Operational Semantics for PBC with Asynchronous Communication

Raymond Devillers, Hanna Klaudel, Maciej Koutny, Elisabeth Pelz, Franck Pommereau

Proc. of HPC'02, SCS, 2002. © 2002 SCS

Operational Semantics for PBC with Asynchronous Communication

R. Devillers[1], H. Klaudel[2], M. Koutny[3], E. Pelz[2], 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,pelz,pommereau}@univ-paris12.fr
[3] Department of Computing Science, University of Newcastle upon Tyne, NE1 7RU, United Kingdom.
Maciej.Koutny@newcastle.ac.uk

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

Abstract

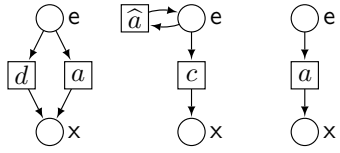
This paper presents two related algebras which can be used to specify and analyse concurrent systems with synchronous and asynchronous communications. The first algebra is based on a class of P/T-nets, called boxes, and their standard transition firing rule. It is an extension of the Petri Box Calculus (PBC). Essentially, the original model is enriched with the introduction of special 'buffer' places where different transitions (processes) may deposit and remove tokens, together with an explicit asynchronous communication operator, denoted by tie, allowing to make them 'private'. We also introduce an algebra of process expressions corresponding to such a net algebra, by augmenting the existing syntax of PBC expressions, and defining a system of SOS rules providing their operational semantics. The two algebras are related through a mapping which, for any extended box expression, returns a corresponding box with an isomorphic transition system.

INTRODUCTION

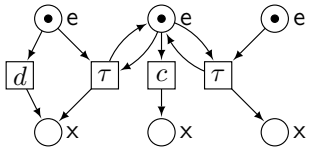
This paper is concerned with the theme of relating process algebras, such as CCS [9] and CSP [6], and Petri nets [11]. In general, the approaches proposed in the literature aim at providing a Petri net semantics to process algebras whose definition has been given independently of any Petri nets semantics (see, e.g., [5]). Another way is to translate elements from Petri nets into process algebras such as ACP [1] (see, e.g., [2]).

A specific framework and the starting point of the current paper is a concrete process algebra, called the *Petri Box Calculus* (PBC [3, 4]), designed with the aim of allowing a compositional Petri net semantics. PBC is composed of an algebra of process expressions (called *box expressions*) with a fully compositional translation into labelled safe Petri nets (called *boxes*). In this paper, the original PBC is extended by the introduction of special 'buffer' places where different transitions (processes) may deposit and remove tokens, together with an explicit asynchronous communication operator, denoted by tie, allowing to make them 'private'.

In the variant of the original PBC considered here, places are labelled by their status symbols (e for entry, x for exit, and i for internal) while transitions are labelled by CCS-like synchronous communication actions, such as a, $\widehat{a}$, and τ (similarly as in CCS, τ represents an invisible action). The operators considered are: *sequence* $E; E'$ (the execution of E is followed by that of E'); *choice* $E \Box E'$ (either E or E' can be executed); *parallel composition* $E \| E'$ (E and E' can be executed concurrently); *iteration* $E \circledast E'$ (E can be executed an arbitrary number of times, and is followed by E'); and *scoping* E sc a (all handshake synchronisations involving pairs of a- and $\widehat{a}$–labelled transitions are enforced, and after that they may no longer be executed alone). We illustrate some of the PBC constructs in figure 1, where the upper net is the translation of the box expression $(d \Box a) \| ((\widehat{a} \circledast c) \| a)$.



the box of $(d \Box a) \| ((\widehat{a} \circledast c) \| a)$



the box of $\overline{((d \Box a) \| ((\widehat{a} \circledast c) \| a))}$ sc a

Figure 1. Two expressions and the corresponding Petri boxes of the original PBC.

The operational semantics of PBC is given through SOS rules in Plotkin's style [10]. Instead of expressing

the evolutions through rules modifying the structure of the expressions, like $a.E \xrightarrow{a} E$ in CCS, the idea is to represent the current state of the evolution using overbars and underbars, respectively representing the initial and final states of the corresponding (sub)expressions. This is illustrated in figure 1, where the lower net is the initially marked scoping of the upper net w.r.t. the communication action a.

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

$$\overline{((d\Box a)\|((\widehat{a}\circledast c)\|a))}\ \mathsf{sc}\ a \equiv \overline{((d\Box a)\|((\widehat{a}\circledast c)\|a))}\ \mathsf{sc}\ a$$
$$\equiv (\overline{(d\Box a)}\|\overline{((\widehat{a}\circledast c)\|a)})\ \mathsf{sc}\ a \equiv ((\overline{d}\Box a)\|\overline{((\widehat{a}\circledast c)\|a)})\ \mathsf{sc}\ a$$

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

$$((\overline{d}\Box a)\|\overline{((\widehat{a}\circledast c)\|a)})\ \mathsf{sc}\ a \xrightarrow{\{d\}} ((\underline{d}\Box a)\|\overline{((\widehat{a}\circledast c)\|a)})\ \mathsf{sc}\ a$$

It was shown in [4, 8] that the two algebras of the original PBC are fully compatible, in the sense that a box expression and the corresponding box generate isomorphic transition systems.

Recently, [7] introduced a new set of basic boxes and a new operator on Petri boxes (denoted by tie) for the modelling of asynchronous interprocess communication. This extension is based on a set of special places s_b labelled with a link symbol b, as shown in figure 2. An intuitive meaning of a b–labelled place is that some transitions can insert tokens into it, while others can remove them, thus effecting asynchronous communication. Moreover, each PBC operator is extended in such a way that all the b–labelled places are combined into a single one (see the box of $b^+\|b^-$ in figure 2). The tie operator w.r.t. a link b, when applied to such a net, changes the status of the b–labelled place into b, defining by this the scope of the asynchronous links w.r.t. b; such a place can no longer be glued with other buffer places (see the box of $(b^+\|b^-)$ tie b in figure 2, and the box in figure 4). The b–labelled places may be viewed as a new kind of internal places, and so there is a single status b for all the link symbols.

Within the domain of box expressions, the specific device for inserting and removing messages from buffers is provided through basic expressions of the form b^+ and b^-, respectively (see the boxes of b^+ and b^- in figure 2). There is also a basic expression $b^\pm$ which both remove and add a token, in effect checking for the presence of a token (message) in the buffer place. Although the resulting model, called *PBC with Asynchronous Communication* (or PBCAC), is no longer based on safe Petri nets since the buffer places can be unbounded, the extension is conservative as the remaining places are still safe, as in the original PBC (see the box in figure 4 in which the b–labelled place is unbounded, whereas all the remaining ones are safe).

The aim of this paper is to introduce the PBCAC model which extends the original PBC with new devices supporting asynchronous interprocess communication. The model will be based on two algebras, an algebra of box expressions and an algebra of boxes, both constituting conservative extensions of their counterparts in the original PBC. On the technical level, we will extend the PBC syntax by introducing specific constructs modelling the tie operator and, in particular, the unbounded nature of the buffer places. The original operational semantics rules of PBC will be augmented, yielding a system of SOS rules providing the operational semantics of PBCAC expressions. The two algebras forming PBCAC will be related through a mapping which, for any box expression, returns a corresponding box with an isomorphic transition system.

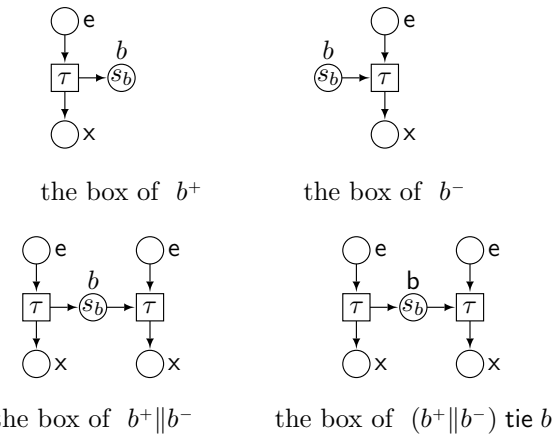


Figure 2. Asynchronous link operation.

AN ALGEBRA OF PETRI BOXES

We assume sets $\mathsf{A} \stackrel{\text{df}}{=} \{\tau, a, \widehat{a}, \ldots\}$ of *actions* and $\mathsf{B} \stackrel{\text{df}}{=} \{b, b', \ldots\}$ of *links* to be given. The former are communication actions which will be used, in particular, to model synchronous handshake communication. The latter are a device which allows one to express asynchronous communication, where the sending and receiving of a message do not need to happen simultaneously. Similarly as in CCS, the actions a and $\widehat{a}$ can synchronise and produce a silent (internal) action τ. A marked labelled net (*labelled net*, for short) is a tuple $\Sigma \stackrel{\text{df}}{=} (S, T, W, \lambda, M)$ such that: S and T are disjoint

sets of respectively *places* and *transitions*; W is a *weight* function from the set $(S\times T)\cup(T\times S)$ to the set of natural numbers $\mathbb{N}$; λ is a *labelling* function for places and transitions such that $\lambda(s)\in\{\mathsf{e},\mathsf{i},\mathsf{x},\mathsf{b}\}\cup\mathsf{B}$ (we assume that $\mathsf{e},\mathsf{i},\mathsf{x},\mathsf{b}\notin\mathsf{B}$), for every place $s\in S$, and $\lambda(t)\in\mathsf{A}$, for every transition $t\in T$; and M is a *marking*, i.e., a multiset over S. A *step* is a finite multiset of transition labels.

If the labelling of a place s in Σ is e then s is an *entry* place, if i then s is an *internal* place, if x then s is an *exit* place, if b or $b\in\mathsf{B}$ then s is a *buffer* place (closed or open, respectively). By convention, ${}^\circ\Sigma$ and Σ° denote respectively the entry and exit places of Σ. The e-, i- and x–labelled places are called *control flow* places.

All nets are assumed to be T-*restricted*, i.e., for every transition t there are control flow places s and r such that $W(t,s)>0$ and $W(r,t)>0$.

A labelled net Σ is *marked* if at least one control flow place is marked; and *unmarked* otherwise (notice that in such a case buffer places may still contain tokens). We will say that Σ is in an *entry marking* if the marking of Σ restricted to the control flow places is ${}^\circ\Sigma$, and in an *exit marking* if the marking of Σ restricted to the control flow places is Σ°. Moreover, we define $\overline{\Sigma}$ and $\underline{\Sigma}$ as, respectively, $(S,T,W,\lambda,{}^\circ\Sigma+M')$ and $(S,T,W,\lambda,\Sigma^\circ+M')$ where M' is the marking of Σ restricted to the buffer places.

A finite *step sequence* semantics for a labelled net Σ captures the potential concurrency in the behaviour of the system modelled by Σ. It is illustrated for the example given in the appendix.

The marking M of Σ is *safe* if for every control flow place $s\in S$, $M(s)\in\{0,1\}$; and it is *clean* if ${}^\circ\Sigma\subseteq M$ or $\Sigma^\circ\subseteq M$ implies that M restricted to the control places is equal to ${}^\circ\Sigma$ or Σ°, respectively. A labelled net is called *safe* (*clean*) if all its reachable markings are safe (respectively, clean).

Boxes with the tie operator

A *box* is a T-restricted labelled net Σ such that ${}^\circ\Sigma\neq\emptyset\neq\Sigma^\circ$. An unmarked box Σ will be called *static* if each marking reachable from $\overline{\Sigma}$ or from $\underline{\Sigma}$ is safe and clean. Static boxes are net counterparts of static expressions, i.e., expressions without overbars and underbars. A marked box Σ is *dynamic* if each marking reachable from its initial marking or from $\overline{\Sigma}$ or from $\underline{\Sigma}$ is safe and clean. Dynamic boxes are net counterparts of dynamic expressions, i.e., expressions with active subexpressions indicated by overbars and underbars. One can show that if Σ is a static box and Θ is derivable from $\overline{\Sigma}$ then Θ is a dynamic box.

The box operators considered in this paper can be divided in two groups: the control flow operators, and the communication ones. The first group, which consists of sequential and parallel compositions, choice and iteration, are synthesized from the refinement meta-operator [4]. Each binary operator on nets, op, is described by a net Ω_{op} with two transitions v_1 and v_2, which can be refined by nets Σ_1 and Σ_2 in the process of forming a new net $\Sigma_1\ \mathsf{op}\ \Sigma_2$. This corresponds formally to the application of a process-algebraic operator op to its two arguments.

Operators of the second group, scoping and asynchronous link, are concerned with the modelling of interprocess communication. Intuitively, scoping is an operation which combines synchronous communication and restriction. In the net $\Sigma\ \mathsf{sc}\ a$, each pair of transitions, t and u, respectively labelled by a and $\widehat{a}$, gives rise to a new τ–labelled transition which inherits the connectivity of both t and u; after that all the transitions labelled by a and $\widehat{a}$ are removed (this operation is illustrated in the lower part of figure 1).

The PBCAC scheme of introducing asynchronous communication is based on the buffer places which can hold tokens representing messages, and two operations: (i) the *merge* operation combines together buffer places labelled by the same $b\in\mathsf{B}$; and (ii) the $\mathsf{tie}\ b$ operator which encapsulates b–labelled buffer places, so that no further gluing with other buffer places is possible (in effect, it acts as a hiding operator for asynchronous communication). Formally, we proceed as follows.

Let $\Sigma\stackrel{\text{df}}{=}(S,T,W,\lambda,M)$ be a box, and $S_{\mathsf{B}}\stackrel{\text{df}}{=}\{s_b\mid b\in\mathsf{B}\wedge\lambda^{-1}(b)\neq\emptyset\}$ be a set of fresh places. The *merge* operation defines a box $\mathsf{m}(\Sigma)\stackrel{\text{df}}{=}((S\setminus\lambda^{-1}(\mathsf{B}))\cup S_{\mathsf{B}},T,W',\lambda',M')$ such that:

- The label of each new place s_b is b; otherwise the labels are unchanged.
- The marking of each new place s_b is the sum of the tokens in all b–labelled places of Σ; otherwise the marking is unchanged.
- For each new place s_b and every $t\in T$, the weight of the arc from s_b to t (from t to s_b) is the sum of weights from each b–labelled place of Σ to t (respectively, the sum of weights from t to each b–labelled place of Σ); otherwise the weights are unchanged.

For a box Σ and $b\in\mathsf{B}$, $\Sigma\ \mathsf{tie}\ b$ is defined as Σ with the status of each b–labelled place changed to b. Notice that if no b–labelled places are present, then $\Sigma\ \mathsf{tie}\ b=\Sigma$.

All original PBC operations are adapted to the new context, with the only difference being the treatment

of b- and b–labelled places (not existing in the original PBC) which are regarded as if internal, i.e., having the status i. Formally, the scoping construct remains the same, $\Sigma\ \mathsf{sc}\ a \stackrel{\text{df}}{=} \Sigma\,\mathsf{sc}_{\text{PBC}}\,a$ and, for any binary PBC operator op_{PBC}, we define $\Sigma\ \mathsf{op}\ \Sigma' \stackrel{\text{df}}{=} \mathsf{m}(\Sigma\ \mathsf{op}_{\text{PBC}}\Sigma')$. Thus, for any $b \in \mathsf{B}$ there will be at most one b–labelled place in $\Sigma\ \mathsf{op}\ \Sigma'$.

AN ALGEBRA OF EXPRESSIONS

The main new feature of PBCac with respect to the existing PBC syntax is the $E.b$ notation reflecting the token buffering feature introduced by asynchronous links. Intuitively, $E.b$ represents a system modelled by E with one extra resource inside a b–labelled buffer, which can subsequently be consumed by actions specified within E (or within F, if $E.b$ is a subexpression of F).

As in the original PBC approach, PBCac distinguishes two kinds of process expressions: the static and dynamic ones. The former model the structure of a concurrent system in which the control flow part is dormant and so no action can be executed (in Petri net terms, this corresponds to having completely unmarked control flow places). However, the system may still have some resources (asynchronous messages) stored in communication buffers (in Petri net terms, this corresponds to having marked buffer places). The dynamic expressions represent concurrent systems where the control flow part is allowed to progress (in Petri net terms, any place can be marked).

We consider the following syntax of *static* PBCac expressions, where $a \in \mathsf{A}$ and $b \in \mathsf{B}$:

$$E ::= \tau \mid a \mid \widehat{a} \mid b^+ \mid b^- \mid b^\pm \mid E\|E \mid E\Box E \mid E;E \mid E\circledast E \mid E\ \mathsf{sc}\ a \mid E\ \mathsf{tie}\ b \mid E.b$$

The syntax of *dynamic* PBCac expressions is a straightforward adaptation of that used by the original PBC (by adding the last two constructs):

$$D ::= \overline{E} \mid \underline{E} \mid D\ \mathsf{sc}\ a \mid D\|D \mid D\Box E \mid E\Box D \mid D;E \mid E;D \mid D\circledast E \mid E\circledast D \mid D.b \mid D\ \mathsf{tie}\ b$$

Moreover, we will use F to denote any static or dynamic expression. As in PBC, the expression $\overline{E}$ represents E in its initial state (in terms of nets, this corresponds to the initially marked box of E). Similarly, $\underline{E}$ represents E in its final state (in terms of nets, this corresponds to the finally marked box of E). Note that, with respect to the notations used in [3, 4], the above syntax uses slightly different symbols to denote scoping ($E\ \mathsf{sc}\ a$ instead of $[a : E]$) and iteration ($E\circledast E'$ instead of $\langle E * E']$). However, their semantics will remain unchanged. Note also that the $.b$ notation is needed for static as well as for dynamic expressions because a dormant part of a dynamic expression (represented by a static subexpression) may still have tokens in buffer places.

One of the key features of the original PBC is that all nets corresponding to box expressions are safe. Here we want to retain this property when considering only the control flow places (this is not desired for buffer places, which may contain unused resources even after the whole system has terminated). To achieve this, we impose the same syntactic constraints as it has been done in [3] (although this is not essential to preserve our final consistency property, see theorem 2 below).

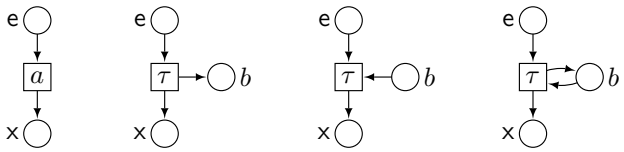


Figure 3. The nets Σ_a, Σ_{b^+}, Σ_{b^-} and $\Sigma_{b^\pm}$, respectively, used in the definition of box.

Denotational semantics

We first translate PBCac expressions into the corresponding boxes. The function box takes a static or dynamic expression F, and returns a box which is the box corresponding to F. It is defined compositionally, by induction on the structure of PBCac expressions:

$$\mathsf{box}(F\ \mathsf{op}\ F') \stackrel{\text{df}}{=} \mathsf{box}(F)\ \mathsf{op}\ \mathsf{box}(F')$$
$$\mathsf{box}(F\ \mathsf{sc}\ a) \stackrel{\text{df}}{=} \mathsf{box}(F)\ \mathsf{sc}\ a$$
$$\mathsf{box}(F\ \mathsf{tie}\ b) \stackrel{\text{df}}{=} \mathsf{box}(F)\ \mathsf{tie}\ b$$

$$\mathsf{box}(\overline{E}) \stackrel{\text{df}}{=} \overline{\mathsf{box}(E)} \qquad \mathsf{box}(\underline{E}) \stackrel{\text{df}}{=} \underline{\mathsf{box}(E)}$$
$$\mathsf{box}(a) \stackrel{\text{df}}{=} \Sigma_a \qquad \mathsf{box}(b^+) \stackrel{\text{df}}{=} \Sigma_{b^+}$$
$$\mathsf{box}(b^-) \stackrel{\text{df}}{=} \Sigma_{b^-} \qquad \mathsf{box}(b^\pm) \stackrel{\text{df}}{=} \Sigma_{b^\pm}$$

where $a \in \mathsf{A}$, $b \in \mathsf{B}$, op is a binary PBCac operator (i.e., $\mathsf{op} \in \{;, \Box, \|, \circledast\}$), and the boxes $\Sigma_a, \Sigma_{b^+}, \Sigma_{b^-}, \Sigma_{b^\pm}$ are given in figure 3. Moreover, $\mathsf{box}(F.b)$ is defined thus: if there is a place s_b in $\mathsf{box}(F)$ (there is only one such place due to the systematic merges of the buffer places), then we add one token to it; otherwise we add to $\mathsf{box}(F)$ an isolated b–labelled place s_b with exactly one token.

Theorem 1. *For every static (dynamic) box expression F, $\mathsf{box}(F)$ is a static (respectively, dynamic) box.*

Operational semantics

We will now introduce the semantics of PBCAC expressions based on execution rules. In the appendix, we demonstrate how they can be applied to a simple system of three concurrent processes.

Structurally equivalent expressions

The operational semantics of PBC identifies structurally equivalent expressions as those for whom the corresponding boxes are identical, and whose operational semantics is the same.

In addition to the structural equivalence rules from [3], we need new rules reflecting the asynchronous communication features of PBCAC. We define them below, where $b, b' \in \mathsf{B}$ and op is a binary PBCAC operator:

$$\begin{aligned}
\overline{E}.b &\equiv \overline{E.b} \\
\underline{E.b} &\equiv \underline{E}.b \\
\overline{E \;\mathsf{tie}\; b} &\equiv \overline{E} \;\mathsf{tie}\; b \\
\underline{E} \;\mathsf{tie}\; b &\equiv \underline{E \;\mathsf{tie}\; b} \\
(F.b) \;\mathsf{sc}\; a &\equiv (F \;\mathsf{sc}\; a).b \\
(F_1.b) \;\mathsf{op}\; F_2 &\equiv (F_1 \;\mathsf{op}\; F_2).b \\
F_1 \;\mathsf{op}\; (F_2.b) &\equiv (F_1 \;\mathsf{op}\; F_2).b \\
(F.b) \;\mathsf{tie}\; b' &\equiv (F \;\mathsf{tie}\; b').b \qquad \text{if } b \neq b'
\end{aligned}$$

The first four rules directly follow those of the original PBC. The remaining ones capture the fact that the asynchronous message, produced by a b^+ expression and represented by $.b$, can freely move within an expression in order to be received by b^- links. However, it may never cross the boundary imposed by the $\mathsf{tie}\; b$ operator (notice that the last rule means that moving outside a tie context is only allowed if $b \neq b'$).

SOS execution rules

The operational semantics has moves of the form $F_1 \xrightarrow{\Gamma} F_2$ such that F_1 and F_2 are box expressions and Γ is a finite multiset of actions in A. In addition to those defined for PBC, we need the following new rules introduced specifically to deal with asynchronous communication (below $b \in \mathsf{B}$):

$$\overline{b^+} \xrightarrow{\{\tau\}} \underline{b^+}.b \qquad \overline{b^-}.b \xrightarrow{\{\tau\}} \underline{b^-}$$

$$\overline{b^\pm}.b \xrightarrow{\{\tau\}} \underline{b^\pm}.b \qquad \frac{D \xrightarrow{\Gamma} D'}{D \;\mathsf{tie}\; b \xrightarrow{\Gamma} D' \;\mathsf{tie}\; b \quad D.b \xrightarrow{\Gamma} D'.b}$$

MAIN RESULT

A common way to represent the branching structure of a concurrent system is to use *(labelled) transition systems*. In this paper, a transition system is a rooted, possibly infinite tree $\mathsf{ts} \stackrel{\text{df}}{=} (V, L, A, v_{in}, l)$ where: V is a set of nodes; L a set of arc labels; $A \subseteq V \times L \times V$ a set of arcs; v_{in} is the root; and l is a labelling function which with every node associates a state of the concurrent system modelled by ts. We associate a transition system both to: (i) an SOS-semantics of a (static or dynamic) PBC expression; and (ii) to an associated Petri net. We then show that the two semantics are equivalent, through equivalences defined at the transition system level.

The transition system generated by a marked labelled net Σ is defined as the smallest tree $\mathsf{ts}_\Sigma \stackrel{\text{df}}{=} (V, L, A, v_{in}, l)$ such that: v_{in} is a root labelled by Σ; if $v \in V$ is a node labelled by a box Θ then for every non-empty step Γ, if $\Theta\ [\Gamma\rangle\ \Psi$ then there is an arc $(v, \Gamma, w) \in A$ such that the label of w is Ψ. In other words, ts_Σ is the labelled reachability tree of Σ. The labelled transition system generated by an unmarked labelled net Σ is $\mathsf{ts}_\Sigma \stackrel{\text{df}}{=} \mathsf{ts}_{\overline{\Sigma}}$.

Let $[F]$ denote the equivalence class of the relation $\equiv$ containing a box expression F. The transition system generated by a dynamic expression D is defined as the smallest tree $\mathsf{ts}_D \stackrel{\text{df}}{=} (V, L, A, v_{in}, l)$ such that: v_{in} is a root labelled by $[D]$; if $v \in V$ is a node labelled by $[H]$ then for every non-empty step Γ, if $[H] \xrightarrow{\Gamma} [J]$ then there is an arc $(v, \Gamma, w) \in A$ such that the label of w is $[J]$. The labelled transition system generated by a static expression E is $\mathsf{ts}_E \stackrel{\text{df}}{=} \mathsf{ts}_{\overline{E}}$.

With the above notations, we can formulate the central property of the PBCAC model.

Theorem 2. *For every box expression F, ts_F and $\mathsf{ts}_{\mathsf{box}(F)}$ are isomorphic transition systems. Moreover, the isomorphism preserves the initial or final aspect of the corresponding nodes.*

CONCLUSION

In this paper, we proposed a framework which supports two consistent concurrent semantics for a class of process expressions with both synchronous and asynchronous communication. Such a model can be used, in particular, to give a semantics of programming languages with timing constraints and exceptions. In our future work, we will aim at handling also recursion, i.e., process definitions of the form $X \stackrel{\text{df}}{=} E$.

This research was partially supported by the ARC JIP and EPSRC BEACON projects.

References

1. J.Baeten and W.P.Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press (1990).
2. T.Basten and M.Voorhoeve. "An Algebraic Semantics for Hierarchical P/T Nets." Proc. of *ICATPN'95*, Springer, LNCS 935 (1995) 45–65.
3. E.Best, R.Devillers and M.Koutny: A Unified Model for Nets and Process Algebras. Handbook of Process Algebra. Jan A. Bergstra, Alban Ponse, Scott A. Smolka, (Eds.) Elsevier (2001) 873–944.
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. C.A.R.Hoare. *Communicating Sequential Processes*. Prentice Hall (1985).
7. H.Klaudel and F.Pommereau. *Asynchronous links in the PBC and M-nets*. Springer, LNCS 1742 (1999). 190–200
8. M.Koutny and E.Best. "Fundamental Study: Operational and Denotational Semantics for the Box Algebra." *Theoretical Computer Science* 211 (1999) 1–83.
9. R.Milner. *Communication and Concurrency*. Prentice Hall (1989).
10. G.D.Plotkin. "A Structural Approach to Operational Semantics." Technical Report FN-19, Computer Science Department, University of Aarhus (1981).
11. W.Reisig. *Petri Nets. An Introduction*. EATCS Monographs on TCS, Springer (1985).

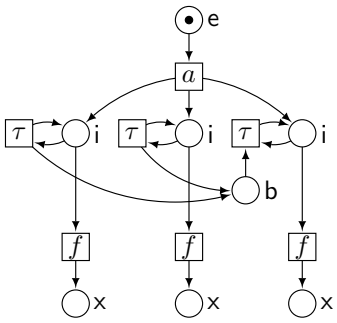
APPENDIX: AN EXAMPLE

Figure 4 presents a model of a concurrent system composed of two producer processes (represented by $(b^+ \circledast f) \| (b^+ \circledast f)$) and one consumer process (represented by $b^- \circledast f$), operating in parallel. In the producers, the b^+ actions can repeatedly send tokens to a buffer place, which then can be received using the b^- actions in the consumer process.

The two-producers/one-consumer system is encapsulated by an application of the tie operator, which makes the buffer place b–labelled. This place is therefore no longer available for future asynchronous links. The whole system is preceded by a startup action a.

The step sequence semantics is illustrated for the above example using the following execution scenario:

- the system is initialised by executing the a–labelled transition;
- the two producers send a token each to the b–labelled place;
- the consumer takes one of the two tokens from the b–labelled place and, at the same time, the first producer sends another token there;



the box of $a; ((((b^+ \circledast f) \| (b^+ \circledast f)) \| (b^- \circledast f))$ tie $b)$

Figure 4. Two-producers/one-consumer system.

- the two producers and the consumer finalise their operation by simultaneously executing the f–labelled transitions.

Such a scenario corresponds to the labelled step sequence $\Sigma\ [\{a\}\{\tau,\tau\}\{\tau,\tau\}\{f,f,f\}\rangle\ \Sigma'$ where Σ' is Σ with two tokens in the b–labelled place, one token in each of the x–labelled places, and no token elsewhere (i.e., Σ' is in an exit marking). The same effect can be achieved using the rules of the operational semantics, as shown below:

$$
\begin{aligned}
&\overline{a; ((((b^+ \circledast f) \| (b^+ \circledast f)) \| (b^- \circledast f))\ \mathsf{tie}\ b)} \\
&\quad\equiv\quad \overline{a}; ((((b^+ \circledast f) \| (b^+ \circledast f)) \| (b^- \circledast f))\ \mathsf{tie}\ b) \\
&\quad\xrightarrow{\{a\}}\quad \underline{a}; ((((b^+ \circledast f) \| (b^+ \circledast f)) \| (b^- \circledast f))\ \mathsf{tie}\ b) \\
&\quad\equiv\quad a; \overline{((((b^+ \circledast f) \| (b^+ \circledast f)) \| (b^- \circledast f))}\ \mathsf{tie}\ b) \\
&\quad\equiv\quad a; ((((\overline{b^+} \circledast f) \| (\overline{b^+} \circledast f)) \| (\overline{b^-} \circledast f))\ \mathsf{tie}\ b) \\
&\quad\xrightarrow{\{\tau,\tau\}}\quad a; ((((\underline{b^+}.b \circledast f) \| (\underline{b^+}.b \circledast f)) \| (\overline{b^-} \circledast f))\ \mathsf{tie}\ b) \\
&\quad\equiv\quad a; ((((\underline{b^+}.b \circledast f) \| (\underline{b^+} \circledast f).b) \| (\overline{b^-} \circledast f))\ \mathsf{tie}\ b) \\
&\quad\equiv\quad a; ((((\underline{b^+}.b \circledast f) \| (\underline{b^+} \circledast f)).b \| (\overline{b^-} \circledast f))\ \mathsf{tie}\ b) \\
&\quad\equiv\quad a; ((((\underline{b^+}.b \circledast f) \| (\underline{b^+} \circledast f)) \| (\overline{b^-} \circledast f)).b\ \mathsf{tie}\ b) \\
&\quad\equiv\quad a; ((((\underline{b^+}.\underline{b} \circledast f) \| (\underline{b^+} \circledast f)) \| (\overline{b^-} \circledast f).b)\ \mathsf{tie}\ b) \\
&\quad\equiv\quad a; ((((\overline{b^+}.\overline{b} \circledast f) \| (\underline{b^+} \circledast f)) \| (\overline{b^-}.b \circledast f))\ \mathsf{tie}\ b) \\
&\quad\equiv\quad a; ((((\overline{b^+}.b \circledast f) \| (\underline{b^+} \circledast f)) \| (\overline{b^-}.b \circledast f))\ \mathsf{tie}\ b) \\
&\quad\xrightarrow{\{\tau,\tau\}}\quad a; ((((\underline{b^+}.b.b \circledast f) \| (\underline{b^+} \circledast f)) \| (\underline{b^-} \circledast f))\ \mathsf{tie}\ b) \\
&\quad\equiv\quad a; ((((b^+.b.b \circledast \overline{f}) \| (b^+ \circledast \overline{f})) \| (b^- \circledast \overline{f}))\ \mathsf{tie}\ b) \\
&\quad\xrightarrow{\{f,f,f\}}\quad a; ((((b^+.b.b \circledast \underline{f}) \| (b^+ \circledast \underline{f})) \| (b^- \circledast \underline{f}))\ \mathsf{tie}\ b) \\
&\quad\equiv\quad \underline{a; ((((b^+.b.b \circledast f) \| (b^+ \circledast f)) \| (b^- \circledast f))\ \mathsf{tie}\ b)}
\end{aligned}
$$