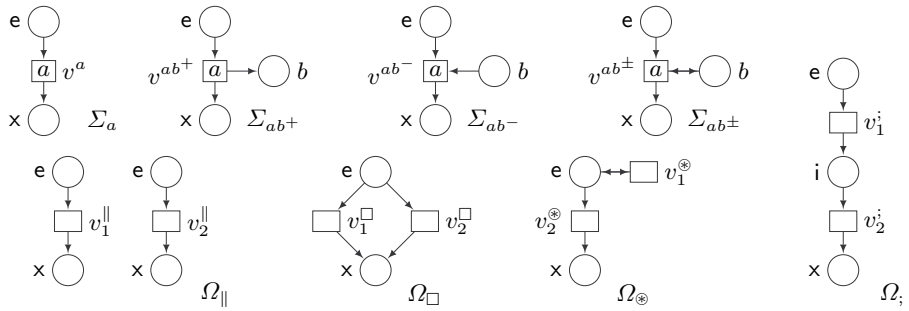

Fig. 4. Asynchronous and operator boxes of ABC, where $a \in \mathbb{A}_\tau$ and $b \in \mathbb{B}$.

– *control-restricted*: for every transition t there is at least one control place in ${}^\bullet t$, and at least one control place in $t^\bullet$.

The execution semantics of Σ is based on finite steps, which capture the potential concurrency in its behaviour. A step is a finite multiset of transitions U and, when enabled, it can be executed leading to a new net Θ; we denote this by $\Sigma[U\rangle\Theta$ or $\Theta \in [\Sigma\rangle$. Transition labelling may be extended to steps, leading to labelled steps of the form $\Sigma\,[\Gamma\rangle_\lambda\,\Sigma'$ which means that there is a step U such that $\Sigma\,[U\rangle\,\Sigma'$ and $\Gamma = \lambda(U)$. 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.

A box Σ is *static* (resp. *dynamic*) if $M_\Sigma^{ctr} = \varnothing$ (resp. $M_\Sigma^{ctr} \neq \varnothing$) and all the markings reachable from M_Σ^{ctr}, ${}^\circ\Sigma$ or Σ° in the box Σ^{ctr} are both clean and ac-free, where Σ^{ctr} is Σ with all its buffer places and adjacent arcs removed. The asynchronous boxes, static boxes and dynamic boxes will respectively be denoted by abox, abox^{stc} and abox^{dyn}. Static boxes do not admit non-empty steps, and if Σ is a dynamic box, then U is a *set* of transitions (but if $\Sigma\,[\Gamma\rangle_\lambda\,\Sigma'$, then the labelled step Γ may be a true *multiset* of actions, as illustrated in the execution scenarios in the introduction) and Σ' is a dynamic box.

The upper row in figure 4 shows four kinds of static boxes Σ_α used in ABC, where $\alpha \in \mathcal{A} \stackrel{\text{df}}{=} \{a\,, ab^+\,, ab^-\,, ab^\pm \mid a \in \mathbb{A}_\tau \wedge b \in \mathbb{B}\}$. They are the basic building blocks, from which other static and dynamic boxes are constructed.

The complete behaviour of a static or dynamic box can be represented by a transition system. And, since we have two kinds of possible steps, we introduce two kinds of transition systems. The *full transition system* of a dynamic box Σ is $\mathsf{fts}_\Sigma \stackrel{\text{df}}{=} (V, L, A, \mathit{init})$ where $V \stackrel{\text{df}}{=} [\Sigma\rangle$ are the states; $L \stackrel{\text{df}}{=} 2^{T_\Sigma}$ are arc labels; $A \stackrel{\text{df}}{=} \left\{(\Sigma', U, \Sigma'') \in V \times L \times V \;\middle|\; \Sigma'\,[U\rangle\,\Sigma''\right\}$ is the set of arcs; and $\mathit{init} \stackrel{\text{df}}{=} \Sigma$ is the initial state. For a static box Σ, $\mathsf{fts}_\Sigma \stackrel{\text{df}}{=} \mathsf{fts}_{\overline{\Sigma}}$. The *labelled transition system* of a static or dynamic box Σ, denoted by lts_Σ, is defined as fts_Σ with each arc label U changed to $\lambda_\Sigma(U)$ (note that different arcs between two states in fts_Σ may give rise to a single arc in lts_Σ).

The presentation of the operators of ABC starts with *marking operators*, which modify the marking of a box Σ (below $b \in \mathbb{B}$ and $B \in \mathsf{mult}(\mathbb{B})$):

- $\Sigma.B$ adds $B(b)$ tokens to the b-labelled open buffer place of Σ, for each $b \in \mathbb{B}$; moreover, $\Sigma.b \stackrel{\text{df}}{=} \Sigma.\{b\}$ (this operation is called *buffer stuffing*);
- $\overline{\Sigma}$ (resp. $\underline{\Sigma}$) is Σ with one additional token in each entry (resp. exit) place, *i.e.*, $M_{\overline{\Sigma}} \stackrel{\text{df}}{=} M_\Sigma + {}^\circ\Sigma$ (resp. $M_{\underline{\Sigma}} \stackrel{\text{df}}{=} M_\Sigma + \Sigma^\circ$);
- $[\![\Sigma]\!]$ is Σ with the empty marking.

For the remaining operators, the identities of transitions will play a key role, especially when defining the SOS semantics of process expressions. More precisely, 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 η of *basic* transition identities and a corresponding set of basic labelled trees with a single node labelled with an element of η. All the transitions in figure 4 are assumed to be of that kind. To express more complex (unordered) finite trees, or sets of trees, used as transition identities in nets obtained through net substitution, we will use the following linear notations (see figure 5 page 9 for an illustration):

- $v \lhd \mathsf{T}$, where $v \in \eta$ 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 v-labelled root;
- $v \lhd \mathsf{t}$ denotes $v \lhd \{\mathsf{t}\}$ and $v \blacktriangleleft \mathsf{T}$ denotes the set of trees $\{v \lhd \mathsf{t} \mid \mathsf{t} \in \mathsf{T}\}$.

A similar naming discipline could be used for the places of the constructed nets following the scheme used in [4]. However, since place trees were essentially needed for the definition of recursion which is not considered in the present paper, we will not use them. Instead, when applying net substitution, we will assume that the place sets of the operands are renamed to make them disjoint. We then construct new places by aggregating the existing ones; *e.g.*, if s_1 and s_2 are places from some boxes, (s_1, s_2) may be the identity of a newly constructed place. We start with the two unary operations (we define them directly, rather than through net substitution).

Scoping $\Sigma\,\mathsf{sc}\,a$. Parameterised by a communication action $a \in \mathbb{A}$ and with the domain of application $dom_{\mathsf{sc}\,a} \stackrel{\text{df}}{=} \mathsf{abox}^{stc} \cup \mathsf{abox}^{dyn}$, $\Sigma\,\mathsf{sc}\,a$ is a labelled net which is like Σ with the only difference that the set of transitions comprises all trees $w \stackrel{\text{df}}{=} v^{\mathsf{sc}\,a} \lhd \{t, u\}$ with $t, u \in T$ such that $\{\lambda(t), \lambda(u)\} = \{a, \widehat{a}\}$, as well as all trees $z \stackrel{\text{df}}{=} v^{\mathsf{sc}\,a} \lhd r$ with $r \in T$ such that $\lambda(r) \notin \{a, \widehat{a}\}$. The label of w is τ, and that of z is $\lambda(r)$; the weight function is given by $W_{\Sigma\,\mathsf{sc}\,a}(p, w) \stackrel{\text{df}}{=} W(p, t) + W(p, u)$ and $W_{\Sigma\,\mathsf{sc}\,a}(p, z) \stackrel{\text{df}}{=} W(p, r)$, and similarly for $W_{\Sigma\,\mathsf{sc}\,a}(w, p)$ and $W_{\Sigma\,\mathsf{sc}\,a}(z, p)$.

Buffer restriction $\Sigma\,\mathsf{tie}\,b$. Parameterised by a buffer $b \in \mathbb{B}$ and with the domain of application $dom_{\mathsf{tie}\,b} \stackrel{\text{df}}{=} \mathsf{abox}^{stc} \cup \mathsf{abox}^{dyn}$, $\Sigma\,\mathsf{tie}\,b$ is like Σ with the only difference that the identity of each transition t is changed to $v^{\mathsf{tie}\,b} \lhd t$, the label of the only b-labelled place is changed to $\natural$, and a new unmarked unconnected b-labelled place is added.

Choice $\Sigma_1 \,\square\, \Sigma_2$*, iteration* $\Sigma_1 \circledast \Sigma_2$*, sequence* $\Sigma_1 \,;\, \Sigma_2$*, and parallel composition* $\Sigma_1 \| \Sigma_2$. We consider here a binary operator box $\Omega_{\mathsf{bin}} \in \{\Omega_\square, \Omega_\circledast, \Omega_;, \Omega_\|\}$, as shown in figure 4 (the labels of the transitions will be irrelevant here), and its application $\Sigma_1 \,\mathsf{bin}\, \Sigma_2$. The first three binary operator boxes specify different ways to sequentially compose behaviours and have the domain of application $dom_{\mathsf{bin}} \stackrel{\text{df}}{=} (\mathsf{abox}^{stc})^2 \cup (\mathsf{abox}^{dyn} \times \mathsf{abox}^{stc}) \cup (\mathsf{abox}^{stc} \times \mathsf{abox}^{dyn})$. The last one has the domain of application $dom_\| \stackrel{\text{df}}{=} (\mathsf{abox}^{stc})^2 \cup (\mathsf{abox}^{dyn})^2$, *i.e.*, its components may evolve concurrently. The net $\Phi = \Sigma_1 \,\mathsf{bin}\, \Sigma_2$ is defined as follows.
The transitions of Φ are the set of all trees $v_i^{\mathsf{bin}} \lhd t$ (with $t \in T_{\Sigma_i}$ and $i = 1, 2$). The label of each $v_i^{\mathsf{bin}} \lhd t$ is that of t. Each i-labelled or b-labelled place $p \in S_{\Sigma_j}$ belongs to S_Φ. Its label and marking are unchanged and, for every transition $v_i^{\mathsf{bin}} \lhd t$, the weight function is given by $W_\Phi(p, v_i^{\mathsf{bin}} \lhd t) \stackrel{\text{df}}{=} W_{\Sigma_i}(p, t)$ if $j = i$ and $W_\Phi(p, v_i^{\mathsf{bin}} \lhd t) \stackrel{\text{df}}{=} 0$ otherwise, and similarly for $W_\Phi(v_i^{\mathsf{bin}} \lhd t, p)$.
For every place $s \in S_{\Omega_{\mathsf{bin}}}$ with ${}^\bullet s = \{v_{l_1}^{\mathsf{bin}}, \ldots, v_{l_k}^{\mathsf{bin}}\}$ and $s^\bullet = \{v_{j_1}^{\mathsf{bin}}, \ldots, v_{j_m}^{\mathsf{bin}}\}$ $(k, m \in \{0, 1, 2\})$, we construct in S_Φ all the places of the form $p \stackrel{\text{df}}{=} (x_1, \ldots, x_k, e_1, \ldots, e_m)$, where each x_r (if any) is an exit place of Σ_{l_r}, and each e_q (if any) is an entry place of Σ_{j_q}. 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 $v_i^{\mathsf{bin}} \lhd t$, the weight function is given by:

$$W_\Phi(p, v_i^{\mathsf{bin}} \lhd t) \stackrel{\text{df}}{=} \begin{cases} W_{\Sigma_i}(x_r, t) + W_{\Sigma_i}(e_q, t) & \text{if } v_i^{\mathsf{bin}} \in {}^\bullet s \cap s^\bullet \text{ and } i = l_r = j_q \\ W_{\Sigma_i}(x_r, t) & \text{if } v_i^{\mathsf{bin}} \in {}^\bullet s \setminus s^\bullet \text{ and } i = l_r \\ W_{\Sigma_i}(e_q, t) & \text{if } v_i^{\mathsf{bin}} \in s^\bullet \setminus {}^\bullet s \text{ and } i = j_q \\ 0 & \text{otherwise,} \end{cases}$$

and similarly for $W_\Phi(v_i^{\mathsf{bin}} \lhd t, p)$.
For every $b \in \mathbb{B}$, there is a unique b-labelled place $p^b \stackrel{\text{df}}{=} (p_1^b, p_2^b) \in S_\Phi$ which merges the two b-labelled places, p_1^b and p_2^b, of the two operands. The marking of p^b is the sum of the markings of p_1^b and p_2^b, and for each transition $v_i^{\mathsf{bin}} \lhd t$, the weight function is given by $W_\Phi(p^b, v_i^{\mathsf{bin}} \lhd t) \stackrel{\text{df}}{=} W_{\Sigma_i}(p_i^b, t)$, and similarly for $W_\Phi(w \lhd t, p^b)$.

Properties. The net operations of ABC (other than $\overline{\Sigma}$ and $\underline{\Sigma}$) always return a syntactically valid object, *i.e.*, a box with a clean and ac-free marking. If one makes no use of buffer stuffing nor buffer restriction nor basic nets other than Σ_a, one gets net operations similar to those defined in the standard box algebra (see [3, 4]), except for the additional b-labelled places which are all isolated and unmarked.

The net operations are illustrated in figure 5, where explicit transition identities are shown for various stages of the construction, from the basic net and transition identities shown in figure 4.

Relating structure and behaviour. We now investigate how the behaviour of composite boxes depends on the behaviours of the boxes being composed. We provide full details for sequential composition (other operators are treated in [8]).

First, we capture situations where different application of sequential composition lead to the same result. For two pairs of boxes, $\mathbf{\Sigma} \stackrel{\text{df}}{=} (\Sigma_1, \Sigma_2)$ and

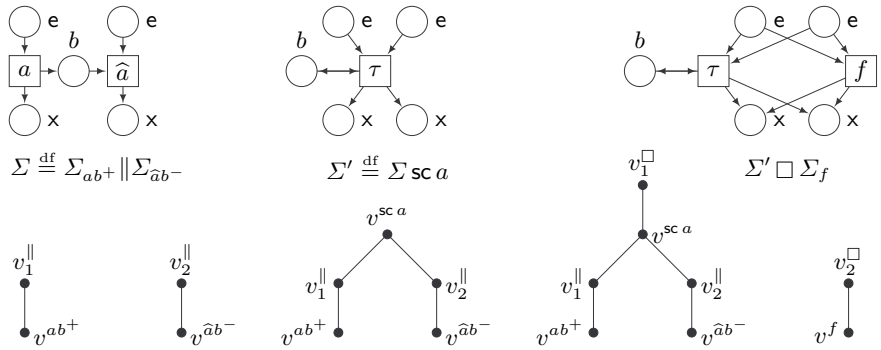

Fig. 5. The trees give the identities of the transitions (from left to right) of the boxes in the upper row; *e.g.*, the fourth tree is $v_1^{\square} \lhd v^{\mathsf{sc}\,a} \lhd \{v_1^{\|} \lhd v^{ab^+}, v_2^{\|} \lhd v^{\widehat{a}b^-}\}$.

$\Theta \stackrel{\text{df}}{=} (\Theta_1, \Theta_2)$, we define:

- $\Sigma \equiv' \Theta$ if there are boxes Ψ_1 and Ψ_2 such that $\{\Sigma, \Theta\} = \{(\underline{\Psi_1}, \Psi_2), (\Psi_1, \overline{\Psi_2})\}$. In other words, if the first operand has reached a final state, then this is equivalent to the second operand being in an initial state;
- $\Sigma \equiv'' \Theta$ if there are boxes Ψ_1 and Ψ_2 and $B_1, B_2, B_1', B_2' \in \mathsf{mult}(\mathbb{B})$ such that $B_1 + B_2 = B_1' + B_2'$, $\Sigma = (\Psi_1.B_1, \Psi_2.B_2)$ and $\Theta = (\Psi_1.B_1', \Psi_2.B_2')$. In other words, Σ and Θ are the same except perhaps the distribution of tokens in open buffer places corresponding to the same b but coming from different components (buffer stuffing never changes the marking of closed buffer places). The sequence operator will glue the corresponding open buffer places thus merging their markings.

We then define $\equiv_{\Omega_{;}}$ to be $(\equiv' \cup id_{\mathsf{abox}}) \circ \equiv''$, where id_{abox} is the identity on abox.

Relations $\equiv_{\Omega_{\mathsf{bin}}}$ like that above are defined for all binary operators of ABC. It may be shown that, if $\Sigma \equiv_{\Omega_{\mathsf{bin}}} \Theta$, then $\Sigma \in dom_{\mathsf{bin}} \Rightarrow \Theta \in dom_{\mathsf{bin}}$ and $[\![\Sigma_i]\!] = [\![\Theta_i]\!]$ $(i = 1, 2)$; and if $[\![\Sigma_i]\!] = [\![\Theta_i]\!]$ $(i = 1, 2)$, then $\Sigma_1 \,\mathsf{bin}\, \Sigma_2 = \Theta_1 \,\mathsf{bin}\, \Theta_2$ iff $\Sigma \equiv_{\Omega_{\mathsf{bin}}} \Theta$. Hence, when restricted to dom_{bin}, $\equiv_{\Omega_{\mathsf{bin}}}$ is an equivalence relation which identifies the tuples of boxes which give rise to the same composite nets.

The next result captures the *behavioural compositionality* of our model, *i.e.*, the way the behaviours of composite nets (in terms of enabled steps) are related to the behaviours of the composed nets. Basically, we want to establish what steps are enabled by $\Sigma_1 \,\mathsf{bin}\, \Sigma_2$ knowing the steps enabled by Σ_1 and Σ_2.

Theorem 1. *Let $\Sigma_1 \,\mathsf{bin}\, \Sigma_2$ be a valid application of a binary ABC operator* bin.

- *If $\Sigma_i \,[U_i\rangle\, \Psi_i$ $(i = 1, 2)$, then $\Sigma_1 \,\mathsf{bin}\, \Sigma_2 \,[V\rangle\, \Psi_1 \,\mathsf{bin}\, \Psi_2$, where $V = (v_1^{\mathsf{bin}} \blacktriangleleft U_1) \uplus (v_2^{\mathsf{bin}} \blacktriangleleft U_2)$.*
- *If $\Sigma_1 \,\mathsf{bin}\, \Sigma_2 \,[V\rangle\, \Delta$ then there are Θ, Ψ, U_1 and U_2, such that: $\Theta \equiv_{\Omega_{\mathsf{bin}}} \Sigma$, $\Theta_i \,[U_i\rangle\, \Psi_i$ $(i = 1, 2)$, $\Psi_1 \,\mathsf{bin}\, \Psi_2 = \Delta$ and V is as above.*

Various important consequences may be derived from the result presented

above and a similar result which holds for the unary operators; *e.g.*, the way static and dynamic boxes are composed guarantees that the result is a static or dynamic box when the domain of application of the operators is respected, *i.e.*, every composite net of ABC is a static or dynamic box.

3 An algebra of asynchronous box expressions

We consider an algebra of process expressions over the signature:

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

where $\mathcal{A}$ is the set of constants. The binary operators $\|$, $;$, $\Box$ and $\circledast$ will be used in the infix mode; the unary operators $\mathsf{sc}\, a$, $\mathsf{tie}\, b$ and $.b$ 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 expressions corresponding respectively to the static and dynamic boxes: the *static* E and *dynamic* D expressions, denoted respectively by aexpr^{stc} and aexpr^{dyn}. Collectively, we will refer to them as the (asynchronous) *box expressions*, aexpr. Their syntax is given by:

$$\begin{array}{rcl} E & ::= & \alpha \mid E\, \mathsf{sc}\, a \mid E\, \mathsf{tie}\, b \mid E.b \mid E \| E \mid E\, \mathsf{bin}\, E \\ D & ::= & \overline{E} \mid \underline{E} \mid D\, \mathsf{sc}\, a \mid D\, \mathsf{tie}\, b \mid D.b \mid D \| D \mid D\, \mathsf{bin}\, E \mid E\, \mathsf{bin}\, D \end{array}$$

where $\alpha \in \mathcal{A}$, $a \in \mathbb{A}$, $b \in \mathbb{B}$, and bin is a binary operator other than $\|$. In the following, we will use E or F to denote any static expression, J or K any dynamic expression, and G or H any static or dynamic expression. For an expression G, $[\![G]\!]$ is G with all occurrences of $\overline{(\cdot)}$, $\underline{(\cdot)}$ and $.b$ removed.

Essentially, a box expression encodes the structure of a box, together with the current marking of the control places (with the bars) and the buffer places (with 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 final state. Note that the $.b$ notation is needed for static as well as for dynamic box expressions because a static part of a dynamic box expression may have $.b$'s which can be used by the active part.

The denotational semantics of box expressions, $\mathsf{box} : \mathsf{aexpr} \to \mathsf{abox}$, is given compositionally. Assuming that $\alpha \in \mathcal{A}$, $b \in \mathbb{B}$, una stands for an unary and bin for a binary operator of ABC, we have: $\mathsf{box}(\alpha) \stackrel{\text{df}}{=} \Sigma_\alpha$, $\mathsf{box}(\overline{E}) \stackrel{\text{df}}{=} \overline{\mathsf{box}(E)}$, $\mathsf{box}(\underline{E}) \stackrel{\text{df}}{=} \underline{\mathsf{box}(E)}$, $\mathsf{box}(G\, \mathsf{una}) \stackrel{\text{df}}{=} \mathsf{box}(G)\, \mathsf{una}$ and $\mathsf{box}(G\, \mathsf{bin}\, H) \stackrel{\text{df}}{=} \mathsf{box}(G)\, \mathsf{bin}\, \mathsf{box}(H)$. One can show that the semantical mapping always returns a static or dynamic box, and that $\mathsf{box}(G)$ is static iff G is static. We now set out to define an operational semantics of box expressions.

A *structural similarity* relation on box expressions, denoted by $\equiv$, is defined as the least equivalence relation on box expressions such that all the equations in the upper part of table 1 are satisfied. The rules either directly follow those of the original PNA or capture the fact that an asynchronous message, produced by a ab^+ expression and represented by $.b$, can freely move within a box expression in order to be received by some action of the form ab^-. However, the $.b$ may never cross the boundary imposed by the $\mathsf{tie}\, b$ operator (notice that moving outside a

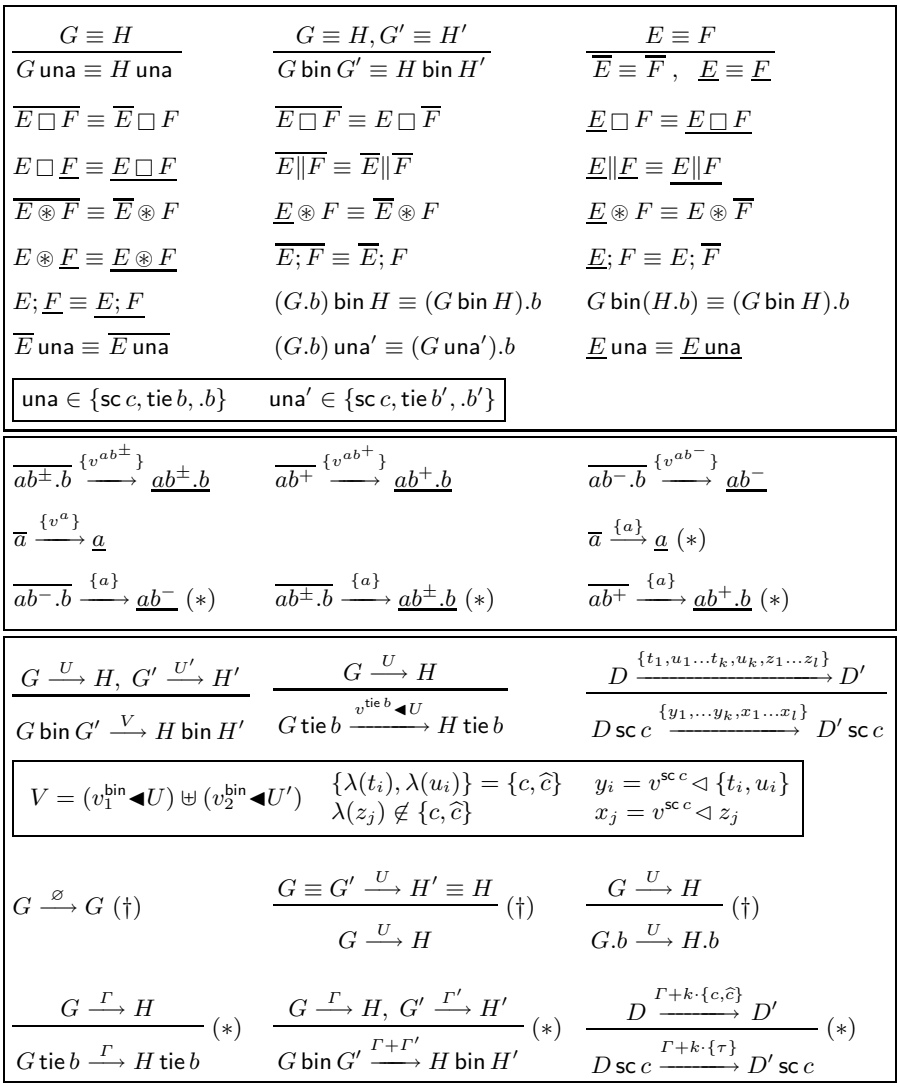

Table 1. Structural similarity, and two operational semantics for ABC, where $a \in \mathbb{A}_\tau$, $c \in \mathbb{A}$, $b \neq b' \in \mathbb{B}$ and $\mathsf{bin} \in \{\|, ;, \Box, \circledast\}$.

buffer restriction context is only allowed if $b \neq b'$). It may be observed that 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 captures the fact that box expressions have the same net translation, *i.e.*, $\mathsf{box}(G) = \mathsf{box}(H)$ iff $G \equiv H$, provided that $[\![G]\!] = [\![H]\!]$.

SOS semantics. In developing the operational semantics of ABC, we first introduce operational rules based on transitions of boxes which provide the denotational semantics of box expressions. Based on these, we formulate our key consistency result. Then we introduce the label based rules, together with the derived consistency results.

Consider the set $\mathbb{T}$ of all transition trees in the boxes derived through the box mapping. The first operational semantics has moves of the form $G \xrightarrow{U} H$ such that G and H are box expressions and $U \in \mathbb{U}$, where $\mathbb{U}$ is the set of all finite subsets of $\mathbb{T}$. The idea here is that U is a valid step for the boxes associated with G and H, *i.e.*, that $\mathsf{box}(G)\,[U\rangle\,\mathsf{box}(H)$ holds. Note that each $t \in \mathbb{T}$ has always the same label in the boxes derived through the box mapping; such a label will be denoted by $\lambda(t)$.

Formally, we define a ternary relation $\longrightarrow$ which is the least relation comprising all $(G, U, H) \in \mathsf{aexpr} \times \mathbb{U} \times \mathsf{aexpr}$ such that the relations in table 1 (middle and bottom parts) other than those marked by $(*)$ hold, where $G \xrightarrow{U} H$ denotes $(G, U, H) \in \longrightarrow$. In the rule for binary operators, we make no restriction on U and U' but the domain of application of bin ensures that this rule will always be used with the correct static/dynamic combination of boxes. E.g., in the case of the choice operator, U or U' will always be empty.

Let D be a dynamic box expression. We will use $[D\rangle$ to denote all the box expressions derivable from D, *i.e.*, the least set of expressions containing D such that if $J \in [D\rangle$ and $J \xrightarrow{U} K$, for some $U \in \mathbb{U}$, then $K \in [D\rangle$. Moreover, $[J]_{\equiv}$ will denote the equivalence class of $\equiv$ containing J. The *full transition system* of D is $\mathsf{fts}_D \stackrel{\text{df}}{=} (V, L, A, \mathit{init})$, where $V \stackrel{\text{df}}{=} \{[J]_{\equiv} \mid J \in [D\rangle\}$ are the states; $L \stackrel{\text{df}}{=} \mathbb{U}$ are arc labels; $A \stackrel{\text{df}}{=} \left\{([J]_{\equiv}, U, [K]_{\equiv}) \in V \times \mathbb{U} \times V \;\middle|\; J \xrightarrow{U} K\right\}$ is the set of arcs; and $\mathit{init} \stackrel{\text{df}}{=} [D]_{\equiv}$ is the initial state. For a static box expression E, $\mathsf{fts}_E \stackrel{\text{df}}{=} \mathsf{fts}_{\overline{E}}$. Note that we base transition systems of box expressions on the equivalence classes of $\equiv$, rather than on box expressions themselves, since we may have $D \xrightarrow{\varnothing} J$ for two different expressions D and J, whereas in the domain of boxes, $\Sigma\,[\varnothing\rangle\,\Theta$ always implies $\Sigma = \Theta$.

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

Theorem 2. *For every G, $\mathsf{iso}_G \stackrel{\text{df}}{=} \left\{([H]_{\equiv}, \mathsf{box}(H)) \;\middle|\; [H]_{\equiv} \text{ is a node of } \mathsf{fts}_G\right\}$ is an isomorphism between the full transition systems fts_G and $\mathsf{fts}_{\mathsf{box}(G)}$.*

To define a label based operational semantics of box expressions, we first retain the structural similarity relation $\equiv$ on box expressions without any change. Next, we define moves of the form $G \xrightarrow{\Gamma} H$, where G and H are box expressions as before, and Γ is a finite multiset of labels in $\mathbb{A}_\tau$. Referring to table 1, we keep the rules marked with $(\dagger)$ with $\varnothing$ now denoting the empty multiset of labels, and U being changed to Γ, and add those marked with $(*)$ (instead of the corresponding rules based on transitions).

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. As a result, the properties concerning transition based operational semantics directly extend to the label based one. For a box expression G, the label based operational semantics is captured by its *labelled transition system*, denoted by lts_G, and defined as fts_G with each arc label U changed to $\lambda(U)$. We then obtain the following result.

Theorem 3. *For every* G, $\mathsf{iso}_G \stackrel{\text{df}}{=} \{([G]_{\equiv}, \mathsf{box}(H)) \mid [H]_{\equiv} \text{ is a node of } \mathsf{lts}_G\}$ *is an isomorphism between the labelled transition systems* lts_G *and* $\mathsf{lts}_{\mathsf{box}(G)}$.

Hence ABC supports two consistent (in a very strong sense since the corresponding transition systems are isomorphic and not only bisimilar, as in [7]) concurrent semantics for a class of process expressions with both synchronous and asynchronous communication.

The rules of the label based operational semantics are put into work below, where we use the second scenario presented in the introduction:

$$\overline{(((pb^+\|pb^+)\circledast f)\|(cb^-\circledast f))\,\mathsf{tie}\,b} \equiv (((\overline{pb^+}\|\overline{pb^+})\circledast f)\|(\overline{cb^-\circledast f}))\,\mathsf{tie}\,b$$

$$\xrightarrow{\{p\}} (((\underline{pb^+.b}\|\overline{pb^+})\circledast f)\|(\overline{cb^-\circledast f}))\,\mathsf{tie}\,b \equiv (((\underline{pb^+}\|\overline{pb^+})\circledast f)\|(\overline{cb^-.b}\circledast f))\,\mathsf{tie}\,b$$

$$\xrightarrow{\{p,c\}} (((\underline{pb^+}\|\underline{pb^+.b})\circledast f)\|(\underline{cb^-}\circledast f))\,\mathsf{tie}\,b \equiv (((pb^+\|pb^+.b)\circledast \overline{f})\|(cb^-\circledast \overline{f}))\,\mathsf{tie}\,b$$

$$\xrightarrow{\{f,f\}} (((pb^+\|pb^+.b)\circledast \underline{f})\|(cb^-\circledast \underline{f}))\,\mathsf{tie}\,b \equiv \underline{(((pb^+\|pb^+)\circledast f)\|(cb^-\circledast f))\,\mathsf{tie}\,b}$$

4 Causality in boxes and box expressions

We now discuss causality in the fragment of the ABC model without buffer-specific construct. This still leads to the possibility of deriving boxes which are non-safe, unlike in PNA (and PBC). Consider, for example, the expression $D \stackrel{\text{df}}{=} \overline{(a\|c)\circledast f}$ and the box $\Sigma \stackrel{\text{df}}{=} \mathsf{box}(D)$ in figure 6, focussing on the behaviours involving a single execution of each of the transitions of Σ.

As we have seen, the step sequences are exactly the same for D and Σ (see theorems 2 and 3). Let us investigate whether the same will hold for causality semantics. For Σ, we have three possible process nets in this case (see, *e.g.*, [2] for the definition of process nets of Petri nets), two of which are shown in figure 6: π_1 specifies that the occurrences of t_2 and t_3 are concurrent and both precede the occurrence of t_1, while π_2 specifies that the occurrences are causally ordered as $t_2t_3t_1$ (the third possibility is π_3 with the occurrences ordered $t_3t_2t_1$).

Whereas the causality expressed by π_1 is something to be expected and can easily be matched using a suitable execution of the expression D, the case of π_2 is less clear. For instance, consider the only possible (up to the structural equivalence of expressions) execution corresponding to the scenario captured by π_2:

$$\overline{(a\|c)\circledast f} \equiv (\overline{a}\|\overline{c})\circledast f \xrightarrow{\{t_2\}} (\underline{a}\|\overline{c})\circledast f \xrightarrow{\{t_3\}} (\underline{a}\|\underline{c})\circledast f \equiv (a\|c)\circledast \overline{f} \xrightarrow{\{t_1\}} (a\|c)\circledast \underline{f}$$

It is not difficult to see that in the above the occurrences of t_2 and t_3 are totally

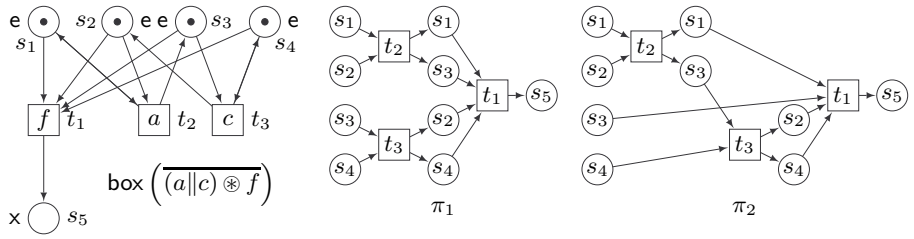

Fig. 6. A box and two of its processes.

unrelated. Indeed, if we replace the a with *any* expression (even one which is deadlocked), we still can derive:

$$\overline{(\cdots\|c)\circledast f} \equiv (\cdots\|\overline{c})\circledast f \xrightarrow{\{t_3\}} (\cdots\|\underline{c})\circledast f,$$

indicating that the overbar responsible for the occurrence of t_3 does not depend on executing anything else. Thus, on the level of box expressions, the partial order semantics does not seem to be suffering from the *token swapping* phenomenon discussed in [2] and which is illustrated by π_1 and π_2. As a result, the causality relationship is as in the case of safe nets, despite the fact that we are now working with non-safe nets. This is highly relevant both from the theoretical and practical point of view (*e.g.*, the unfolding technique based on branching processes is more efficient if token swapping is absent, see [9]). Briefly, to implement such a semantics one can annotate the overbars and underbars with the positions of the atomic actions responsible for their creation. For our example, this would yield:

$$\begin{aligned}\overline{(a\|c)\circledast f}^{\varnothing} \quad &\equiv \quad (\overline{a}^{\varnothing}\|\overline{c}^{\varnothing})\circledast f \xrightarrow{\{t_2\}} (\underline{a}_{\xi_1}\|\overline{c}^{\varnothing})\circledast f \\ &\xrightarrow{\{t_3\}} (\underline{a}_{\xi_1}\|\underline{c}_{\xi_2})\circledast f \quad \equiv \quad (a\|c)\circledast \overline{f}^{\xi_1,\xi_2} \xrightarrow{\{t_1\}} (a\|c)\circledast \underline{f}_{\xi_2}\end{aligned}$$

where $\xi_1 = v_1^{\circledast} v_1^{\|} v^a$, $\xi_2 = v_1^{\circledast} v_2^{\|} v^c$ and $\xi_3 = v_2^{\circledast} v^f$. Then we generate a causal precedence relation between occurrences of t_i and t_j provided that there is a path ξ_l from the root to a leaf of t_i labelling an overbar used to generate the occurrence of t_j. Since, in our case, $t_1 = v_2^{\circledast} \lhd v^f$, $t_2 = v_1^{\circledast} \lhd v_1^{\|} \lhd v^a$ and $t_3 = v_1^{\circledast} \lhd v_2^{\|} \lhd v^c$, the causality relation generated is exactly as in π_1, and not as in π_2.

To transfer the above way of generating causal partial orders for box expressions into the domain of boxes amounts to requiring that whenever there is a choice between two occurrences of the same place (like s_3 in figure 6 needed to generate an instance of t_3), then the one which leads to lesser causal orderings is chosen (and so neither π_2 nor π_3 would be generated).

5 Concluding remarks

In this paper, we proposed a framework which supports two consistent (in a very strong sense, since the corresponding transition systems are isomorphic and not only bisimilar) concurrent semantics for a class of process expressions with both synchronous and asynchronous communication. We eliminated the need to maintain the safeness of the non-buffer places (required in the original PNA), which

might be awkward for practical applications. It was replaced by auto-concurrency freeness which is a property still guaranteeing suitable concurrency semantics. We then discussed how a partial order semantics of boxes in a fragment of PNA could be obtained using additional annotations in process expressions.

Acknowledgements We would like to thank the referees for helpful comments. This research was supported by the Arc Jip and Epsrc Beacon projects.

References

1. T.Basten and M.Voorhoeve: An Algebraic Semantics for Hierarchical P/T Nets. *ICATPN'95*, Springer, LNCS 935 (1995) 45–65.
2. E. Best and R. Devillers: Sequential and Concurrent Behaviour in Petri Net Theory. *Theoretical Computer Science* 55 (1988) 87–136.
3. E.Best, R.Devillers and J.Hall: The Petri Box Calculus: a New Causal Algebra with Multilabel Communication. In: *Advances in Petri Nets 1992*, G.Rozenberg (Ed.). Springer, LNCS 609 (1992) 21–69.
4. E.Best, R.Devillers and M.Koutny: *Petri Net Algebra*. EATCS Monographs on TCS, Springer (2001).
5. G.Boudol and I.Castellani: Flow Models of Distributed Computations: Three Equivalent Semantics for CCS. *Information and Computation* 114 (1994) 247–314.
6. P.Degano, R.De Nicola and U.Montanari: A Distributed Operational Semantics for CCS Based on C/E Systems. *Acta Informatica* 26 (1988) 59–91.
7. R.Devillers, H.Klaudel, M.Koutny, E.Pelz and F.Pommereau: Operational Semantics for PBC with Asynchronous Communication. *HPC'02*, SCS (2002) 314–319.
8. R.Devillers, H.Klaudel, M.Koutny and F.Pommereau: Asynchronous Box Calculus. Technical Report CS-TR-759, Dept. of Comp. Sci., Univ. of Newcastle (2002).
9. J.Esparza, S.Römer and W.Vogler: An Improvement of McMillan's Unfolding Algorithm. *TACAS'96*, Springer, LNCS 1055 (1996) 87–106.
10. R.J.van Glabbeek and F.V.Vaandrager: Petri Net Models for Algebraic Theories of Concurrency. *PARLE'87*, Springer, LNCS 259 (1987) 224–242.
11. U.Goltz and R.Loogen: A Non-interleaving Semantic Model for Nondeterministic Concurrent Processes. *Fundamentae Informaticae* 14 (1991) 39–73.
12. R.Gorrieri and U.Montanari: On the Implementation of Concurrent Calculi in Net Calculi: two Case Studies. *Theoretical Computer Science* 141(1-2) (1995) 195–252.
13. C.A.R.Hoare: *Communicating Sequential Processes*. Prentice Hall (1985).
14. P.W.Hoogers, H.C.M.Kleijn and P.S.Thiagarajan: An Event Structure Semantics for General Petri Nets. *Theoretical Computer Science* 153 (1996) 129–170.
15. H.Klaudel and F.Pommereau: Asynchronous links in the PBC and M-nets. *ASIAN'99*, Springer, LNCS 1742 (1999) 190–200.
16. H.Klaudel and F.Pommereau: A concurrent and Compositional Petri Net Semantics of Preemption. *IFM'2000*, Springer, LNCS 1945 (2000) 318–337.
17. R.Milner: *Communication and Concurrency*. Prentice Hall (1989).
18. E.R.Olderog: *Nets, Terms and Formulas*. Cambridge Tracts in Theoretical Computer Science 23, Cambridge University Press (1991).
19. G.D.Plotkin: A Structural Approach to Operational Semantics. Technical Report FN-19, Computer Science Department, University of Aarhus (1981).
20. W.Reisig: *Petri Nets. An Introduction*. EATCS Monographs, Springer (1985).