Extending the Petri Box Calculus with Time

Olga Marroquín Alonso and David de Frutos Escrig

Departamento de Sistemas Informáticos y Programación
Universidad Complutense de Madrid, E-28040 Madrid, Spain
{alonso,defrutos}@sip.ucm.es

Abstract. PBC (*Petri Box Calculus*) is a process algebra where real parallelism of concurrent systems can be naturally expressed. One of its main features is the definition of a denotational semantics based on Petri nets, which emphasizes the structural aspects of the modelled systems. However, this formal model does not include temporal aspects of processes, which are necessary when considering real-time systems. The aim of this paper is to extend the existing calculus with those temporal aspects. We consider that actions are not instantaneous, that is, their execution takes time. We present an operational semantics and a denotational semantics based on timed Petri nets. Finally, we discuss the introduction of other new features such as time-outs and delays. Throughout the paper we assume that the reader is familiar with both Petri nets and PBC.

1 Introduction

Formal models of concurrency are widely used to specify concurrent and distributed systems. In this research field, process algebras and Petri nets are well-known. Each of them has its own advantages and drawbacks: In Petri nets and their extensions one can make assertions about events, even if causality relations are not given explicitly. Emphasis is put on the partial order of events and on structural aspects of the modelled systems. However, their algebraic basis is poor, thus modelling and verification of systems are affected. On the other hand, the main feature of process algebras is the simple algebraic characterization of the behaviour of each of the syntactic operators, although it is true that in most of the cases, we obtain it by losing important information concerning event causality.

Recently, a new process algebra, PBC (*Petri Box Calculus*) [1,2,3], has arisen from the attempts to combine the advantages of both Petri nets and process algebras. When defining PBC the starting point were Petri nets and not any well known process algebra, so their authors looked for a suitable one whose operators could be easily defined on Petri nets. As a consequence, they obtained a Petri net's algebra which can be seen as the denotational semantics of PBC [1, 2,3] and since Petri nets are endowed with a natural operational semantics, we also also derive an operational semantics of the given process algebra which can be also directly defined by means of Plotkin-like syntax-guided rules.

J.-M. Colom and M. Koutny (Eds.): ICATPN 2001, LNCS 2075, pp. 303–322, 2001.

Nevertheless, these models do not include any temporal information, and it is obvious that some kind of quantitative time representation is needed for the description of real-time systems. Many timed process algebras have been proposed. Among them we will mention: timed CCS [11], temporal CCS [7], timed CSP [10] and timed Observations [8]. Besides, since long time ago we have several timed extensions of Petri nets like Petri nets with time [6], and timed Petri nets [9].

The aim of this paper is to extend PBC with time maintaining its main properties and the basic concepts in which it is based. In this way, we propose TPBC (*Timed Petri Box Calculus*). Our model differs from the previous approach by M. Koutny [4] in several relevant aspects, among which we mention the duration of actions and the nonexistence of illegal action occurrences. Both models correspond to different ways of capturing time information, and as a consequence they are not in competition but are complementary.

Here we consider a discrete time domain. Most of the systems which we are interested in can be modelled under this hypothesis, and some of the definitions and results in the paper can be presented in a more simple way. Our results cannot be generalized to the case of continuous time, because of the fact that discrete time cannot be considered just as a simplification of continuous time.

In order to improve the readability of the paper, we will layer the presentation: first we present a simple extension in which actions have a minimal duration, that is, we allow that their executions will take more time than expected. We can find an intuitive justification of this fact considering the behaviour in practice of real-time systems. One usually knows which is the minimum time needed to execute a task, but depending on how it interacts with the environment, it could take more time than expected. Besides, even if we consider that the duration of each action is fixed, when we execute an action α whose minimal duration is d but the real duration is $d' > d$, we could consider that we are just representing a delayed execution of the action, which would start after $d' - d$ time units, in such a way that d would still be the effective duration of the action.

In this simplified timed extension there is no limit to the time a process can be idle without changing its state. So, we cannot model in it any kind of urgency or time-out. In order to do it, we present in Section 6 of the paper a more elaborate extension where these characteristics are introduced.

As we said before, we introduce time information by means of duration of actions instead of combining delays and instantaneous actions. There are several reasons that justify our choice. First, one can find in the literature both kind of timed models; since Koutny has already investigated the other case, by studying here the case of actions with duration we are somehow completing the picture. It is true that similar results to these in the paper could be obtained for the case of delays and instantaneous actions, although the corresponding translation could be non-immediate in all the cases. But it is not the consideration of actions with duration what makes our models rather more complicated than Koutny's one: The introduced complications are necessary in order to avoid *illegal action occurrences*. We will compare more in depth the two timed models in the last sections of the paper.

2 The TPBC Language

Although probably it would be more than desirable, by lack of space we cannot give here a fast introduction to PBC. Unfortunately, this is a technically involved model whose presentation require several pages [1,2,3]. But you can also infer from our paper the main characteristics of PBC, just by abstracting all the references to time in it.

Throughout the paper we use standard mathematical notation. In particular, the set of finite multisets over a set $\mathcal{S}$ is denoted by $\mathcal{M}(\mathcal{S})$, and defined as the set of functions $\alpha : \mathcal{S} \longrightarrow I\!N$ such that $\{ \, s \in \mathcal{S} \mid \alpha(s) \neq 0 \, \}$ is finite. $\varnothing$ denotes the empty multiset, and $\{a\}$ is a unitary multiset containing a.

To define the syntax of TPBC, we consider a countable alphabet of *labels* $\mathcal{A}$, which will denote atomic actions. In order to support synchronization we assume the existence of a bijection $\widehat{\ } : \mathcal{A} \longrightarrow \mathcal{A}$, called *conjugation*, by means of which we associate to each label $a \in \mathcal{A}$ a corresponding one $\hat{a} \in \mathcal{A}$. This function must satisfy the following property: $\forall a \in \mathcal{A} \quad \hat{a} \neq a \, \wedge \, \hat{\hat{a}} = a$.

As in plain PBC, the basic actions of TPBC are finite multisets of labels, called *bags*. In the prefix operator of the language each bag $\alpha \in \mathcal{M}(\mathcal{A})$ carries a duration $d \in I\!N^+$, thus we obtain the *basic action* $\alpha : d$. In the following, we will denote by $\mathcal{BA} = \mathcal{M}(\mathcal{A}) \times I\!N^+$ the set of those basic actions.

The rest of the syntactic operators in TPBC are those in PBC. Although recursive processes have not been included, the calculus is not finite: We have infinite behaviours due to the presence of the iteration operator.

Definition 1 (Static expressions). *A **static expression** of TPBC is any expression generated by the following BNF grammar:*

$$E ::= \alpha : d \mid E ; E \mid E \Box E \mid E \| E \mid [E * E * E] \mid E[f] \mid E \, \mathtt{sy} \, a \mid E \, \mathtt{rs} \, a \mid \, [a : E]$$

where $d \in I\!N^+$ and $f : \mathcal{A} \longrightarrow \mathcal{A}$ is a conjugate-preserving function. The set of static expressions of TPBC is denoted by $Expr^s$, and we use letters E and F to denote its elements.

3 Operational Semantics

The operational semantics of TPBC is defined by means of a labelled transition system including two types of transitions: instantaneous transitions, which relate equivalent processes, and non-instantaneous transitions, which express how processes evolve due to the execution of actions and the progress of time. The set of states of this transition system corresponds to a new class of expressions, the so called *dynamic expressions*.

Definition 2 (Dynamic expressions). *A **dynamic expression** of TPBC is any expression generated by the following BNF grammar:*

$$\begin{aligned} G ::= {} & \overline{E} \mid \underline{E} \mid \widetilde{\alpha : d, d'} \mid G ; E \mid E ; G \mid G \Box E \mid E \Box G \mid G \| G \mid \\ & [G * E * E] \mid [E * G * E] \mid [E * E * G] \mid G[f] \mid G \, \mathtt{sy} \, a \mid G \, \mathtt{rs} \, a \mid [a : G] \end{aligned}$$

where $d \in \mathbb{N}^+$, $d' \in \mathbb{N}$ and f is a conjugate-preserving function from $\mathcal{A}$ to $\mathcal{A}$. The set of dynamic expressions of TPBC will be denoted by $Expr^d$, and we use letters G and H to represent its elements.

In the definition above the static expressions of the calculus are marked with three types of barring: *overlining*, *underlining* and *executing barring*. The first two have the same meaning as in PBC: $\overline{E}$ denotes that E has been activated and it offers all the behaviours E represents, whereas $\underline{E}$ denotes that the process E has reached its final state and the only move it can perform is letting time pass. Finally, the dynamic expression $\widetilde{\alpha : d}, d'$ represents that the action $\alpha \in \mathcal{M}(\mathcal{A})$ has begun its execution some time ago with a minimum duration of d time units, and from now on its execution will take d' time units until it terminates.

Since we are interested in expressing locally the passage of time, and it is at the level of basic actions where this can be done in a proper way, the overline operator will be distributed over the current expression until a basic action is reached. Then the *executing bar* $\widetilde{\quad}$ will only be applied to this kind of actions.

As we already mentioned, non-instantaneous transitions represent the execution of actions and their labels indicate which bags have just begun to execute together with their durations. This information is expressed by means of *timed bags*, $\alpha_{d'}$, which consist of a bag α and a temporal annotation $d' \in \mathbb{N}^+$. The set of timed bags will be denoted by $\mathcal{TB}$.

Timed bags express the first level of concurrency we can distinguish in a concurrent system. They represent the simultaneous execution of atomic actions in the same component of the system. To cover also a second level, which represents the concurrent evolution of the different components of the system, we introduce *timed multibags*, which are finite multisets of timed bags.

Thus non-instantaneous transitions have the form $G \xrightarrow{\Gamma} G'$, where $\Gamma \in \mathcal{M}(\mathcal{TB})$. This can be interpreted as follows: process G has changed its state to G' by starting to execute the multiset of timed bags Γ during one unit of time. More in general, we assume that each labelled transition represents the passing of one time unit. As a consequence, the execution of a multibag Γ is only observable (at the level of labels of the transition system) at the first instant of it. Afterwards, we will let time to progress until the execution of Γ terminates, although in between the execution of some other actions whose performance do not need the termination of Γ could be initiated.

The passage of one time unit without starting the execution of any new bag is represented by transitions labelled by $\varnothing$. Since for any overlined or underlined process we let time pass without any change in the state, we have the rules:

$$\overline{E} \xrightarrow{\varnothing} \overline{E}\ (\mathbf{V1}) \qquad \underline{E} \xrightarrow{\varnothing} \underline{E}\ (\mathbf{V2})$$

Instantaneous transitions take the form $G \longleftrightarrow G'$, and their intended meaning is that the involved process has these two different syntactic representations: G and G'. Therefore this kind of transitions relates expressions with the same operational behaviour.

3.1 Transition Rules

In this section we first present those transition rules that represent the timed aspects of our model. They are basic actions rules and synchronization rules. We also provide the operational semantics of iteration, since it is not common in most of process algebras. The rest of the operators behave as in the untimed model [1,2,3]. A complete set of rules will appear in the PhD thesis of the first author (see [5] for a partial preliminary version).

$$\overline{\alpha : d} \xrightarrow{\{\alpha_{d'}\}} \alpha : \widetilde{d, d'} - 1 \text{ if } d' \geq d \ \textbf{(B1)} \quad \alpha : \widetilde{d, d'} \xrightarrow{\varnothing} \alpha : \widetilde{d, d'} - 1 \text{ if } d' > 0 \ \textbf{(B2)}$$

$$\widetilde{\alpha : d}, 0 \longleftrightarrow \underline{\alpha : d} \quad \textbf{(B3)}$$

Operational semantics of *basic actions*

$$\overline{E \ \mathsf{sy}\ a} \longleftrightarrow \overline{E} \ \mathsf{sy}\ a \quad \textbf{(S1)}$$

$$\frac{G \longleftrightarrow G'}{G \ \mathsf{sy}\ a \longleftrightarrow G' \ \mathsf{sy}\ a} \ \textbf{(S2a)} \quad \frac{G \xrightarrow{\mathit{\Gamma}} G'}{G \ \mathsf{sy}\ a \xrightarrow{\mathit{\Gamma}} G' \ \mathsf{sy}\ a} \ \textbf{(S2b)}$$

$$\frac{G \ \mathsf{sy}\ a \xrightarrow{\{\{\alpha + \{a\}\}_{d'}\} + \{\{\beta + \{\hat{a}\}\}_{d'}\} + \mathit{\Gamma}} G' \ \mathsf{sy}\ a}{G \ \mathsf{sy}\ a \xrightarrow{\{\{\alpha + \beta\}_{d'}\} + \mathit{\Gamma}} G' \ \mathsf{sy}\ a} \quad \textbf{(S2c)}$$

$$\underline{E} \ \mathsf{sy}\ a \longleftrightarrow \underline{E \ \mathsf{sy}\ a} \quad \textbf{(S3)}$$

Operational semantics of *synchronization*

$$\overline{[E * F * E']} \longleftrightarrow [\overline{E} * F * E'] \quad \textbf{(It1)}$$

$$\frac{G \longleftrightarrow G'}{[G * F * E] \longleftrightarrow [G' * F * E]} \ \textbf{(It2a)} \quad \frac{G \xrightarrow{\mathit{\Gamma}} G'}{[G * F * E] \xrightarrow{\mathit{\Gamma}} [G' * F * E]} \ \textbf{(It2b)}$$

$$[\underline{E} * F * E'] \longleftrightarrow [E * \overline{F} * E'] \ \textbf{(It2c)} \quad [\underline{E} * F * E'] \longleftrightarrow [E * F * \overline{E'}] \ \textbf{(It2d)}$$

$$\frac{G \longleftrightarrow G'}{[E * G * E'] \longleftrightarrow [E * G' * E']} \ \textbf{(It3a)} \quad \frac{G \xrightarrow{\mathit{\Gamma}} G'}{[E * G * E'] \xrightarrow{\mathit{\Gamma}} [E * G' * E']} \ \textbf{(It3b)}$$

$$[E * \underline{F} * E'] \longleftrightarrow [E * \overline{F} * E'] \ \textbf{(It3c)} \quad [E * \underline{F} * E'] \longleftrightarrow [E * F * \overline{E'}] \ \textbf{(It3d)}$$

$$\frac{G \longleftrightarrow G'}{[E * F * G] \longleftrightarrow [E * F * G']} \ \textbf{(It4a)} \quad \frac{G \xrightarrow{\mathit{\Gamma}} G'}{[E * F * G] \xrightarrow{\mathit{\Gamma}} [E * F * G']} \ \textbf{(It4b)}$$

$$[E * F * \underline{E'}] \longleftrightarrow \underline{[E * F * E']} \quad \textbf{(It5)}$$

Operational semantics of *iteration*

Rule **(B1)** states that basic processes can only leave its initial state by starting the execution of the corresponding bag. The duration of this execution will be greater or equal than the annotated minimum duration. Once this execution starts, the process will let time pass until its termination (rule **(B2)**). Then the basic action has finished its execution, what is represented by rule **(B3)**.

The synchronization is activated whenever its first argument becomes active (rule **(S1)**), but synchronization is not forced, so that $G \ \mathsf{sy}\ a$ can mimic all the behaviours of G (rule **(S2b)**). Rule **(S2c)** shows what happens when the process synchronizes with itself: The timed bags involved in the operation must have the same real duration, and they join together in a new timed bag in which we have removed a pair of labels $(a, \hat{a})$ such that each component is in a different multiset. Finally, the process finishes when its argument terminates, as indicated by rule **(S3)**. All the rules can be applied on any equivalent state of G (rule **(S2a)**).

Iteration rules define the behaviour of this syntactic operator. Control is transmitted from each argument to some other (either the next, or the same in the case of the second argument), until the last one terminates. More in detail, rules **(It2c)** and **(It2d)** state that once the entry condition has finished (first argument) we can choose between executing the loop body (second argument) or the exit condition (third argument). Rules **(It3c)** and **(It3d)** state that each time the second argument terminates, we can choose between executing it again or we advance to execute the last argument.

4 Denotational Semantics

The denotational semantics of any language is defined by means of a function which maps the set of its expressions into the adequate semantic domain. In this way we associate to each expression of the language an object which reflects its structure and behaviour. The denotational semantics of TPBC is based on timed Petri nets, whose transitions have a duration. A *labelled timed Petri net*, denoted by TPN, is a tuple $(P, T, F, W, \delta, \lambda)$ such that P and T are disjoint sets of *places* and *transitions*; $F \subseteq (P \times T) \cup (T \times P)$ is the set of *arcs* of the net; W is a *weight function* from F to the set of positive natural numbers $\mathbb{N}^+$; δ is a function from the transition set T to $\mathbb{N}^+$ that defines the duration of each transition; and λ is a function from $P \cup T$ into a set of labels $\mathcal{L}$. In our case, λ maps elements in P into the set $\{\mathtt{e}, \mathtt{i}, \mathtt{x}\}$ and transitions in T into the set $\mathcal{C} = \mathcal{P}(\mathcal{M}(\mathcal{BA}) \backslash \{\varnothing\} \times \mathcal{BA})$. Its intended meaning is the same as in PBC, that is, we consider that a net has three types of places: *entry places* (those with $\lambda(p) = \mathtt{e}$), *internal places* (those with $\lambda(p) = \mathtt{i}$), and *exit places* (those with $\lambda(p) = \mathtt{x}$). Moreover, each transition v is labelled with a binary relation $\lambda(v) \subseteq \mathcal{M}(\mathcal{BA}) \backslash \{\varnothing\} \times \mathcal{BA}$, whose elements are pairs of the form $(\{\alpha_1 : d_1, \alpha_2 : d_2, \ldots, \alpha_n : d_n\}, \alpha : d)$. The informal meaning of such a pair is that the behaviour represented by the multiset $\{\alpha_1 : d_1, \alpha_2 : d_2, \ldots, \alpha_n : d_n\}$ will be substituted by the execution of the bag α with a minimum duration of d time units. The most usual binary relations are the following:

1. Constant relation: $\rho_{\alpha:d} = \{\ (\{\beta : d\}, \alpha : d)\ \}$.
2. Identity: $\rho_{id} = \{\ (\{\alpha : d\}, \alpha : d)\ \}$.
3. Synchronization: $\rho_{\mathtt{sy}\ a}$ is defined as the smallest relation satisfying:
   - $\rho_{id} \subseteq \rho_{\mathtt{sy}\ a}$,
   - $(\Gamma, \alpha + \{a\} : d_1), (\Delta, \beta + \{\hat{a}\} : d_2) \in \rho_{\mathtt{sy}\ a}$ $$\Longrightarrow (\Gamma + \Delta, \alpha + \beta : \max\{d_1, d_2\}) \in \rho_{\mathtt{sy}\ a}.$$
4. Basic relabelling: $\rho_{[f]} = \{\ (\{\alpha : d\}, f(\alpha) : d) \mid f(\hat{a}) = \widehat{f(a)}\ \forall a \in \mathcal{A}\ \}$.
5. Restriction: $\rho_{\mathtt{rs}\ a} = \{\ (\{\alpha : d\}, \alpha : d) \mid a, \hat{a} \notin \alpha\ \}$.
6. Hiding: $\rho_{[a:\_]} = \{\ (\Gamma, \alpha : d) \in \rho_{\mathtt{sy}\ a} \mid a, \hat{a} \notin \alpha\ \}$.

As it is done in PBC, we distinguish two kinds of nets: *plain nets*, whose transitions are labelled with a constant relation; and *operator nets*, in which transitions are labelled with non-constant relations.

Only plain nets will be marked, and therefore they are the only ones able to fire transitions. Due to this fact, it is not necessary to include time information in operator nets. The formal definitions are the following:

Definition 3 (Timed plain net). *A* ***timed plain net*** $N = (P, T, F, W, \delta, \lambda)$ *is a timed Petri net such that* $\lambda : P \cup T \longrightarrow \{\mathtt{e}, \mathtt{i}, \mathtt{x}\} \cup \{\rho_{\alpha:d} | \alpha \in \mathcal{M}(\mathcal{A}), d \in \mathbb{N}^+\}$ *where* $\forall p \in P \;\; \lambda(p) \in \{\mathtt{e}, \mathtt{i}, \mathtt{x}\}$ *and* $\forall v \in T \;\; \lambda(v) = \rho_{\alpha:d}$ *with* $\delta(v) = d$.

Definition 4 (Operator net). *An* ***operator net*** N *is a labelled Petri net* (P, T, F, W, λ) *such that* $\lambda : P \cup T \longrightarrow \{\mathtt{e}, \mathtt{i}, \mathtt{x}\} \cup \mathcal{C}$ *where* $\forall p \in P \;\; \lambda(p) \in \{\mathtt{e}, \mathtt{i}, \mathtt{x}\}$ *and* $\forall v \in T \;\; \lambda(v) \in \mathcal{C} \backslash \{ \rho_{\alpha:d} \mid \alpha \in \mathcal{M}(\mathcal{A}), d \in \mathbb{N}^+ \}$.

To support the duration of transitions, markings of timed plain nets will be constituted by two components, M^1 and M^2. M^1 represents the available marking of the net, that is, where the tokens are and how many of them are. M^2 is the multiset of transitions currently in execution, each one carrying the time units its execution will still take from now on. Formally speaking, if $N = (P, T, F, W, \delta, \lambda)$ is a timed plain net, a *marking* M of N is a pair (M^1, M^2) where $M^1 \in \mathcal{M}(P)$ and M^2 is a finite multiset of tuples in $T \times \mathbb{N}^+$. We say that a transition v is in M^2, $v \in M^2$, iff there is some $d' \in \mathbb{N}^+$ such that $(v, d') \in M^2$. In order to visualize the set of transitions in execution, and also simplify some results, it is useful to introduce the derived concept of *frozen token*. A place $p \in P$ is occupied by a *frozen token* iff there is a transition $v \in T$ such that $(p, v) \in F$ and $v \in M^2$. In this case, if there exists only one value d' for v, we denote it by $rem(v)$. Then we define the *token-marking* $T(M)$ associated to $M = (M^1, M^2)$ by $T(M) = (M^1, \overline{M}^2)$ where $\overline{M}^2$ is the multiset of places defined by $\overline{M}^2(p) = \sum\limits_{(p,v) \in F,\; d \in \mathbb{N}^+} M^2(v, d)$. In the following we will call *available tokens* to the ordinary tokens in a marking in order to avoid confusion with frozen tokens.

A marking $M = (M^1, M^2)$ is *safe* if each place is occupied, at most, by one token, either ordinary or frozen. That is, $\forall p \in P$

$$M^1(p) + \overline{M}^2(p) \leq 1$$

A safe marking $M = (M^1, M^2)$ is *clean* if the following conditions hold:

- $(\forall p \in {}^\bullet N \quad M^1(p) + \overline{M}^2(p) \neq 0 \implies (\forall p \in P \quad \lambda(p) \neq \mathtt{e} \Rightarrow M^1(p) + \overline{M}^2(p) = 0)$
- $(\forall p \in N^\bullet \quad M^1(p) \neq 0 \,) \implies N^\bullet = M^1$

In order to define the firing of transitions we need to know their real durations. So, we consider *timed transitions*, which are pairs of the form (v, d') where v is a transition and $d' \in \mathbb{N}^+$. Then we say that a multiset of timed transitions RT is *enabled* at a marking M if the following conditions are satisfied:

- $\forall (v, d') \in RT \quad d' \geq \delta(v)$
- $\forall p \in P \quad M^1(p) \geq \sum\limits_{v \in T} RT(v) \cdot W(p, v)$ where $RT(v) = \sum\limits_{d' \in \mathbb{N}^+} RT(v, d')$.

Once we know when a multiset of timed transitions RT is enabled at a marking, the following firing rule defines the effect of its firing:

Definition 5 (Firing rule). *Let $N = (P, T, F, W, \delta, \lambda)$ be a TPN, and $M = (M^1, M^2)$ be a marking of N at some instant $\beta \in I\!N$. If a multiset of timed transitions RT is enabled at M and its transitions are fired, then the marking $M' = (M'^1, M'^2)$ reached at the instant $\beta + 1$ is defined as follows:*

- $M'^1 = M^1 - \sum\limits_{v \in C_0} RT(v)W(-, v) + \sum\limits_{v \in C_1} RT(v)W(v, -) + \sum\limits_{(v,1) \in C_2} M^2(v, 1)W(v, -)$ *where* $C_0 = \{\, v \in T \mid \exists d' \in I\!N^+, RT(v, d') > 0 \,\}$, $C_1 = \{\, v \in T \mid (v, 1) \in RT \,\}$, *and* $C_2 = \{\, (v, 1) \in T \times I\!N^+ \mid M^2(t, 1) > 0 \,\}$.
- $M'^2 : T \times I\!N^+ \longrightarrow I\!N$ *with* $$M'^2(v, \beta') = \begin{cases} RT(v, d') & \mathit{if}\ (v, d') \in RT \ \wedge\ \beta' = d' - 1 \\ M^2(v, \beta' + 1) & \mathit{otherwise} \end{cases}$$

The step generated by the firing of a set of transitions RT is denoted by $M[RT\rangle M'$. Step sequences are defined as usual, and the set of reachable markings in N from M is denoted by $Reach(N, M)$.

So, frozen tokens are those consumed by a transition in execution. Whenever that execution finishes they become available tokens in the postconditions of the fired transitions.

4.1 A Domain of Timed Boxes

Timed Petri boxes are equivalence classes of labelled timed Petri nets. A suitable equivalence relation should allow, at least, the derivation of identities such as those induced by associativity and commutativity of several operators, such as parallel composition or choice. Besides, this relation must allow us to abstract away the names of places and transitions; also it must provide a mechanism to identify duplicate elements in the nets. The relation that we propose is a natural extension of the one used in plain PBC, by adequately considering the temporal aspects of the nets, which means to preserve the duration of related transitions.

Definition 6 (Structural equivalence). *Being $N_1 = (P_1, T_1, F_1, W_1, \lambda_1)$ and $N_2 = (P_2, T_2, F_2, W_2, \lambda_2)$ two operator nets, they are said to be* ***structurally equivalent*** *(or just equivalent) iff there is a relation $\varphi \subseteq (P_1 \cup T_1) \times (P_2 \cup T_2)$ such that:*

1. $\varphi(P_1) = P_2$ *and* $\varphi^{-1}(P_2) = P_1$,
2. $\varphi(T_1) = T_2$ *and* $\varphi^{-1}(T_2) = T_1$,
3. $\forall (p_1, p_2), (v_1, v_2) \in \varphi \quad W_1(p_1, v_1) = W_2(p_2, v_2)$, $W_1(v_1, p_1) = W_2(v_2, p_2)$,
4. *If* $(x_1, x_2) \in \varphi$ *then* $\lambda_1(x_1) = \lambda_2(x_2)$,
5. $\forall v_1 \in T_1, v_2 \in T_2 \quad |\varphi(v_1)| = 1$ *and* $|\varphi^{-1}(v_2)| = 1$.

Being $N_1 = (P_1, T_1, F_1, W_1, \delta_1, \lambda_1, M_1)$ and $N_2 = (P_2, T_2, F_2, W_2, \delta_2, \lambda_2, M_2)$ two marked timed plain nets, they are said to be ***structurally equivalent*** *(or just equivalent) iff there is a binary relation $\varphi \subseteq (P_1 \cup T_1) \times (P_2 \cup T_2)$ such that:*

1-4. *As before,*
5. *If* $(v_1, v_2) \in \varphi$ *then* $\delta_1(v_1) = \delta_2(v_2)$,
6. *If* $(p_1, p_2) \in \varphi$ *then* $M_1^1(p_1) = M_2^1(p_2)$,
7. *If* $(v_1, v_2) \in \varphi$ *then* $\forall d \in \mathbb{N}^+ \quad M_1^2(v_1, d) = M_2^2(v_2, d)$.

There are several conditions that a net N has to satisfy in order to generate a timed box. First, we impose *T-restrictedness* which means that the pre-set and post-set of each transition are non-empty sets. Besides, we impose *ex-restrictedness* (there is at least one entry place and one exit place) and *ex-directedness* (pre-sets of entry places and post-sets of exit places are empty). All these assumptions are inherited from the untimed version of the calculus.

Remark 7. Next we will only consider nets N satisfying the following conditions:

- N is T-restricted: $\forall v \in T \quad {}^\bullet v \neq \varnothing \neq v^\bullet$,
- There are no side conditions : $\forall v \in T \quad {}^\bullet v \cap v^\bullet = \varnothing$,
- N has at least one entry place: ${}^\bullet N \neq \varnothing$,
- N has at least one exit place: $N^\bullet \neq \varnothing$,
- There are no incoming arcs to entry places and no outgoing arcs from exit places: ${}^\bullet({}^\bullet N) = \varnothing \ \wedge \ (N^\bullet)^\bullet = \varnothing$,
- N is simple: $\forall p \in P \ \forall v \in T \quad W(p, v), W(v, p) \in \{0, 1\}$.

Definition 8 (Plain and operator timed boxes).

- *A marked* ***timed plain box*** *B is an equivalence class $B = [N]$ induced by the structural equivalence over labelled nets, where N is a marked plain net.*
- *An* ***operator box*** *Ω is an equivalence class $\Omega = [N]$ induced by the structural equivalence over labelled nets, where N is an operator net.*

Plain timed boxes will be the semantic objects to be associated with syntactic expressions, that is, the denotational semantics of an expression will be always a plain box. Its structural construction relies on operator boxes. Each semantic operator has a certain number of arguments (the same as the corresponding syntactic operator has). By applying them to a tuple of arguments we can obtain a new plain box, using the refinement procedure which we will explain later.

Next we define *static* and *dynamic* boxes. A plain box B is *static* if the marking of its canonical representative is empty ($M^1 = \varnothing \ \wedge \ M^2 = \varnothing$), and the reachable markings from the initial one $({}^\bullet B, \varnothing)$ are safe and clean. A plain box $B = [(P, T, F, W, \lambda)]$ is *dynamic* if the following conditions are satisfied:

- The marking of its canonical representative is non-empty,
- The plain box $[(P, T, F, W, \delta, \lambda, (\varnothing, \varnothing))]$ is static,
- The reachable markings from M are safe and clean.

The set of static boxes is Box^s, and the set of dynamic boxes is Box^d.

Plain boxes are classified in several classes depending on the type of tokens that they contain, and the labels of the places they are in.

Definition 9 (Classes of plain boxes). *Let $B = [(P, T, F, W, \delta, \lambda, (M^1, M^2))]$ be a plain box. We say that B is a* ***stable box*** *if $M^2 = \varnothing$; otherwise we say that B is* ***unstable****. If B is a stable box, then we say that it is an* ***entry-box*** *if $M^1 = {}^\bullet B$; an* ***exit-box*** *if $M^1 = B^\bullet$; and an* ***intermediate-box****, otherwise. All these classes are denoted by Box^{st}, Box^{ust}, Box^{e}, Box^{x}, Box^{i}, respectively.*

For operator boxes we need the additional property of being *factorisable*. In order to extend this notion to the timed case we first present the concept of (reachable) marking of an operator box. In this case it is enough to know how available and frozen tokens are distributed. By means of them we define which arguments of the operator are stable and which ones are in execution.

Definition 10 (Markings of operator boxes). *Being $\Omega = [(P, T, F, W, \lambda)]$ an operator box, a* ***marking*** *M of Ω is a pair $(M^1, M^2) \in \mathcal{M}(P) \times \mathcal{M}(P)$.*

In the definition above M^1 represents the multiset of available tokens, while M^2 defines the set of places where we have frozen tokens.

Definition 11 (Reachable markings of operator boxes). *Let $\Omega = [(P, T, F, W, \lambda)]$ be an operator box. We say that a multiset of transition $RT \in \mathcal{M}(T)$ is* ***enabled*** *at a marking M if the following condition is satisfied:*

$$\forall p \in P \quad M^1(p) \geq \sum_{v \in T} RT(v) \cdot W(p, v)$$

The set of ***reachable markings*** *of Ω after the firing of RT is defined as the set of markings (M'^1, M'^2) such that:*

- $M'^1 = M^1 - \sum_{v \in T} RT(v) \cdot W(-, v) + \sum_{v \in C} W(v, -)$
- $M'^2 = M^2 + RT - C$ *where* $C \subseteq M^2 \cup RT$.

Available tokens indicate the arguments of the connective which are in stable form (that is, they have no executing transition), while frozen tokens say us the ones that are currently in execution. The condition of factorisability tries to capture these distributions of tokens. Basically, it means that when a postcondition (precondition) of a transition is marked, all its postconditions (preconditions) must be also marked with tokens of the same type. To define the condition three sets of transitions are considered, one for frozen tokens and two for available tokens. This distinction is necessary because frozen tokens are always placed in the preconditions of the transitions in execution, while available tokens can either be consumed by the firing of a transition or obtained as a consequence of the execution of another transition.

Definition 12 (Factorisability). *Let $\Omega = [(P, T, F, W, \lambda)]$ be an operator box and $M = (M^1, M^2)$ be a marking of Ω. A* ***factorisation*** *of M is a triple of sets of transitions $\Phi = (\Phi_1, \Phi_2, \Phi_3)$ such that the following conditions hold:*

$$\bullet\ M^1 = \left(\biguplus_{v \in \Phi_1} {}^\bullet v \right) \uplus \left(\biguplus_{v \in \Phi_3} v^\bullet \right) \bullet\ M^2 = \left(\biguplus_{v \in \Phi_2} {}^\bullet v \right)$$

We say that Ω is ***factorisable*** *iff every safe marking $M \in Reach(\Omega, ({}^\bullet\Omega, \varnothing))$ satisfies that for every set U of transitions enabled at M, there is a factorisation*

$\Phi = (\Phi_1, \Phi_2, \Phi_3)$ of M such that $U \subseteq \Phi_1$. In the following, the set of factorisations of Ω is denoted by $fact_\Omega$, and we will use $\tilde{\Phi}$ to denote $\Phi_1 \cup \Phi_2 \cup \Phi_3$.

If factorisability were violated, the marking of an operator box could not be distributed over its arguments. Indeed, when factorisability is violated there must be a token (available or frozen) which neither can have been produced by the firing of any transition, nor enables by itself any marking; otherwise all the preconditions or postconditions of the involved transition would be marked.

Now we can finally define operator boxes:

Definition 13 (Acceptable operator boxes). *Let (P, T, F, W, λ) be an operator net satisfying the requirements in Remark 7. The equivalence class $\Omega = [(P, T, F, W, \lambda)]$ is an **acceptable operator box** if it is factorisable, and all the markings reachable from $({}^\bullet\Omega, \varnothing)$ are safe and clean.*

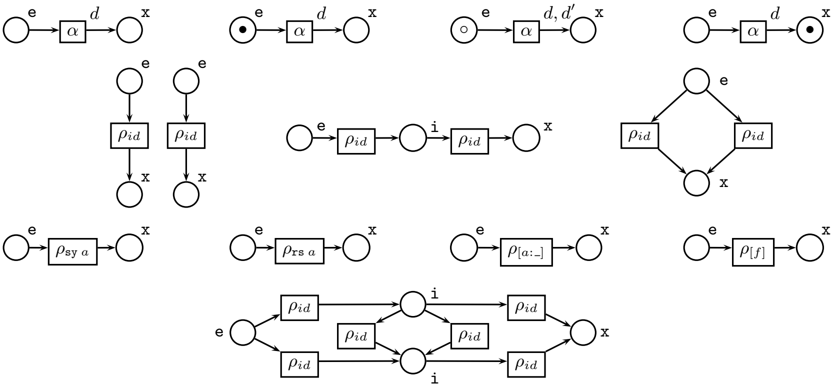

Fig. 1. Denotational semantics of TPBC

The denotational semantics of the algebra is defined in Figure 1. From left to right and top to bottom, we show the semantics of: basic actions ($\alpha : d$, $\overline{\alpha : d}$, $\widetilde{\alpha : d, d'}$ and $\underline{\alpha : d}$); disjoint parallelism ($\_\|\_$), sequential composition ($\_;\_$), and choice ($\_\square\_$); synchronization ($\_\mathtt{sy}\ a$), restriction ($\_\mathtt{rs}\ a$), hiding ($[a : \_]$), basic relabelling ($\_[f]$) and iteration ($[\_ * \_ * \_]$). In this figure, frozen tokens only appear in the semantics of $\widetilde{\alpha : d, d'}$, where the corresponding marking is given by $M^1 = \varnothing$ and $M^2 = \{(\alpha, d')\}$.

4.2 Refinement

Refinement is the mechanism to obtain a plain box $op_\Omega(B_1, B_2, \ldots, B_n)$ from a given operator box Ω and a tuple of plain boxes $(B_1, B_2, \ldots, B_n)$. The basic idea is that every transition of the operator box is replaced by one element of the tuple $(B_1, B_2, \ldots, B_n)$, where we have previously done the changes indicated by the label of the transition.

In the static case, when the involved boxes are unmarked, the mechanism followed is the same as in plain PBC. The same happens when frozen tokens are not involved. We will just comment how it works in the remaining case.

When frozen tokens appear the refinement basically involves the same structural changes as those in plain PBC [1,2,3]. However this procedure has been slightly modified with two purposes. First, we avoid the existence of arcs with a weight greater than one, which in fact were useless, since they only increase the size of the obtained box, without allowing new firings. The second and more important aim is the inclusion of time. The underlying intuition is that any transition in execution in a box remains the same when this box is "mixed" with some others, unless it proceeds from a synchronization, in which case the synchronized transitions can be considered in execution.

The following definition of the refinement domain reflects the fact that each operator box is a partial operation from plain boxes to plain boxes.

Definition 14 (Refinement domain). *Let $\Omega = (P, T, F, W, \lambda)$ be an operator box with n arguments. The* ***domain*** *of application of Ω, denoted by dom_Ω, is defined by the following conditions:*

1. *It comprises all the tuples $(B_{v_1}, B_{v_2}, \ldots, B_{v_n})$ of static plain boxes,*
2. *For every factorisation of Ω, $\Phi = (\Phi_1, \Phi_2, \Phi_3) \in fact_\Omega$, it comprises all tuples of boxes $\mathbb{B} = (B_{v_1}, B_{v_2}, \ldots, B_{v_n})$ such that:*
   - $\forall v \in \Phi_1 \quad B_v \in Box^e \cup Box^i$,
   - $\forall v \in \Phi_2 \quad B_v \in Box^{ust}$,
   - $\forall v \in \Phi_3 \quad B_v \in Box^x$,
   - $\forall v \in T \backslash \tilde{\Phi} \quad B_v \in Box^s$.

Φ_1 indicates which transitions will be replaced by boxes that contribute with stable non-final markings; Φ_2 represents the transitions corresponding to unstable boxes; Φ_3 denotes the transitions instantiated by boxes with terminal markings, and $T \backslash \tilde{\Phi}$ are the remaining transitions, that is, those which contribute with empty markings to the generated plain box.

In order to simplify the formal definition of refinement, we assume that in operator boxes all place and transition names of their canonical representatives are primitive names from P_{op} and T_{op}, respectively. We shall further assume that in basic plain boxes, that is, in the denotational semantics of basic actions, all places and transitions have primitive names from, respectively, P_{box} and T_{box}. As we will see, when we apply this mechanism, we have as names of the newly constructed places and transitions labelled trees in two sets which are denoted by P_{tree} and T_{tree}, respectively. These trees are defined in a recurrent way and then each subtree of such a tree is also in the corresponding set.

Next we define a collection of auxiliary concepts which will be later used in order to adequately define the frozen tokens of any net obtained by refinement.

Definition 15. *If $q = l(q_1, \ldots, q_n) \in T_{tree}$, where l is the label of its root and $q_1, \ldots, q_n$ are the root's sons, we say that each q_i is a* ***component*** *of q. In the following, we will denote the set of components of q, $\{q_1, \ldots, q_n\}$, by $comp(q)$.*

Definition 16. *Given $T_1, T_2 \subseteq T_{tree}$, we say that $dec : T_1 \longrightarrow T_2 \times I\!N$ is a* ***decomposition*** *of T_1 iff the following condition holds:*

$$\forall q \in T_1 \quad dec(q) = \langle l(q_1, \ldots, q_n), k \rangle \text{ with } q_k = q$$

dec is a ***consistent decomposition*** *iff the following condition is satisfied:*

$$\forall q' = l(q_1,\ldots,q_n) \in T_2 \quad (\exists k \in I\!N\; \exists q \in T_1, dec(q) = \langle q', k\rangle) \implies \\ \implies (\forall k \in 1..n\; \exists q \in T_1, dec(q) = \langle q', k\rangle)$$

Finally, if $B = [(P,T,F,W,\delta,\lambda,(M^1,M^2)]$ is a timed plain box, $T_1 \subseteq T$, and $dec : T_1 \longrightarrow T_2 \times I\!N$ is a decomposition of T_1, we say that dec is a ***timed consistent decomposition*** *iff the following condition holds:*

$$\forall q' = l(q_1,\ldots,q_n) \in T_2 \quad (\exists k \in I\!N\; \exists q \in T_1, dec(q) = \langle q', k\rangle) \implies \\ \implies (\exists d \in I\!N^+\; \forall k \in 1..n\; \exists q \in T_1, (q,d) \in M^2 \;\wedge\; dec(q) = \langle q', k\rangle)$$

The formal definition of refinement is the following:

Definition 17 (Refinement). *Let $\Omega = [(P,T,F,W,\lambda)]$ be an operator box, and for each $v \in T$ $B_v = [(P_v,T_v,F_v,W_v,\delta_v,\lambda_v,(M_v^1,M_v^2))]$ be a timed plain box. Under the assumptions above on Ω and $\mathbb{B} = (B_{v_1},B_{v_2},\ldots,B_{v_n})$, the result of the simultaneous substitution of the nets B_{v_i} for the transitions in Ω is any plain timed box whose canonical representative is a timed plain net*

$$op_\Omega(\mathbb{B}) = (P_0,T_0,F_0,W_0,\delta_0,\lambda_0,(M_0^1,M_0^2))$$

defined as follows:

1. ***Places, their labels and markings:*** *The set of places, P_0, is given by*

$$P_0 = \left(\bigcup_{v\in T} IP^v_{new}\right) \cup \left(\bigcup_{p\in P} OP^p_{new}\right)$$

where the sets IP^v_{new} and OP^p_{new} are defined as follows:

- *For each $v \in T$, IP^v_{new} is the set of places $\{v.p_v \,|\, p_v$ internal place in $P_v\}$. The label of all these places is* i *and their marking is given by $M_v^1(p_v)$.*
- *Let $p \in P$ with ${}^\bullet p = \{v_1,v_2,\ldots,v_k\}$ and $p^\bullet = \{v_{k+1},v_{k+2},\ldots,v_{k+m}\}$. Then, OP^p_{new} is the set of places $p(v_1 \lhd p_1,\ldots,v_{k+m} \lhd p_{k+m})$ where*

$$\begin{cases} p_i \in (P_{v_i})^\bullet \;\; \forall i \in \{1,\ldots,k\} \\ p_i \in {}^\bullet(P_{v_i}) \; \forall i \in \{k+1,\ldots,k+m\}. \end{cases}$$

Each one of these places will be labelled by $\lambda(p)$ and its marking is

$$M^1_{v_1}(p_1) + M^1_{v_2}(p_2) + \ldots + M^1_{v_{k+m}}(p_{k+m}).$$

2. ***Transitions, their labels and durations:*** *The set of transitions T_0 is defined by*

$$T_0 = \bigcup_{v\in T} T^v_{new}$$

where for each $v \in T$ the set T^v_{new} is obtained as follows: Whenever we have a pair of the form $(\{\lambda(q_1) : d_1, \lambda(q_2) : d_2, \ldots, \lambda(q_n) : d_n\}, \alpha : d) \in \lambda(v)$ for $q_i \in T_v$, and the following condition holds:

$$\forall i,j \in \{1,\ldots,n\}\; i \neq j \implies ({}^\bullet q_i \cap {}^\bullet q_j = \varnothing) \wedge (q_i^\bullet \cap q_j^\bullet = \varnothing),$$

a new transition $v.\alpha(q_1,\ldots,q_n)$ is generated. Its duration is d and its label is α. In the following, we will denote the transition v by $root(v_0)$.

3. ***Set of arcs:*** *For each transition v_0 in T_0, the set of arcs leaving or reaching v_0 in F_0 are those obtained as follows:*
   - $p_0 \in IP^v_{new}$
     $$\begin{cases} \text{if } \exists q_i \in comp(v_0),\ (p_v, q_i) \in F_v \ then\ (p_0, v_0) \in F_0 \\ \text{if } \exists q_i \in comp(v_0),\ W_v(q_i, p_v) = 1 \ then\ (v_0, p_0) \in F_0 \end{cases}$$
   - $p_0 \in OP^p_{new}$
     $$\begin{cases} \text{if } \exists i \in \{1, \ldots, k+m\} \ \wedge\ \exists q_j \in comp(v_0),\ (p_i, q_j) \in F_v \ then\ (p_0, v_0) \in F_0 \\ \text{if } \exists i \in \{1, \ldots, k+m\} \ \wedge\ \exists q_j \in comp(v_0),\ (q_j, p_i) \in F_v \ then\ (v_0, p_0) \in F_0 \end{cases}$$

   The weight function returns 1 for all the arcs in F_0.
4. ***Transitions in execution (frozen tokens):*** *The frozen marking M_0^2 will be defined as the union of a collection of submarkings $M_{0,v}^2$ with $v \in T$. As a matter of fact, each M_v^2 will be not defined in a unique way and therefore we will have several possible frozen markings M_0^2. This is justified for technical reasons, since once we prove the equivalence between the operational and the denotational semantics, we obtain as a consequence that all the different boxes defining the denotational semantics of an expression are in fact equivalent. Thus, we could also select any of the possibilities to obtain a function instead of a relation, but if we would do it in this way the definition would be much less readable.*

   Then, in order to get a value for each $M_{0,v}^2$ we consider any timed consistent decomposition $dec : M_v^2 \longrightarrow T_0 \times I\!N$, and we take as $M_{0,v}^2$
   $$M_{0,v}^2 = \{\, q' \in T_0 \mid \exists q \in M_v^2, dec(q) = q' \,\}$$
   where for each $q' \in M_{0,v}^2$ we take $rem(q') = rem(q)$.

Theorem 18.

1. *Every step sequence of the operational semantics of G is also a step sequence of any of the timed plain boxes corresponding to G.*
2. *Every step sequence of any of the timed plain boxes corresponding to G is also a step sequence of the operational semantics of G.*

Proof. It follows the lines of that for plain PBC [1,2,3] with the changes needed to cope with time information (see [5]). Unfortunately, even the original proof is so long and involved that even it is not possible to sketch it here. It would be nice to comment at least the necessary changes to extend the proof, but this is neither possible since they are both local and distributed all along the proof. Therefore, we will only justify here why we need nondeterminism in the definition of the denotational semantics. This is needed in order to preserve a one to one relation between the evolution of the expressions and that of their denotational semantics, since there are some expressions which evolve to the same one after the execution of different multibags, but the same is not true for the evolution of a single box.

5 Example: A Producer/Consumer System

In this section we present an example that models the producer/consumer system with a buffer of capacity 1. The system describes the behaviour of a simple production line which involves two workers. A conveyor belt is between them, so it can hold any item that has been produced by the first worker and has to be consumed by the second.

The given specification consists on three components that are combined to obtain the TPBC expression of the system: the $\mathsf{producer}$ system, the $\mathsf{consumer}$ system and the $\mathsf{conveyorBelt}$ system. The timing assigned to them reflects a set of hypotheses about the production and consumption speed and the characteristics of the conveyor belt.

- $\mathsf{producer} = [(\{\mathsf{a}, \mathsf{b}\} : 1) * ((\mathsf{p} : 2); (\mathsf{p_b} : 1)) * (\{\neg\mathsf{a}, \neg\mathsf{b}\} : 1)]$
- $\mathsf{consumer} = [(\{\hat{\mathsf{a}}, \mathsf{b}\} : 1) * ((\mathsf{g_b} : 1); (\mathsf{c} : 1)) * (\{\widehat{\neg\mathsf{a}}, \neg\mathsf{b}\} : 1)]$
- $\mathsf{conveyorBelt} = [\mathsf{G_1} *\ [(\widehat{\mathsf{p_b}} : 1) * ((\widehat{\mathsf{g_b}} : 1; \widehat{\mathsf{g_b}} : 1)\square(\widehat{\mathsf{g_b}} : 1; \widehat{\mathsf{p_b}} : 1)) * (\widehat{\mathsf{g_b}} : 1)] *\ \mathsf{G_2}]$
  where $\mathsf{G_1} = (\{\mathsf{b}, \mathsf{b}\} : 1)$ and $\mathsf{G_2} = (\{\widehat{\neg\mathsf{b}}, \widehat{\neg\mathsf{b}}\} : 1)$
- $\mathsf{system} = (((\mathsf{producer}\|\mathsf{consumer})\mathtt{sy}\ \mathsf{A}\|\mathsf{conveyorBelt})\mathtt{sy}\ \mathsf{B})\mathtt{rs}(\mathsf{A} \cup \mathsf{B})$
  where $\mathsf{A} = \{\mathsf{a}, \neg\mathsf{a}\} \quad \mathsf{B} = \{\mathsf{b}, \neg\mathsf{b}, \mathsf{p_b}, \mathsf{g_b}\}$

The meaning of the actions is the following:

p: Produce an item.
c: Consume an item.
$\mathsf{p_b}$: Put an item into the conveyor belt.
$\mathsf{g_b}$: Get an item from the conveyor belt.
$\mathsf{b}, \neg\mathsf{b}$: Activate and deactivate the running of the conveyor belt.
$\mathsf{a}, \neg\mathsf{a}$: Activate and deactivate the production/consumption of items.

Figure 2 shows the denotational semantics of $\mathsf{producer}$ and $\mathsf{consumer}$ expressions. The construction of the boxes corresponding to $\mathsf{conveyorBelt}$ and the full system expressions can be completed in the same way, but due to lack of space we can not show the obtained nets. You can check by hand the equivalence between the operational semantics of the given expressions and those of the obtained nets.

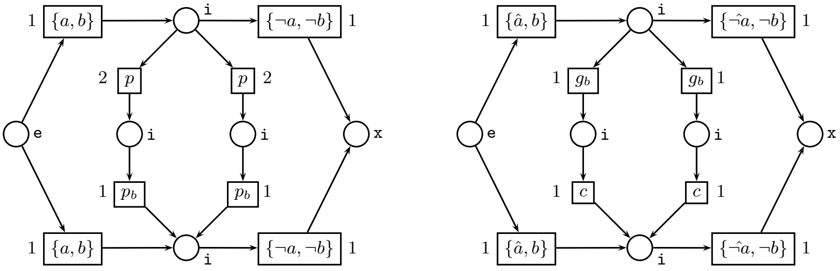

Fig. 2. The producer/consumer problem

6 The Time-Constraining Operators

The previous sections describe a simple timed extension of the basic model where we need not to introduce any new operator. Note that due to rules **(V1)** and

(V2), processes can let time progress indefinitely. More precisely, for any overlined or underlined process time passes without causing any change of the state, out of the fact that processes become stable after finishing their pending actions.

This property, called *unlimited waiting*, is not too useful from a practical point of view. We need to introduce some kind of urgency in the formalism; otherwise, the expressive power of TPBC would be limited to specify very simple systems. Therefore, in this section we will add to our model some new features that force systems to evolve, namely *timed choice* and *time-out with exception handler*.

6.1 Timed Choice

Timed choice constraints an action to occur at some instant of a given interval $[t_0, t_1]$. That is, a so restricted action cannot begin its execution either before t_0 or after t_1. The static expression used to implement this behaviour is $\alpha^{[t_0,t_1]} : d$, where $\alpha \in \mathcal{M}(\mathcal{A})$, $t_0 \in \mathbb{N}$, $d \in \mathbb{N}^+$, and $t_1 \in \mathbb{N}^+ \cup \{\infty\}$.

Depending on the values of t_0 and t_1 there are three particular cases of timed choice. A rough description of each one is as follows:

1. **Finite delay:** It corresponds to expressions of the form $\alpha^{[t_0,\infty]} : d$, where $t_0 \neq 0$ (note that processes of the form $\alpha^{[0,\infty]} : d$ will be just equivalent to $\alpha : d$.) When such an expression gets the control of the system, it delays the execution of the action $\alpha : d$ for at least t_0 time units.
2. **Time-out:** It is obtained when $t_0 = 0$ and $t_1 \neq \infty$. If a process of the form $\alpha^{[0,t_1]} : d$ is activated, the beginning of the execution of $\alpha : d$ cannot be postponed more than t_1 units of time.
3. **Time-stamped actions:** We say that an action is *time-stamped* when its execution is enforced to start at a given instant. The corresponding static expressions are $\alpha^{[t_0,t_0]} : d$ with $t_0 \neq \infty$, which we will usually syntactic sugar by $\alpha^{t_0} : d$.

Now, activated timed choices need to have information about the passage of time over the system components. More exactly, it is necessary to know for how long a timed choice has been activated, in order to restrict the beginning of the execution of the corresponding action. Until now, dynamic expressions only give us information about the global state of processes, in such a way that an external observer is able to determine whether a process has pending actions or not, but there is no way to obtain quantitative information about time aspects, unless the observer knows the full history of the process.

With the purpose of avoiding this limitation, we introduce a temporal annotation over both overlined and underlined expressions. In the first case, the new clock tells us for how long the process has been activated, while in the second case it tells us how long ago the process finished. This annotation will be included as an index close to the corresponding bars.

Due to these modifications, the operational semantics previously defined must be changed. In particular, any clock of the system needs to be updated when time progresses. This is done by means of new rules **(V1)** and **(V2)**:

$$\overline{E}^{k} \xrightarrow{\varnothing} \overline{E}^{k+1} \ \mathbf{(V1)} \qquad \underline{E}_{k} \xrightarrow{\varnothing} \underline{E}_{k+1} \ \mathbf{(V2)}$$

Table 1. New transition rules with time information

$\overline{\alpha : d}^{k} \xrightarrow{\{\alpha_{d'}\}} \alpha : \widetilde{d, d'} - 1 \;\; d' \geq d$	**(B1)**	$\widetilde{\alpha : d}, 0 \longleftrightarrow \underline{\alpha : d}_{0}$	**(B3)**
$\overline{E \parallel F}^{k} \longleftrightarrow \overline{E}^{k} \parallel \overline{F}^{k}$	**(Pc1)**	$\underline{E}_{k} \parallel \underline{F}_{k'} \longleftrightarrow \underline{E \parallel F}_{min\{k,k'\}}$	**(Pc3)**
$\overline{E; F}^{k} \longleftrightarrow \overline{E}^{k}; F$	**(Sc1)**	$\underline{E}_{k}; F \longleftrightarrow E; \overline{F}^{k}$	**(Sc3)**
$E; \underline{F}_{k} \longleftrightarrow \underline{E; F}_{k}$	**(Sc5)**		
$\overline{E \Box F}^{k} \longleftrightarrow \overline{E}^{k} \Box F$	**(Ch1a)**	$\overline{E \Box F}^{k} \longleftrightarrow E \Box \overline{F}^{k}$	**(Ch1b)**
$\underline{E}_{k} \Box F \longleftrightarrow \underline{E \Box F}_{k}$	**(Ch2a)**	$E \Box \underline{F}_{k} \longleftrightarrow \underline{E \Box F}_{k}$	**(Ch2b)**
$\overline{E \ \mathtt{sy}\ a}^{k} \longleftrightarrow \overline{E}^{k} \ \mathtt{sy}\ a$	**(S1)**	$\underline{E}_{k} \ \mathtt{sy}\ a \longleftrightarrow \underline{E \ \mathtt{sy}\ a}_{k}$	**(S3)**
$\overline{E \ \mathtt{rs}\ a}^{k} \longleftrightarrow \overline{E}^{k} \ \mathtt{rs}\ a$	**(Rs1)**	$\underline{E}_{k} \ \mathtt{rs}\ a \longleftrightarrow \underline{E \ \mathtt{rs}\ a}_{k}$	**(Rs3)**
$\overline{[a : E]}^{k} \longleftrightarrow [a : \overline{E}^{k}]$	**(Sp1)**	$[a : \underline{E}_{k}] \longleftrightarrow \underline{[a : E]}_{k}$	**(Sp3)**
$\overline{E[f]}^{k} \longleftrightarrow \overline{E}^{k}[f]$	**(Rl1)**	$\underline{E}_{k}[f] \longleftrightarrow \underline{E[f]}_{k}$	**(Rl3)**
$\overline{[E * F * E']}^{k} \longleftrightarrow [\overline{E}^{k} * F * E']$	**(It1)**		
$[\underline{E}_{k} * F * E'] \longleftrightarrow [E * \overline{F}^{k} * E']$	**(It2c)**	$[\underline{E}_{k} * F * E'] \longleftrightarrow [E * F * \overline{E'}^{k}]$	**(It2d)**
$[E * \underline{F}_{k} * E'] \longleftrightarrow [E * \overline{F}^{k} * E']$	**(It3c)**	$[E * \underline{F}_{k} * E'] \longleftrightarrow [E * F * \overline{E'}^{k}]$	**(It3d)**
$[E * F * \underline{E'}_{k}] \longleftrightarrow \underline{[E * F * E']}_{k}$	**(It5)**		

In addition to this, we have to modify most of the rules concerned with control transmission. The resulting rules are those shown in Table 1, and the reasoning supporting them is straightforward. Next the definition of the operational semantics of timed choice, which is shown in Table 2. Rule **(TCh1)** states that such a process can perform the empty timed bag at any time. In order to restrict the execution of the action $\alpha : d$, the value k of the system clock is compared with the limits of the interval $[t_0, t_1]$. If $t_0 \leq k \leq t_1$ we can apply rule **(TCh2)**, which establishes that the corresponding (activated) timed choice, $\overline{\alpha^{[t_0,t_1]} : d}^{k}$, can perform the timed bag $\{\alpha_{d'}\}$. Afterwards, rules **(TCh3)** and **(TCh4)** define the expected behaviour: The remaining execution of $\{\alpha_{d'}\}$ is hidden to the observer, who only sees the passage of time (rule **(TCh3)**). When the action terminates, the process is underlined and the value of its clock is reset to zero (rule **(TCh4)**).

6.2 Time-Out with Exception Handler

By means of the timed choice operator just introduced, we are able to limit the time at which a process will be able to start its execution. If this time is exceeded, the process dies, and there is no way to express that some alternative continuation will be activated. In order to get this, we introduce a new operator called *time-out with exception handler*. This operator has two arguments, $E, F \in Expr^{s}$, and a parameter $t_0 \in I\!N^{+}$. E is called the *body* and F the *exception handler* of the time-out.

Table 2. Operational semantics of *timed choice*

$\overline{\alpha^{[t_0,t_1]} : d}^{k} \xrightarrow{\varnothing} \overline{\alpha^{[t_0,t_1]} : d}^{k+1}$	**(TCh1)**
$\overline{\alpha^{[t_0,t_1]} : d}^{k} \xrightarrow{\{\alpha_{d'}\}} \widetilde{\alpha^{[t_0,t_1]} : d, d'} - 1$ if $k \geq t_0$, $k \leq t_1$ and $d' \geq d$	**(TCh2)**
$\widetilde{\alpha^{[t_0,t_1]} : d, d'} \xrightarrow{\varnothing} \widetilde{\alpha^{[t_0,t_1]} : d, d'} - 1$ if $d' > 0$	**(TCh3)**
$\widetilde{\alpha^{[t_0,t_1]} : d, 0} \longleftrightarrow \underline{\alpha^{[t_0,t_1]} : d}_0$	**(TCh4)**

When a time-out with exception handler gets the control, it behaves as its body whenever it begins to perform actions before the instant t_0; otherwise, after time t_0, the process behaves as defined by its exception handler.

The static form of a time-out with exception handler is $\lfloor E \rfloor^{t_0} F$. The possible dynamic forms are equivalent to either $\lfloor G \rfloor^{t_0;t} F$, with $t \geq -1$, or $\lfloor E \rfloor^{t_0;-1} G$. In both cases, the expressions include two temporal annotations. The former is the parameter of the time-out and it is immutable, that is, it does not change during the execution of the process. If t is the value of the latter, this means that the exception handler F will be activated within t time units, unless the body of the time-out begins its execution before. t is set to -1 when either the body of the time-out has begun to execute its first action before t_0 or that limit has been exceeded without performing any non-empty transition of E. The operational semantics reflecting these ideas is shown in Table 3.

Rules **(Te1a)** and **(Te1b)** state that we have two cases depending on the elapsed time from the activation of the process: If $k \leq t_0$ then the body gets the control of the system, and it has $t_0 - k$ time units to execute its first action (rule **(Te1a)**). Otherwise, the exception handler of the time-out is activated, just as rule **(Te1b)** states. Rule **(Te2a)**[1] shows how the passage of time affects this class of processes. Notice carefully that control is transmitted according to the second temporal annotation, so the first argument remains active as long as $t \geq 0$. Otherwise, the body has totally consumed its disposal time and the exception handler is activated. The performance of actions is formalized by means of rules **(Te2b)**, **(Te3b)**, and **(Te4b)**, whose explanation is straightforward. They can be applied on any equivalent form of the arguments (rules **(Te3a)** and **(Te4a)**). Finally, a time-out with exception handler finishes its execution when either its body or its exception handler terminates (rules **(Te5a)** and **(Te5b)**).

6.3 Remarks on Denotational Semantics

Denotational semantics of the time-constraining operators can be found in [5]. Plain nets have been modified by adding temporal restrictions $[t_0, t_1]$ to transitions, and labelling the available tokens in the net with their age. Then, a modified firing rule uses this information, in such a way that a transition v restricted by $[t_0, t_1]$ can only consume tokens younger than t_1.

[1] Predicate `Init`(G) (see [5]) returns `true` if and only if G is equivalent to some overlined expression H.

Table 3. Operational semantics of the *time-out with exception handler*

$\overline{\lfloor E \rfloor^{t_0} F}^{k} \longleftrightarrow \lfloor \overline{E}^{k} \rfloor^{t_0; t_0 - k} F$ if $k \leq t_0$	**(Te1a)**
$\overline{\lfloor E \rfloor^{t_0} F}^{k} \longleftrightarrow \lfloor E \rfloor^{t_0; -1} \overline{F}^{(k - t_0 - 1)}$ if $k > t_0$	**(Te1b)**
$\dfrac{G \xrightarrow{\varnothing} G' \quad \mathtt{Init}(G) \wedge t \geq 1}{\lfloor G \rfloor^{t_0; t} F \xrightarrow{\Gamma} \lfloor G' \rfloor^{t_0; t-1} F}$	**(Te2a)**
$\dfrac{G \xrightarrow{\Gamma} G' \quad \Gamma \neq \varnothing \quad \mathtt{Init}(G) \wedge t \geq 0}{\lfloor G \rfloor^{t_0; t} F \xrightarrow{\Gamma} \lfloor G' \rfloor^{t_0; -1} F}$	**(Te2b)**
$\dfrac{G \longleftrightarrow G'}{\lfloor G \rfloor^{t_0; t} F \longleftrightarrow \lfloor G' \rfloor^{t_0; t} F}$	**(Te3a)**
$\dfrac{G \xrightarrow{\Gamma} G' \quad \neg \mathtt{Init}(G)}{\lfloor G \rfloor^{t_0; -1} F \xrightarrow{\Gamma} \lfloor G' \rfloor^{t_0; -1} F}$	**(Te3b)**
$\dfrac{G \longleftrightarrow G'}{\lfloor E \rfloor^{t_0; -1} G \longleftrightarrow \lfloor E \rfloor^{t_0; -1} G'}$	**(Te4a)**
$\dfrac{G \xrightarrow{\Gamma} G'}{\lfloor E \rfloor^{t_0; -1} G \xrightarrow{\Gamma} \lfloor E \rfloor^{t_0; -1} G'}$	**(Te4b)**
$\lfloor \underline{E}_{k} \rfloor^{t_0; -1} F \longleftrightarrow \underline{\lfloor E \rfloor^{t_0} F}_{k}$	**(Te5a)**
$\lfloor E \rfloor^{t_0; -1} \underline{F}_{k} \longleftrightarrow \underline{\lfloor E \rfloor^{t_0} F}_{k}$	**(Te5b)**

As a consequence, we obtain that a transition generated by synchronization is enabled at a marking M if and only if all its component transitions would be enabled. This is also true even if any of these components have been removed by application of a restriction operator. This is probably the most important difference between our model and that in [4]. There the author had to introduce *illegal action occurrences* in order to reflect some (intuitively undesirable) synchronizations whenever they become executable in the box defining the denotational semantics of an expression. This happens when the application of a restriction operator removes some timing restrictions which made that the synchronization was non-executable before the application of the restriction operator.

Although in that paper the author does not explain why he decided to get the equivalence between both semantics by allowing the execution of *impossible* synchronizations, we assume that he wants to maintain that equivalence result without changing too much the ideas of untimed PBC. As a consequence, it is not possible to avoid the fireability of those *impossible* synchronizations. To do it, it is mandatory the introduction of age information in the tokens, as we have done in our model. So, we have to pay the price of a more complicated model, but as a reward we avoid those undesirable transitions.

7 Conclusions and Work in Progress

In this paper we have presented a new proposal to introduce time in PBC, preserving the main ideas of this model as far as possible. In the discussed model, called TPBC (*Timed PBC*), processes execute actions which have a duration.

The work by Koutny [4] has been somehow completed by considering an alternative model. More importantly, we have defined a more involved model in order to avoid the generation of undesired transitions. This cannot be done if we just work with plain time Petri nets since these are not prepared to support the correct definition of some time operators such as urgency.

As work in progress, currently we are introducing in the calculus new features enhancing its time characteristics, mainly *maximal parallelism* and *urgency*. Another line of research concerns not only TPBC but also the original PBC. In order to exploit these Petri box calculi, we are working on the axiomatization of the semantics, as it is done in the framework of process algebras. Once we will get the equations in this axiomatization, we will try to interpret them at the level of Petri nets, obtaining a collection of basic correct transformations. We are sure that this will be a fine way to exploit the facilities of both PBC and TPBC.

References

1. E. Best, R. Devillers and J. Hall *The Petri Box Calculus: A New Causal Algebra with Multi-label Communication.* Advances in Petri Nets 1992, LNCS vol.609, pp.21-69. Springer-Verlag, 1992.
2. E. Best and M. Koutny *A Refined View of the Box Algebra.* Petri Net Conference'95, LNCS vol.935, pp.1-20. Springer-Verlag, 1995.
3. M. Koutny and E. Best *Operational and Denotational Semantics for the Box Algebra.* Theoretical Computer Science 211, pp.1-83, 1999.
4. M. Koutny *A Compositional Model of Time Petri Nets.* Application and Theory of Petri Nets 2000, LNCS vol.1825, pp.303-322. Springer-Verlag, 2000.
5. O. Marroquín Alonso and D. Frutos Escrig. *TPBC: Timed Petri Box Calculus.* Technical Report, Dept. Sistemas Informáticos y Programación. UCM, 2000. In Spanish.
6. P. Merlin *A Study of the Recoverability of Communication Protocols.* PhD. Thesis, University of California, 1974.
7. F. Moller and C. Tofts *A Temporal Calculus of Communicating Systems.* CONCUR'90: Theories of Concurrency: Unification and Extension, LNCS vol.458, pp.401-415. Springer-Verlag, 1990.
8. Y. Ortega Mallén *En Busca del Tiempo Perdido.* PhD. Thesis, Universidad Complutense de Madrid. 1990.
9. C. Ramchandani *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets.* Technical Report 120. Project MAC. 1974.
10. G.M. Reed and A.W.Roscoe *Metric Spaces as Models for Real-time Concurrency.* Mathematical Foundations of Programming, LNCS vol.298, pp.331-343. Springer-Verlag, 1987.
11. Wang Yi *A Calculus of Real Time Systems.* PhD. Thesis, Chalmers University of Technology, 1991.