A Refined View of the Box Algebra

Eike Best[⋆] and Maciej Koutny[⋆⋆]

Abstract. This paper presents the operational semantics and the Petri net semantics of a fragment of the box algebra in tutorial style. For the operational semantics, inductive rules for marked expressions are given. For the net semantics, a general mechanism of refinement and relabelling is introduced, using which the connectives of the algebra are defined. A companion paper shows how this mechanism can be extended to handle recursion.

Keywords: Petri nets, Process algebra, Refinement.

1 Introduction

The box algebra [1, 2, 12] is a variant of CCS [13] whose main purpose is to support the compositional translation from high level concurrent programming languages into Petri nets. To achieve this purpose, some modifications with respect to CCS are provided, such as extending its synchronisation operation with a multisynchronisation capability[1] and separating concurrent composition from synchronisation. The present paper and its companion paper [5] explain the Petri net semantics of the box algebra using a uniform framework having generalised relabellings and simultaneous refinements as its basic net-forming constructors.

Before delving into its main subject, we discuss very briefly some motivation for the box algebra, as far as it is concerned with concurrent programming languages[2]. Consider, for example, the program fragment

$$\begin{array}{lll} \ldots & \mathbf{begin} & \mathbf{var}\, x : \{0,1\}; \\ & & \langle x := x \oplus 1 \rangle \parallel \langle x := y \rangle \\ & \mathbf{end} & \ldots \end{array}$$

where y is assumed to be declared with type $\{0,1\}$ in some outer block and $\oplus$ denotes addition modulo 2. The problem is to construct as small as possible a 1-safe place/transition net [16] describing this block. In addition, the construction must be compositional, that is, the net describing the block should be composed from similar nets for its three constituents, the declaration $\mathbf{var}\, x : \{0,1\}$ and the two atomic actions $\langle x := x \oplus 1 \rangle$ and $\langle x := y \rangle$. The box algebra solves this problem in the following way. Basic ingredients are allowed to be of the form x_{vw} which denotes the change of the value of x from v to w (where $v,w \in \{0,1\}$). Using these elementary ingredients, the following translation of the two atomic actions of the program is reasonable:

$$\langle x := x \oplus 1 \rangle \quad \rightsquigarrow \quad (\, \{x_{01}\} \,\Box\, \{x_{10}\}\,)$$

$$\langle x := y \rangle \quad \rightsquigarrow$$
$$(\, \{x_{00}, y_{00}\} \,\Box\, \{x_{10}, y_{00}\} \,\Box\, \{x_{01}, y_{11}\} \,\Box\, \{x_{11}, y_{11}\}\,)$$

where $\Box$ denotes alternative composition. The two-element sets in the last line mention both x and y, because the atomic action $\langle x := y \rangle$ involves both variables. Every set should be interpreted as denoting two simultaneously executable value changes. For instance, the set $\{x_{10}, y_{00}\}$ denotes the value

[⋆] Institut für Informatik, Universität Hildesheim, Marienburger Platz 22, D-31141 Hildesheim, E.Best@informatik.uni-hildesheim.de

[⋆⋆] Department of Computing Science, University of Newcastle upon Tyne, NE1 7RU, U.K., Maciej.Koutny@newcastle.ac.uk

[1] Not unlike that described in synchronous CCS [13], but retaining asynchronous concurrency.

[2] Readers wishing to know more about the translation of concurrent languages into the box algebra are referred to [4] for an elementary approach and to [3] for a high level approach.

of x being changed from 1 to 0 and the value of y being changed from 0 to 0, that is, remaining 0. For uniformity, the value changes corresponding to $\langle x := x \oplus 1\rangle$ are also expressed in the form of a set, even though this happens to be a singleton set because $\langle x := x \oplus 1\rangle$ mentions only one variable.

The expressions for $\langle x := x \oplus 1\rangle$ and $\langle x := y\rangle$ have corresponding Petri nets with respectively two and four alternative transitions labelled by the corresponding sets. The net corresponding to the declaration $\mathbf{var}\, x : \{0, 1\}$, on the other hand, involves transitions of the form $\{\widehat{x_{vw}}\}$ for all possible values ${}_{v,w} \in \{0, 1\}$ [4], where $\widehat{x_{vw}}$ is the *conjugate* of x_{vw}. The synchronisation operator of the box algebra allows a transition labelled $\{\widehat{x_{10}}\}$ coming from the declaration of x to be synchronised with a transition labelled $\{x_{10}, y_{00}\}$ coming from the action $\langle x := y\rangle$ selectively, that is, using and consuming the conjugate pair $(\widehat{x_{10}}, x_{10})$ but retaining y_{00}. Thus, the resulting transition has label $\{y_{00}\}$. This transition can be synchronised again with a transition labelled $\{\widehat{y_{00}}\}$ coming from the declaration of y in the outer block, using and consuming a conjugate pair $(\widehat{y_{00}}, y_{00})$ and yielding a transition labelled $\emptyset$. Such a transition corresponds to the CCS τ action.

For casting this idea in general terms, the box algebra assumes a ground set A of *action particles* on which an operation of conjugation is defined (in the above example, $x_{00}, x_{10}, \ldots, y_{00}, \ldots \in A$) and allows arbitrary finite multisets over A to comprise a basic action. The synchronisation operator of the box algebra has a parameter stating which particle is to be used for synchronisation. Σ **sy** a means that the transitions of the net Σ are to be synchronised using (only) pairs $(a, \hat{a})$. Thus, for instance, $\{a, b, \hat{c}\}$ and $\{\hat{a}, b, c\}$ are two transition labels that could lead to synchronisation using the pair $(a, \hat{a})$ or using the pair $(c, \hat{c})$. As another example, consider three transitions labelled $\{a, b\}$, $\{\hat{a}, \hat{a}\}$ and $\{a, c\}$. If these three transitions are synchronised over a then, amongst others, a 3-way synchronisation results from the two $(a, \hat{a})$ pairs that link the first and second, and the second and third, transitions, respectively. This 3-way transition has the label $\{b, c\}$ because both $(a, \hat{a})$ pairs are consumed in the synchronisation.

In the remainder of this paper we forget about programming languages and focus, instead, on the algebra and its translation into Petri nets. More precisely, we examine a fragment of the box algebra without repetitive constructs (iteration or recursion). In the first part of the paper we define the Petri net independent operational semantics of this subalgebra. In the second part we define its semantics in terms of special labelled nets called boxes. Technically, the box semantics is based on a general refinement and relabelling mechanism which has been inspired by Devillers' theory for simultaneous refinements [8]. This mechanism permits the uniform description of all connectives of the algebra. Namely, the flow of control (such as sequential composition and alternative composition) is driven by refinement, and connectives that relate to the addition or the deletion of activities (such as synchronisation and restriction) are described by relabelling.

The paper is structured as follows. In Section 2 a fragment of the box algebra is introduced and some examples are given. Section 3 specifies the operational semantics (adapted from [15]) for this algebra. In Section 4 the class of boxes, that is, labelled Petri nets that serve as the targets of the box algebra semantics, is defined. Section 5 explains the two general operations of refinement and relabelling, and develops a mechanism by which boxes can be compositionally defined. In Section 6 the Petri net semantics of the box algebra fragment is defined using this mechanism, and the main results of the theory are stated and explained.

2 A fragment of the box algebra

Let A be a countable alphabet of *action particles*. For synchronisation (as explained previously), we assume that A carries a bijection $\hat{}: A \to A$ called *conjugation* which satisfies

$$\forall a \in A\colon\ \hat{a} \neq a \wedge \hat{\hat{a}} = a.$$

We use the letters $a, b, a_i, \hat{a}, \ldots$ to denote elements of A. An action particle of A may be interpreted as *visible* or *observable* when it occurs. However, in contrast to CCS, a basic action of the box algebra may consist of more than one action particles. The basic actions of the box algebra, therefore, are finite multisets over A. We denote the set of basic actions by $\mathcal{L} = \mathcal{M}(A)$, where $\mathcal{M}(.)$ denotes the set of finite multisets over a set. The elements of $\mathcal{L}$ will be denoted by α, α_j etc. The letter $\mathcal{L}$ is chosen to remind of *label*.

For example, the action $\{\hat{a}, b\} \in \mathcal{L}$ should be interpreted as specifying the simultaneous execution of the two visible action particles $\hat{a}$ and b. Such an action can be synchronised in the CCS style either with an action containing an a particle, or with an action containing a $\hat{b}$ particle, or with both. The empty set $\emptyset$ is also a member of $\mathcal{L}$. It should be interpreted as specifying the execution of an action containing no visible action particles. Such an action is called unobservable, or also, in the terminology of CCS, *silent* or *internal*.

An *expression* of the finite box algebra, by definition, is a word of the syntax

$$\begin{aligned} E ::= {}& \alpha \mid E; E \mid E \square E \mid E \| E \mid \\ & E[f] \mid E \ \mathbf{sy}\ a \mid E \ \mathbf{rs}\ a. \end{aligned}$$

An expression of the form α, as we have said, is called a *basic action* (or multiaction). The connectives ; (sequence), $\square$ (choice) and $\|$ (disjoint parallelism) in the first line of the syntax are called the *control connectives*. The operators $[f]$ (basic relabelling), $\mathbf{sy}\ a$ (synchronisation) and $\mathbf{rs}\ a$ (restriction) in the second line of the syntax are called *(generalised) relabelling operators*. It is assumed that f is a function from A to A that preserves conjugates, that is, $\forall a \in A\colon f(\hat{a}) = \widehat{f(a)}$. We will use the following ten expressions as our running examples:

$$\begin{aligned}
E_0 &= \emptyset \\
E_1 &= \{\hat{a}, b\} \\
E_2 &= (\{a\}; \{b\}) \square \{c\} \\
E_3 &= \{a\}; (\{b\} \square \{c\}) \\
E_4 &= \{a\} \| \{b\} \\
E_5 &= (\{a\} \| \{b\}) \square \{c\} \\
E_6 &= ((\{a\}; \{b\}) \square \{c\})\ \mathbf{rs}\ c \\
E_7 &= (\{a\} \| \{b\})\ \mathbf{rs}\ b \\
E_8 &= (\{a\} \| \{\hat{a}\})\ \mathbf{sy}\ a \\
E_9 &= ((\{a\} \| \{\hat{a}, \hat{a}\}) \| \{a\})\ \mathbf{sy}\ a.
\end{aligned} \tag{1}$$

Intuitively, expression E_0 can do a silent $\emptyset$ move without visible actions and terminate. Expression E_1 can do a non-silent $\{\hat{a}, b\}$ move. This move is interpreted as performing synchronously (i.e., inseparably at the same time) the two visible action particles $\hat{a}$ and b. Expression E_2 can either do an $\{a\}$ move followed by a $\{b\}$ move, or a $\{c\}$ move, and terminate. Expression E_3 can do an $\{a\}$ move, followed either by a $\{b\}$ move or by a $\{c\}$ move (and terminate). Expression E_4 can make a single $\{a\}$ move followed by a $\{b\}$ move (and terminate), but also a single $\{b\}$ move followed by an $\{a\}$ move and terminate, and also a move consisting of a concurrent execution of $\{a\}$ and $\{b\}$ (and then terminate). Expression E_5 behaves either as expression E_4, or it can make a single $\{c\}$ move and terminate.

Expression E_6 can make an $\{a\}$ move followed by a $\{b\}$ move and terminate, but cannot make a $\{c\}$ move. Expression E_7 can make an $\{a\}$ move but it cannot make a $\{b\}$ move, nor can it terminate properly. Expression E_8 can make all moves of $\{a\} \| \{\hat{a}\}$ and, in addition, also a silent synchronisation. Expression E_9, finally, can make the same moves as the expression $(\{a\} \| \{\hat{a}, \hat{a}\}) \| \{a\}$ and, in addition, some number of synchronisations. For example, it can make a $\{\hat{a}\}$ move synchronising the left and the middle components, or a $\{\hat{a}\}$ move synchronising the middle and the right hand side components, or also a silent move synchronising all three components (and terminate).

The next section defines the spectrum of possible moves of an expression of the syntax. Having given that definition, it is then the objective of Sections 4-6 to define a canonical association of an elementary labelled Petri net to each expression. The class of nets used for this purpose, called boxes, is defined in Section 4. The translation mechanism is explained in Section 5. It allows every control to be treated by transition refinement, and every operator in the second line of the syntax by a generalised type of relabelling. The translation from expressions to nets is defined in Section 6.

3 Operational semantics

In this section, we define a Petri net independent operational semantics for the finite box algebra. We define a *step sequence* semantics, that is, one in terms of sequences of concurrently occurring multiactions. Formally, a step is a finite multiset $\gamma = \{\alpha_1, \ldots, \alpha_m\}$ such that each α_j is a multiaction, that is, a multiset over A. We always denote steps by $\gamma, \delta, \ldots \in \mathcal{M}(\mathcal{L})$. For example, $\{\{\hat{a}, b\}\}$ is a step that could be executed by expression E_1. Also, $\{\emptyset\}$ is a step that could be executed by expression E_0 and $\{\{a\}, \{b\}\}$ is a step that could be executed by expression E_4. Furthermore, $\emptyset$ is also a step. This step contains no multiactions and is therefore interpreted as *inaction*. The step $\{\emptyset\}$, by contrast, denotes an execution of the silent multiaction $\emptyset$ and is therefore distinct from inaction. When steps are concatenated to give step sequences, inaction plays the rôle of the empty word.

The current marking (or state) of an expression is indicated formally by overbarrings and/or underbarrings as in [6, 12]. An *initially marked expression*, by convention, is denoted by $\overline{E}$. Any activity associated with E starts from the initially marked expression $\overline{E}$. If an activity terminates successfully then it leads to the *terminally marked expression* which, by convention, is denoted by $\underline{E}$. We reserve the letters E and F for unmarked expressions defined by the above syntax, and the letters G and H for *marked expressions*, i.e., expressions decorated with over-/underbarrings. We give no syntax for marked expressions, as their possible syntactic forms follow from the rules of the operational semantics.

We define the step sequence semantics of marked expressions according to the format

$$G \xrightarrow{\gamma} G'.$$

This specifies a relation between G, γ and G' that is to be read as follows: the marked expression G is transformed by the step γ into the marked expression G'. The semantics is given axiomatically by induction over the syntax. To start with, we postulate a general axiom of inaction:

$$\text{(Inaction rule)} \qquad G \xrightarrow{\emptyset} G$$

for any marked expression G. The axiom for a single move is as follows:

$$\text{(Basic action rule)} \qquad \overline{\alpha} \xrightarrow{\{\alpha\}} \underline{\alpha}.$$

For instance, using the inaction rule and the basic action rule we may derive for the first two expressions $E_0 = \emptyset$ and $E_1 = \{\hat{a}, b\}$, under their initial markings:

$$\begin{array}{ccc} \overline{\emptyset} & \xrightarrow{\emptyset} & \overline{\emptyset} \\ \overline{\emptyset} & \xrightarrow{\{\emptyset\}} & \underline{\emptyset} \\ \overline{\{\hat{a}, b\}} & \xrightarrow{\emptyset} & \overline{\{\hat{a}, b\}} \\ \overline{\{\hat{a}, b\}} & \xrightarrow{\{\{\hat{a}, b\}\}} & \underline{\{\hat{a}, b\}}. \end{array}$$

Notice the difference between line 1 (inaction) and line 2 (silent action). The former maintains the initial marking of E_0 while the latter transforms the initial marking of E_0 into its final marking.

The following specifies three rules for concurrent composition[3]:

$$(\| \text{ rule}) \qquad \begin{array}{c} \overline{E\|F} \xrightarrow{\emptyset} \overline{E}\|\overline{F} \\[1ex] \dfrac{G \xrightarrow{\gamma} G' \;,\; H \xrightarrow{\delta} H'}{G\|H \xrightarrow{\gamma+\delta} G'\|H'} \\[2ex] \underline{E}\|\underline{F} \xrightarrow{\emptyset} \underline{E\|F}. \end{array}$$

The $\|$ rule combines two additional inaction rules (in the first line and the last line) with a so-called *inference rule* (in the middle). The first inaction rule $\overline{E\|F} \xrightarrow{\emptyset} \overline{E}\|\overline{F}$ can be understood as allowing the marked expression $\overline{E\|F}$ to be rewritten equivalently as $\overline{E}\|\overline{F}$, and the other inaction rule can be read as allowing $\underline{E}\|\underline{F}$ to be rewritten equivalently as $\underline{E\|F}$. More generally, let us define an equivalence relation

$$\equiv \quad = \quad (\xrightarrow{\emptyset} \cup \xleftarrow{\emptyset})^{\star}$$

between marked expressions. The operational semantics we are in the process of defining has the property that two $\equiv$-equivalent marked expressions can make exactly the same steps, and the Petri net semantics we are going to define later has the property that two $\equiv$-equivalent expressions have the same associated marked Petri box.

The inference rule in the middle of the $\|$ rule should be read in the following way:

$$\begin{array}{ll} \text{from} & G \xrightarrow{\gamma} G' \;\text{ and }\; H \xrightarrow{\delta} H' \\ \text{infer} & G\|H \xrightarrow{\gamma+\delta} G'\|H'. \end{array}$$

[3] Here and later, we use $+$ to denote multiset addition.

For example, for expression $E_4 = \{a\}\|\{b\}$ we expect to be able to infer the step

$$\overline{\{a\}\|\{b\}} \xrightarrow{\{\{a\},\{b\}\}} \underline{\{a\}\|\{b\}}.$$

This is possible in the following way:

$$\begin{array}{rcl} \overline{\{a\}\|\{b\}} & \xrightarrow{\emptyset} & \overline{\{a\}}\|\overline{\{b\}} \\ & \xrightarrow{\{\{a\},\{b\}\}} & \underline{\{a\}}\|\underline{\{b\}} \\ & \xrightarrow{\emptyset} & \underline{\{a\}\|\{b\}}. \end{array}$$

In the first line of this inference, the first inaction rule of the $\|$ operator is used with the special case $E = \{a\}$ and $F = \{b\}$. In the second line, the inference rule of $\|$ is used with $G = \overline{\{a\}}$, $H = \overline{\{b\}}$, $G' = \underline{\{a\}}$ and $H' = \underline{\{b\}}$. However, in order to be allowed to use this rule, we have to appeal to the basic action rule twice, deriving $\overline{\{a\}} \xrightarrow{\{\{a\}\}} \underline{\{a\}}$ and $\overline{\{b\}} \xrightarrow{\{\{b\}\}} \underline{\{b\}}$ in turn. Thus, the second line of the above inference embodies a threefold application of the rules given so far. Finally, the third line of the derivation is an application of the second inaction rule of the $\|$ operator.

The following (and its symmetric version with respect to exchanging left hand sides and right hand sides of $\square$) specifies the set of rules for the choice operator $\square$:

$$(\square \text{ rule}) \qquad \begin{array}{c} \overline{E \square F} \xrightarrow{\emptyset} \overline{E} \square F \\[1ex] \dfrac{G \xrightarrow{\gamma} G'}{G \square F \xrightarrow{\gamma} G' \square F} \\[2ex] \underline{E} \square F \xrightarrow{\emptyset} \underline{E \square F}. \end{array}$$

Let us check the $\square$ rule on the example $E_5 = (\{a\}\|\{b\}) \square \{c\}$. We expect that from the marked expression $\overline{(\{a\}\|\{b\}) \square \{c\}}$, the following steps (amongst others) are possible:

$\{\{a\},\{b\}\}$	(leading to a final state)
$\{\{a\}\}$	(leading to a non-final state)
$\{\{c\}\}$	(leading to a final state).

Also, we expect that $\{\{a\},\{c\}\}$, for instance, is not a step. We show that the first of those expectations is true. The verification of the other claims is left to the reader.

$$\begin{aligned}\overline{(\{a\}\|\{b\})\,\Box\,\{c\}} &\xrightarrow{\emptyset} \overline{(\{a\}\|\{b\})}\,\Box\,\{c\}\\ &\xrightarrow{\{\{a\},\{b\}\}} \underline{(\{a\}\|\{b\})}\,\Box\,\{c\}\\ &\xrightarrow{\emptyset} \underline{(\{a\}\|\{b\})\,\Box\,\{c\}}.\end{aligned}$$

In this inference, the first line comes from the first inaction rule for $\Box$ (left hand side). The second line is due to the $\Box$ inference rule (three times) in combination with the $\|$ inference rule, twice the basic action rule, and the two $\|$ inaction rules. Thus, this line comprises an eightfold application of the rules. The last line of the inference comes from the second $\Box$ inaction rule.

The following rules describe sequential composition. This set of rules combines three inaction rules with one inference rule[4]:

$$\text{(; rule)}\qquad \begin{gathered}\overline{E;F} \xrightarrow{\emptyset} \overline{E};F\\ \underline{E};F \xrightarrow{\emptyset} E;\overline{F}\\ E;\underline{F} \xrightarrow{\emptyset} \underline{E;F}\\ \dfrac{G \xrightarrow{\gamma} G'\ ,\ H \xrightarrow{\delta} H'}{G;F \xrightarrow{\gamma} G';F\ ,\ E;H \xrightarrow{\delta} E;H'.}\end{gathered}$$

Let us explain the semicolon rule on example $E_2 = (\{a\};\{b\})\,\Box\,\{c\}$:

$$\begin{aligned}\overline{(\{a\};\{b\})\,\Box\,\{c\}} &\xrightarrow{\emptyset} \overline{(\{a\};\{b\})}\,\Box\,\{c\}\\ &\xrightarrow{\emptyset} (\overline{\{a\}};\{b\})\,\Box\,\{c\}\\ &\xrightarrow{\{\{a\}\}} (\underline{\{a\}};\{b\})\,\Box\,\{c\}\\ &\xrightarrow{\emptyset} (\{a\};\overline{\{b\}})\,\Box\,\{c\}\\ &\xrightarrow{\{\{b\}\}} (\{a\};\underline{\{b\}})\,\Box\,\{c\}\\ &\quad \ldots\end{aligned}$$

Thus, indeed, as may be expected, $\overline{E_2}$ may do an $\{a\}$ move followed by a $\{b\}$ move (after some inaction steps). In this inference, the first line comes from $\Box$ inaction. The second line comes from ';' inaction together with the $\Box$ inference rule. Line 3 uses three inference rules: the basic action rule, the sequence rule and the choice rule. Line 4 follows from the second ';' inaction rule and the choice inference rule. The second part of the inference is symmetrical to its first part.

Let us now turn our attention to the operators in the second line of the syntax, i.e., relabelling, restriction and synchronisation. Let f be a conjugate-preserving function from A to A. By the inductive definition $f(\emptyset) = \emptyset$ and $f(\alpha + \{a\}) = f(\alpha) + \{f(a)\}$, f can be lifted to finite multisets over A (multiactions). The expression $E[f]$ has the following rules:

$$\text{([}f\text{] rule)}\qquad \begin{gathered}\overline{E[f]} \xrightarrow{\emptyset} \overline{E}[f]\\ \dfrac{G \xrightarrow{\{\alpha_1\}+\ldots+\{\alpha_m\}} G'}{G[f] \xrightarrow{\{f(\alpha_1)\}+\ldots+\{f(\alpha_m)\}} G'[f]}\\ \underline{E}[f] \xrightarrow{\emptyset} \underline{E[f]}.\end{gathered}$$

The next set of rules describes restriction.

$$\text{(}\mathbf{rs}\text{ rule)}\qquad \begin{gathered}\overline{E\ \mathbf{rs}\ a} \xrightarrow{\emptyset} \overline{E}\ \mathbf{rs}\ a\\ \dfrac{G \xrightarrow{\gamma} G'\ ,\ \gamma \in \mathcal{M}(\mathcal{M}(A\backslash\{a,\hat{a}\}))}{G\ \mathbf{rs}\ a \xrightarrow{\gamma} G'\ \mathbf{rs}\ a}\\ \underline{E}\ \mathbf{rs}\ a \xrightarrow{\emptyset} \underline{E\ \mathbf{rs}\ a}.\end{gathered}$$

We omit examples for the last two rules as they are fairly self-explanatory. As an exercise, the reader might like to verify that $E_6 = ((\{a\};\{b\})\,\Box\,\{c\})\ \mathbf{rs}\ c$ can make exactly the same moves as $E_6' = \{a\};\{b\}$ and to explore the difference between $E_7 = (\{a\}\|\{b\})\ \mathbf{rs}\ b$ and $E_7' = \{a\}$.

Finally, we specify a set of rules for the synchronisation scheme defined in [2]. Because this scheme works inductively, allowing new synchronisations to be created from basic ones, the inference rule for synchronisation

[4] Or two inference rules, depending on how one counts them.

needs two constituent rules as follows:

$$\overline{E \textbf{ sy } a} \xrightarrow{\emptyset} \overline{E} \textbf{ sy } a$$

$$(\textbf{sy} \text{ rule}) \quad \frac{G \xrightarrow{\gamma} G'}{G \textbf{ sy } a \xrightarrow{\gamma} G' \textbf{ sy } a} \qquad \frac{G \textbf{ sy } a \xrightarrow{\{\alpha+\{a\}\}+\{\beta+\{\hat{a}\}\}+\gamma} G' \textbf{ sy } a}{G \textbf{ sy } a \xrightarrow{\{\alpha+\beta\}+\gamma} G' \textbf{ sy } a}$$

$$\underline{E} \textbf{ sy } a \xrightarrow{\emptyset} \underline{E \textbf{ sy } a}.$$

The first constituent of the inference rule simply states that G **sy** a can mimic all the steps of G. The second constituent of the inference rule states that if G **sy** a can make a move to G' **sy** a involving two multiactions, one containing a and the other containing $\hat{a}$, then G **sy** a can also make a move in which these two actions are combined into a single one, less the synchronising pair, also leading to G' **sy** a. This rule can be applied repeatedly provided that conjugate pairs are suitably distributed, because above and below the horizontal line the same marked expressions appear.

We may check this rule on the example E_8, but instead, let us check the more complicated example

$$E_9 = ((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\}) \textbf{ sy } a$$

because this example involves a repeated application of the second inference rule, and because the example E_8 can be treated analogously. We wish to show that the initially marked expression $\overline{E_9}$ can make a $\{\emptyset\}$ move synchronising all of its concurrent components and terminating its execution. Because this inference is somewhat more involved than the others, we split it into several parts.

$$\begin{aligned}\overline{E_9} &\xrightarrow{\emptyset} \overline{((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\})} \textbf{ sy } a \\ &\xrightarrow{\{\emptyset\}} \underline{((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\})} \textbf{ sy } a \\ &\xrightarrow{\emptyset} \underline{((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\}) \textbf{ sy } a}.\end{aligned}$$

The first and third lines of this inference follow from the first inaction rule and the second inaction rule, respectively, of **sy**. The second line incorporates one application of the first **sy** inference rule, two applications of the second **sy** inference rule, several applications of all parts of the $\|$ rule, and three applications of the basic action rule. This conglomerate of rule applications is decomposed as follows. First, with the same argument as before (using three times the basic action rule, twice both $\|$ inaction rules and twice the $\|$ inference rule), the following may be derived:

$$\begin{aligned}&\overline{(\{a\}\|\{\hat{a},\hat{a}\})\|\{a\}} \\ &\xrightarrow{\{\{a\},\{\hat{a},\hat{a}\},\{a\}\}} \\ &\underline{(\{a\}\|\{\hat{a},\hat{a}\})\|\{a\}}.\end{aligned}$$

Using the first **sy** inference rule, the following can subsequently be deduced:

$$\begin{aligned}&\overline{((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\})} \textbf{ sy } a \\ &\xrightarrow{\{\{a\},\{\hat{a},\hat{a}\},\{a\}\}} \\ &\underline{((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\})} \textbf{ sy } a.\end{aligned}$$

We may rewrite this as follows:

$$\begin{aligned}&\overline{((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\})} \textbf{ sy } a \\ &\xrightarrow{\{\emptyset+\{a\}\}+\{\{\hat{a}\}+\{\hat{a}\}\}+\{\{a\}\}} \\ &\underline{((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\})} \textbf{ sy } a.\end{aligned}$$

Using $\alpha = \emptyset$, $\beta = \{\hat{a}\}$ and $\gamma = \{\{a\}\}$, this matches the premise of the second **sy** inference rule. We may now use this rule to deduce the following:

$$\begin{aligned}&\overline{((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\})} \textbf{ sy } a \\ &\xrightarrow{\{\emptyset+\{\hat{a}\}\}+\{\{a\}\}} \\ &\underline{((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\})} \textbf{ sy } a.\end{aligned}$$

This may be rewritten as follows:

$$\begin{aligned}&\overline{((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\})} \textbf{ sy } a \\ &\xrightarrow{\{\emptyset+\{a\}\}+\{\emptyset+\{\hat{a}\}\}+\emptyset} \\ &\underline{((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\})} \textbf{ sy } a.\end{aligned}$$

Using $\alpha = \emptyset = \beta$ and $\gamma = \emptyset$, this again matches the premise of the second **sy** rule. The rule can be used again, yielding

$$\begin{array}{l} \overline{((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\})}\ \mathbf{sy}\ a \\ \xrightarrow{\{\emptyset+\emptyset\}+\emptyset} \\ \underline{((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\})}\ \mathbf{sy}\ a. \end{array}$$

The step $\{\emptyset + \emptyset\} + \emptyset$ is another way of writing the step $\{\emptyset\}$. Thus, the fact that the initially marked expression $\overline{E_9}$ can execute the step $\{\emptyset\}$ and terminate, has been formally deduced from the rules.

4 A domain of boxes

In this section we define a domain of labelled nets with certain additional properties called *boxes*. This domain contains the objects to be associated with expressions. An (arc-weighted) *net* is a triple (S, T, W) such that S and T are disjoint sets of *places* and *transitions*, respectively, and W is a *weight function* from the set $((S \times T) \cup (T \times S))$ to the set $\mathbf{N}$ of natural numbers. A net (S, T, W) is called an *S-net* iff

$$\forall t \in T\colon \sum_{s \in S} W(s,t) = 1 = \sum_{s \in S} W(t,s).$$

A *labelled net* is a four-tuple $\Sigma = (S, T, W, \lambda)$ such that (S, T, W) is a net and λ is a function from $S \cup T$ to some suitable sets which, for our purpose, are chosen as follows. By definition, λ maps S onto the set $\{e, i, x\}$. If $\lambda(s) = e$ then s is called an *entry place*. If $\lambda(s) = i$ then s is called an *internal place*. If $\lambda(s) = x$ then s is called an *exit place*. For $t \in T$, $\lambda(t)$ is a relation, more specifically, a subset of $(\mathcal{M}(\mathcal{L})) \times \mathcal{L}$. We will use the letter ρ for such relations and μ for elements of $\mathcal{M}(\mathcal{L})$ ranging over the domain of ρ. For example, the *identity relation* is defined as follows:

$$id = \{(\{\alpha\}, \alpha) \mid \alpha \in \mathcal{L}\} \subseteq (\mathcal{M}(\mathcal{L})) \times \mathcal{L}.$$

Other important relations are the *constant relations* which associate a constant element of $\mathcal{L}$ to every element of $\mathcal{M}(\mathcal{L})$. Each such relation can - and will - be identified with an element of $\mathcal{L}$. In the companion paper [5], a simplified setup is considered where only the identity relation and the constant relations are relevant. In that case, the labelling function for transitions can simply be defined as a partial function from T to $\mathcal{L}$.

If Σ is a labelled net, by convention, ${}^\bullet\Sigma$ and $\Sigma^\bullet$ denote the set of e-labelled places and the set of x-labelled places, respectively, of Σ. For any place $s \in S$, ${}^\bullet s$ ($s^\bullet$) denotes the set of input (output, respectively) transitions of s. For a set of places $R \subseteq S$, ${}^\bullet R = \bigcup\{{}^\bullet s \mid s \in R\}$ and $R^\bullet = \bigcup\{s^\bullet \mid s \in R\}$. Similar notation is used for transitions and sets of transitions. A *box* is defined to be a labelled net such that the following hold:

$$\begin{array}{ll} & {}^\bullet({}^\bullet\Sigma) = \emptyset = (\Sigma^\bullet)^\bullet \\ \text{and} & {}^\bullet\Sigma \neq \emptyset \neq \Sigma^\bullet \\ \text{and} & \forall t \in T\colon {}^\bullet t \neq \emptyset \neq t^\bullet. \end{array}$$

The first of these conditions requires entry places to be free of incoming arcs and exit places to be free of outgoing arcs. The second condition requires that there is at least one entry place and at least one exit place. The third condition requires that every transition has at least one preplace and at least one postplace.

A box is called *plain* if for all $t \in T$, $\lambda(t)$ is a constant relation, that is, an element of $\mathcal{L}$. A finite box is called an *operator box* iff the underlying net is the disjoint union of connected S-nets, each such S-net comprising exactly one e-labelled place and one x-labelled place and, furthermore, there are no side conditions[5] (i.e., $\forall t \in T\colon {}^\bullet t \cap t^\bullet = \emptyset$). We will henceforth be interested only in these two kinds of boxes. Plain boxes are interpreted as

[5] The last requirement is added because it makes the theory simple. It can be lifted smoothly, however; see [8].

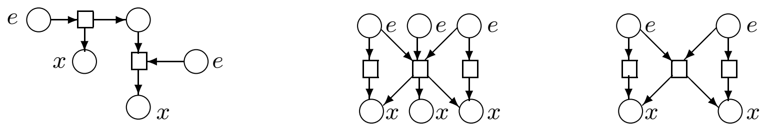

Fig. 1. Two non-factorisable boxes (left and middle) and a factorisable box (right)

the basic semantical objects of interest. The net semantics of an expression, in particular, will always be a plain box[6]. An operator box, by contrast, is interpreted as taking a certain number (say, n) of plain boxes as arguments and creating a new plain box from them. That is, it specifies an n-ary function from plain boxes to plain boxes. We will describe this interpretation of operator boxes in the next section.

A *marking* M of (S, T, W, λ) is a function from S to the natural numbers. A marking is *safe* iff $M(S) \subseteq \{0, 1\}$. A safe marking can - and will - be represented as a subset of S. A *marked box* (Σ, M) is a box Σ together with a marking M. The *initially marked box corresponding to* Σ is defined to be $(\Sigma, {}^\bullet\Sigma)$. That is, the initial marking of a box contains a single token on every entry place and no token anywhere else.

The behaviour of a marked box is defined as usual in net theory [16]. In particular, a step sequence of an initially marked box is defined as a sequence of multisets of concurrently enabled transitions (notice that they are multisets in general since a transition may be concurrently enabled to itself; but if the box is safe - see below - then they are sets).

Applying the labelling function λ elementwise to every transition in some step sequence, one gets an *abstracted step sequence*, i.e., a sequence of multisets over $\mathcal{L}$ (they are multisets in general, even if the box is safe, since two concurrent transitions may carry the same label). Notice that the abstracted step sequences of a plain box are the same kinds of objects as the step sequences generated by the operational semantics defined in the previous section.

[6] The reader will find plenty of examples - ten, to be precise - later on in the paper.

The behavioural properties of a box can be defined in the usual way. In particular, a box Σ is called *safe* iff all markings reachable in $(\Sigma, {}^\bullet\Sigma)$ are safe. We define two further properties that are of special importance for boxes. A safe box Σ is called *clean* iff all markings M reachable in $(\Sigma, {}^\bullet\Sigma)$ satisfy $\Sigma^\bullet \subseteq M \Rightarrow \Sigma^\bullet = M$. A safe box Σ is called *factorisable* iff all markings M reachable in $(\Sigma, {}^\bullet\Sigma)$ satisfy that for any set U of transitions concurrently enabled by M, the tokens in $M \setminus {}^\bullet U$ can be distributed in such a way that there are just enough of them to concurrently enable (or backward enable) another set of transitions.

Figure 1 shows three examples explaining the factorisability property. The first two boxes shown in this figure are not factorisable while the third is factorisable (all three are safe and clean). Intuitively, factorisability means that every reachable marking can be suitably distributed over sets of concurrently enabled (and/or backward enabled) transitions. This is violated, for instance, for the initial marking of the middle net of Figure 1, because the two outermost tokens enable the two outermost transitions, but after taking them away the middle token is not sufficient for enabling any other set of transitions.

Lemma: Operator boxes are safe, clean and factorisable.

Proof: Straightforward from the S-net decomposability property.

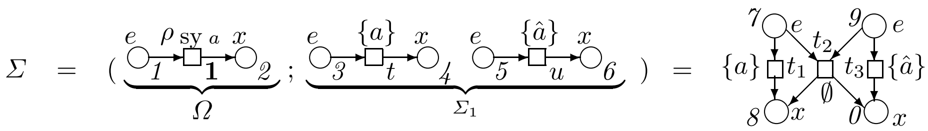

Fig. 2. An example for refinement with relabelling

5 Refinement and relabelling

In this section we describe a general mechanism by which refinement and relabelling are combined. Both refinement and relabelling are done by an operator box, the former by its transitions and the latter by the labels of those transitions. More precisely, refinement is done by letting an operator box with n transitions take n plain boxes as arguments and letting it create a new plain box by refining its transitions by the argument boxes. Relabelling is done by applying the labelling of an operator box transition every time this transition is refined by some plain box, as described below.

We exemplify the mechanism on an operator box with only one transition, and only one plain box by which this transition is to be refined, because the main principle can be understood most easily in this simple case. In what follows, let Ω be an operator box with a labelling λ_Ω and one transition called $\mathbf{1}$ with a label $\rho = \lambda_\Omega(\mathbf{1})$. Moreover, let Σ_1 be a plain box with labelling λ_1. Then by

$$\Sigma \;=\; (\Omega; \Sigma_1)$$

we denote the fact that Σ (with labelling function λ) arises out of Ω by the refinement of transition $\mathbf{1}$ by Σ_1 according to the following general rule:

- The place sets ${}^\bullet\mathbf{1}$ and ${}^\bullet\Sigma_1$ are combined pairwise, and[7] the place sets $\mathbf{1}^\bullet$ and $\Sigma_1^\bullet$ are combined pairwise, yielding new places connecting Σ_1 with its environment in Σ. The labels of the new places are inherited from Ω.
- For every pair $\pi = (\mu, \alpha)$ in ρ and every non-empty multiset τ of transitions of Σ_1 such that their labels match the first element of the pair π, i.e., $\mu = \{\lambda_1(t) \mid t \in \tau\}$, a new transition is created in Σ. This new transition inherits its connectivity from the multiset sum of the connectivites of τ in Σ_1, and its label is α, the second element of the pair π.

[7] Separately, because there are no side conditions in the operator box.

We illustrate this construction on the example shown in Figure 2. The reader will have no difficulty in recognising a possible net semantics of the expression $E_8 = (\{a\}\|\{\hat{a}\})\ \mathbf{sy}\ a$ in this figure. First, we explain the four places of Σ. Place *7* comes from combining place *1* of the operator box with place *3* of Σ_1. In the same way,

place *8* comes from *2* and *4*,
place *9* comes from *1* and *5*,
place *0* comes from *2* and *6*.

In order to explain the three transitions of Σ it is necessary to examine the relation $\rho_{\,\mathrm{sy}\ a}$ more closely. This relation describes the relabelling that is to be applied to Σ_1 during the refinement of transition $\mathbf{1}$ by it. The synchronisation discipline of the box algebra [2] can be embodied in the relation $\rho_{\,\mathrm{sy}\ a}$ by requiring it to contain pairs (μ, α) if and only if the labels in μ, synchronised according to that discipline, yield a transition labelled α. A relation defined in such a general way contains a large number of elements, several ones for every pair $a, \hat{a}$ to describe all of its possible combinations with other action names. For any given Σ_1 that is to be relabelled by $\rho_{\text{ sy } a}$, only a few of these pairs are actually relevant. In our case depicted in Figure 2, the following three elements of $\rho_{\text{ sy } a}$ are relevant because they can be matched by the labels of multisets of transitions of Σ_1:

$$\rho_{\text{ sy } a} = \{\ldots, (\{\{a\}\}, \{a\}), (\{\{\hat{a}\}\}, \{\hat{a}\}), \\ (\{\{a\}, \{\hat{a}\}\}, \emptyset), \ldots\}.$$

Transition t_1 of Σ (on the right hand side of Figure 2) is created by the pair $(\{\{a\}\}, \{a\})$ of $\rho_{\text{ sy } a}$ using the singleton multiset $\{t\}$ of transitions of Σ_1. Similarly, transition t_3 is created by the pair $(\{\{\hat{a}\}\}, \{\hat{a}\})$ using the multiset $\{u\}$. Finally, transition t_2 is created by the pair $(\{\{a\}, \{\hat{a}\}\}, \emptyset)$ of $\rho_{\text{ sy } a}$ using the multiset $\{t, u\}$.

Using this example as a guideline it is not hard to see how the labelling of an operator box must be defined so that it describes restriction rather than synchronisation. Let us assume that the operator **rs** a is to be described by the relation $\rho_{\text{ rs } a}$. Then any multiset μ which mentions either an a or a $\hat{a}$ or both, must lie outside the domain of $\rho_{\text{ rs } a}$, leading to the creation of the empty set of transitions. Otherwise, $\rho_{\text{ rs } a}$ is simply the identity relation. In other words, $\rho_{\text{ rs } a}$ contains exactly all pairs $(\{\alpha\}, \alpha)$ with $a, \hat{a} \notin \alpha$. In a similar way, elementary relabelling $E[f]$ with a conjugate-preserving function $f\colon A \to A$ is described by a relation $\rho_{[f]}$ containing all pairs of the form $(\{\alpha\}, f(\alpha))$ for $\alpha \in \mathcal{L}$. Finally, if the label of transition $\mathbf{1}$ of Ω is the identity relation id, then refining Σ_1 into Ω simply creates an isomorphic copy of Σ_1.

Let us now proceed by exploring meaningful generalisations of this refinement mechanism. First, we may consider an operator box Ω with an arbitrary (finite) number of transitions, say n, ordered as indicated by the names $\mathbf{1}, \mathbf{2}, \mathbf{3}, \ldots$ Then, by convention, the equation

$$\Sigma \;=\; (\Omega; \Sigma_1, \ldots, \Sigma_n)$$

defines Σ to be the box arising from the simultaneous refinement of the n transitions $\mathbf{1}, \ldots, \mathbf{n}$ of Ω by the respective plain boxes $\Sigma_1, \ldots, \Sigma_n$ as explained in [5, 8]. For the present purpose, it suffices to think of simultaneous refinement as refinement in any arbitrary order, since the resulting box is (up to isomorphism) independent of this order. Note that the box Σ is plain as well, which follows from the fact that all refining boxes Σ_i are plain and that all transitions of the operator box Ω are replaced. A next generalisation is to allow the re-use of Σ on the right hand side of another equation, that is, to allow the nesting of such equations. In general, we may consider a system of equations as follows:

$$\begin{aligned} \Sigma_1 &= (\Omega_1; \Sigma_1^1, \Sigma_1^2, \ldots) \\ \Sigma_2 &= (\Omega_2; \Sigma_2^1, \Sigma_2^2, \ldots) \\ &\cdots \end{aligned} \tag{2}$$

with mutually distinct letters Σ_j (box variables) and each Σ_j^l being either a plain box or a letter Σ_i occurring on the left hand side of another equation. Such a system may be infinite, but in this paper we require it to be finite and recursion free in the sense that its dependency graph contains no cycles[8]. The set of plain boxes $\Sigma_1, \Sigma_2, \ldots$ can be calculated by executing the following procedure: starting with equations that do not refer to other equations on their right hand sides (such equations must exist due to recursion freeness), use simultaneous refinement repeatedly until all left hand sides have been computed. Note that the solution is unique.

Lemma: If all plain boxes occurring on the right hand sides of (2) are safe and clean then the same is true for the box Σ_j obtained for the j'th equation ($j = 1, 2, \ldots$).

[8] The dependency graph, by definition, is the directed graph whose nodes are the set of letters $\{\Sigma_1, \Sigma_2, \ldots\}$ and whose arcs (Σ_j, Σ_i) indicate that Σ_i occurs on the right hand side of the equation having Σ_j as its left hand side.

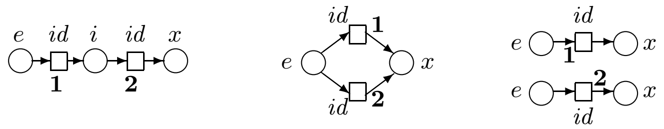

Fig. 3. Three operator boxes for ; (left), + (middle) and $\|$ (right)

Fig. 4. Equations associated to expressions $E_0 = \emptyset$ and $E_1 = \{\hat{a}, b\}$

Proof: Operator boxes are safe and clean by the previous lemma, and the property propagates through refinement as proved in [8].

A further generalisation would allow such systems of equations to be infinite and/or to contain recursion, but we will not deal with this generalisation in the present paper. The reader is referred to [5] for a description of what happens when recursion is added.

6 Box semantics and results

Using the general refinement and relabelling mechanism defined in the previous section we may now define a translation which associates a box to every expression E of the syntax defined in Section 2. The first step of the translation transforms E syntactically into an equivalent set of (auxiliary) operator box equations of the form defined in Section 5. This is always possible in a canonical way using only the three operator boxes shown in Figure 3 and, furthermore, operator boxes like Ω in Figure 2 but with the transition labelling being any of the specific relations id, $\rho_{\text{sy } a}$, $\rho_{\text{rs } a}$ or $\rho_{[f]}$. Moreover, in the right hand sides of the box equations, we use two-place one-transition plain boxes corresponding to multiactions. Note that the three operator boxes in Figure 3 correspond directly to the connectives of the first line of the syntax of E, and that the four relabelling relations just mentioned correspond to the second line of the syntax (except for the relation id which may be used to describe multiactions).

By the results of the previous section, these equations define a unique plain box as the semantics of E. We illustrate the translation so defined on the examples E_0–E_9 given earlier. Figures 4 to 10 show the equation systems associated with each of the ten expressions E_0–E_9, yielding boxes Σ_0 for E_0, Σ_1 for E_1, Σ_2^1 for E_2, and so on. Finally, Figure 11 shows all those boxes directly rather than expressed by equations.

The following theorems (whose proofs are in [11]) express consistency and completeness of the operational semantics with respect to the Petri box semantics. Note that the fact that these two theorems can be formulated is due in the first place to the step sequences of the operational semantics E and the (abstracted) step sequences of the box of E being comparable objects.

Theorem 1: Every step sequence of the operational semantics of $\overline{E}$ is also an abstracted step sequence of the initially marked box of E.

Theorem 2: Every abstracted step sequence of the initially marked box of E is also a step sequence of the operational semantics of $\overline{E}$.

The proof of the first result depends on the

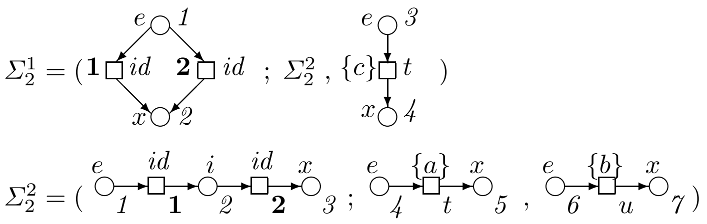

Fig. 5. Equations associated to expression $E_2 = (\{a\};\{b\}) \square \{c\}$

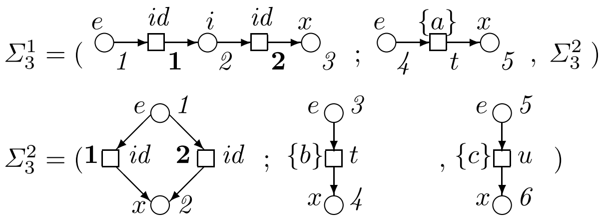

Fig. 6. Equations associated to expression $E_3 = \{a\};(\{b\} \square \{c\})$

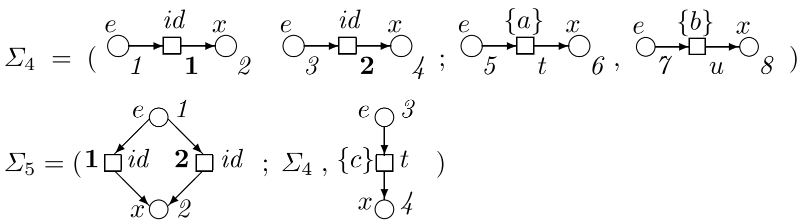

Fig. 7. Equations associated to expressions $E_4 = \{a\}\|\{b\}$ and $E_5 = (\{a\}\|\{b\}) \square \{c\}$

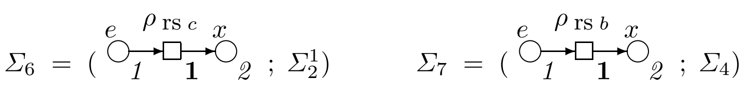

Fig. 8. Equations for $E_6 = ((\{a\};\{b\}) \square \{c\})$ **rs** c and $E_7 = (\{a\}\|\{b\})$ **rs** b

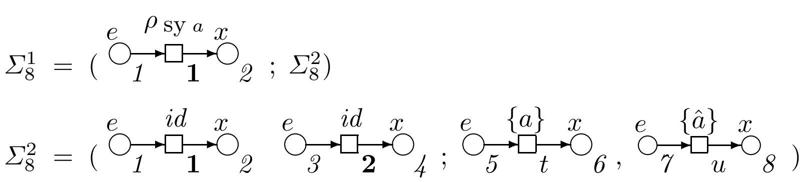

Fig. 9. Equations associated to expression $E_8 = (\{a\}\|\{\hat{a}\})$ **sy** a

way boxes are constructed and is done by an induction argument over the tableau of an inference. The proof of the second result (completeness) is done by induction over the syntax of E. It depends on operator boxes being safe, clean and factorisable and on plain boxes being safe and clean. Factorisability corresponds directly to the property expressed, for instance, in the two inaction rules of the $\|$ rule. The first inaction rule states that the initial marking $\overline{E\|F}$ of the whole expression $E\|F$ can equivalently be interpreted as being factorised into two initial markings, one for each of the constituents, that is, as $\overline{E}\|\overline{F}$. The second inaction rule states the same for the final markings. This observation is general. More precisely, without infringing the two theorems cited above, it is possible to add to the syntax any control connective whose operator box is safe, clean and factorisable and whose operational semantics is equivalent to the semantics of the operator box. In particular, iteration can be treated in this way because its operator box is a connected S-net. One might even go so far as define the control connectives of the syntax directly by means of operator boxes. In terms of relabellings, too, a significant extension of the syntax is embodied in the mechanism. Any operator that can be expressed in terms of a relabelling relation

$$\rho \subseteq (\mathcal{M}(\mathcal{L})) \times \mathcal{L}$$

can be added to the syntax. The three operators of basic relabelling, restriction and synchronisation are just exemplary instances of such operators. In particular, other synchronisation schemes with other rules (such as the more liberal schemes devised and explored in [7, 9]) can be accommodated in the framework by a modification of the synchronisation relation $\rho_{\,\mathrm{sy}\ a}$. In some sense, the refinement/relabelling mechanism takes the box algebra idea of separating control connectives from relabelling operators to its limit by separating out the two types of connectives almost orthogonally. In addition, the same idea can be applied to recursive expressions, but the extension is significant and cannot be described in this paper; the reader is referred to [5].

7 Concluding remarks

The consistency and completeness results captured by theorems 1 and 2 can be further strengthened [11, 12] to a kind of bisimulation relation between the operational semantics defined directly for process expressions, and the step sequence semantics of the corresponding boxes.

Acknowledgements

The inductive rule for synchronisation is due to Richard P. Hopkins. We would like to thank Raymond Devillers and Javier Esparza for helpful comments both on the contents and on presentation, and Burkhard Graves for detecting a few last-minute mistakes. This work was done within the Esprit Basic Research Working Group 6067 CALIBAN (Causal Calculi Based on Nets).

References

1. E.Best, R.Devillers, J.Esparza: General Refinement and Recursion Operators for the Petri Box Calculus. Springer-Verlag, Lecture Notes in Computer Science Vol. 665, 130-140 (1993).
2. E.Best, R.Devillers, J.Hall: The Petri Box Calculus: a New Causal Algebra with Multilabel Communication. Advances in Petri Nets 1992, G.Rozenberg (ed.), Springer-Verlag, Lecture Notes in Computer Science Vol. 609, 21-69 (1992).
3. E.Best, H.Fleischhack, W.Frączak, R.P.Hopkins, H.Klaudel, E.Pelz: An M-net Semantics of $B(PN)^2$. Proc. of STRICT'95 (1995).

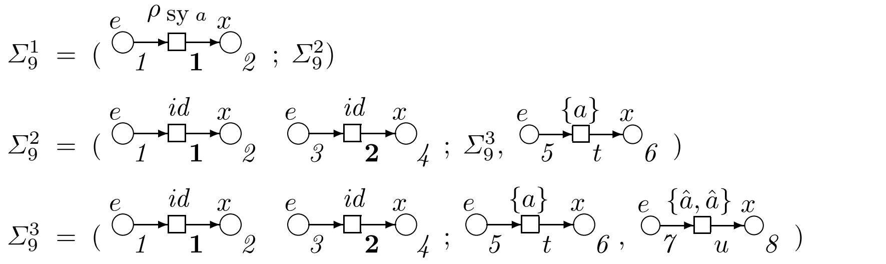

Fig. 10. Equations associated to expression $E_9 = ((\{a\}\|\{\hat{a},\hat{a}\})\|\{a\})$ **sy** a

4. E.Best, R.P.Hopkins: $B(PN)^2$ - a Basic Petri Net Programmming Notation. Proc. of PARLE'93, Springer-Verlag, Lecture Notes in Computer Science Vol. 694, 379-390 (1993).
5. E.Best, M.Koutny: Solving Recursive Net Equations. Proc. of ICALP-95, Springer-Verlag, Lecture Notes in Computer Science (1995).
6. G.Boudol, I.Castellani: Flow Models of Distributed Computations: Event Structures and Nets. Rapport de Recherche, INRIA, Sophia Antipolis (July 1991).
7. R.Devillers: The Synchronisation Operator Revisited for the Petri Box Calculus. Technical Report LIT-290, Laboratoire d'Informatique Théorique, Université Libre de Bruxelles (1994).
8. R.Devillers: S-invariant Analysis of Petri Boxes. Technical Report LIT-273, Laboratoire d'Informatique Théorique, Université Libre de Bruxelles (1993). To appear in Acta Informatica (1995).
9. W.Frączak, H.Klaudel: A Multi-action Synchronisation Scheme and its Application to the Petri Box Calculus. Proc. of ESDA (Engineering Systems Design and Analysis Conference), 91-100, London (1994).
10. U.Goltz, R.van Glabbeek: Refinement of Actions in Causality Based Models. Springer-Verlag, Lecture Notes in Computer Science Vol.430, 267-300 (1989).
11. M.Koutny, E.Best: Operational Semantics for the Box Algebra. Draft paper (1995).
12. M.Koutny, J.Esparza, E.Best: Operational Semantics for the Petri Box Calculus. Proc. of CONCUR'94 (ed. B.Jonsson and J.Parrow), Lecture Notes in Computer Science Vol.836, Springer-Verlag, 210-225 (1994).
13. R.Milner: *Communication and Concurrency.* Prentice Hall (1989).
14. E.R.Olderog: *Nets, Terms and Formulas.* Cambridge Tracts in Theoretical Computer Science 23 (1991).
15. G.Plotkin: A Structural Approach to Operational Semantics. DAIMI Technical Report FN-19, Computer Science Department, University of Århus (1981).
16. W.Reisig: *Petri Nets. An Introduction.* EATCS Monographs on Theoretical Computer Science Vol. 3, Springer-Verlag (1985).

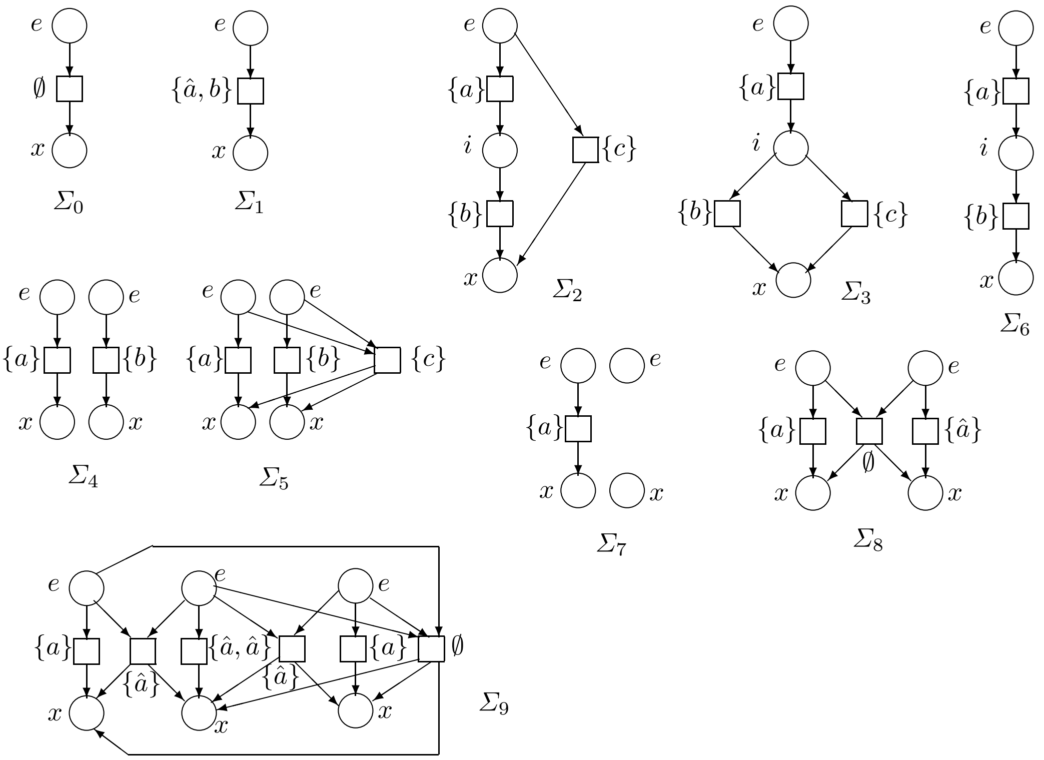

Fig. 11. The boxes Σ_0–Σ_9 associated to expressions E_0–E_9