# Kleene Theorems for Synchronous Products with Matching

Ramchandra Phawade and Kamal Lodaya

The Institute of Mathematical Sciences, CIT Campus, Chennai *600113*, India

**Abstract.** In earlier work [LMP11], we showed that a graph-theoretic condition called "structural cyclicity" enables us to extract syntax from a conflict-equivalent product system of automata. In this paper we have a "pairing" property in our syntax which allows us to connect to a broader class of synchronous product systems [Arn94] with a "matching" property, where the conflict-equivalence is not statically fixed. These systems have been related to labelled free choice nets.

## 1 Introduction

The Kleene and Büchi theorems link finite automata, a model of sequential computation, to regular expressions and monadic second-order logic, both syntactic entities. More than a decade ago [LW00], these ideas were extended to branching automata, a model of bounded fork-join concurrency. It was also observed [LRR03] that on the models side we can have a labelled Petri net representation called SR-systems. The lack of an explicit synchronization mechanism is manifest, and a first work was to solve the Kleene problem for 1-bounded T-systems, going through an intermediate automaton mechanism called T-products. It was observed in this work [LMP11] that the proofs extend to a class of FC-products (conflict-equivalent products of automata, these will be defined below, they are known to be weaker than 1-bounded nets [Zie87,Muk11]), restricted to a graph-theoretic property called "structural cyclicity" which translates in the syntax to disallowing nested Kleene star operators as in regular expressions. But we did not rely on a renaming operator in the syntax, as has been the case with earlier efforts on 1-bounded nets [Gra81,GR92].

In the present paper[1], we make an effort to match the live and 1-bounded free choice nets, a very well-studied subclass [Hac72] with more efficient analysis and algorithms [DE95], but with labelled transitions. It has been claimed that free choice nets can be useful in business process modelling [SH96], but our motivation is more conceptual than dictated by business concerns. We have not studied the logical (Büchi) side of our problem, we note that it has been argued that it suffices to consider free choice nets for the decidability of the monadic second-order theory [TY14].

[1] A preliminary version of this paper appeared at the 8th PNSE workshop in Tunis [PL14]. We thank the organizers for all the help provided for the presentation.

As in our earlier paper [LMP11], we rely on an intermediate formalism of products of automata, without any structural cyclicity property. This time the products are equipped with an "FC-matching" condition derived from Zielonka automata [Zie87] (which can describe 1-bounded nets) and Arnold and Nivat's synchronous products [Arn94], but restricted to stay within the labelled free choice nets. On the syntax side we do not place any restriction on the Kleene stars, thus (unlike in our earlier paper) including all regular expressions. We do have global restrictions in the syntax. A "pairing" condition identifies synchronizations which will take place at run-time.

Our products translate easily into labelled free choice nets. The converse from nets to products shown in [Pha14a] requires a distributed choice property to give an FC-matching product. This is a generalization of the property that in a synchronization cluster of a free choice net all the synchronizations are differently labelled (call this "deterministic synchronization"). In brief we can say the intermediate representation in this paper is related to 1-bounded labelled distributed free choice nets.

The problem of syntactically characterizing a natural subclass of 1-bounded labelled free choice nets, those which can be modelled using deterministic synchronization, is not fully solved. Consider a "distributed counter" product system which we denote by the informal expression $fsync((a^2)^*, (a^3)^*)$, for which no "pairing" in our sense exists. There is an intuitively corresponding free choice net which has deterministic synchronization, by pairing three iterations of the loop on the left with two iterations of the loop on the right. Applying our theorems to such an unfolded net produces a product system with matching and then the expression $fsync((a^6)^*, (a^6)^*)$ with obvious pairing, defining the same language. Thus our syntax is expressive but our pairings are deficient: they have to be relaxed to also incorporate distributed counting of this kind. We conjecture this can be done.

Ths paper is organized as follows. In the next section, we introduce some properties of regular expression using derivatives. In Section 3, we give syntax of connected expressions and define some properties in terms of properties of regular expressions defined earlier, and derivatives of connected expressions. In subsequent section, we formally define the subclass of product systems we work with, and discuss its properties. In Section 5, we present main results of this paper: two way conversion between connected expressions and product systems defined earlier. In the next section, we outline how combining the results of this paper with those in [Pha14a] gives Kleene theorems for a subclass of free choice net systems.

## 2 Regular Expressions, Derivatives and States

Let $\Sigma$ be a finite alphabet and $\Sigma^*$ be the set of all words over alphabet $\Sigma$, including the empty word $\varepsilon$. A language over an alphabet $\Sigma$ is a subset $L \subseteq \Sigma^*$.

The projection of a word $w \in \Sigma^*$ to a set $\Delta \subseteq \Sigma$, denoted as $w{\downarrow_\Delta}$, is defined by:

$$\varepsilon{\downarrow_\Delta} = \varepsilon \text{ and } (a\sigma){\downarrow_\Delta} = \begin{cases} a(\sigma{\downarrow_\Delta}) & \text{if } a \in \Delta, \\ \sigma{\downarrow_\Delta} & \text{if } a \notin \Delta. \end{cases}$$

**Definition 1.** *Let Loc denote the set $\{1, 2, \ldots, k\}$. A distribution of $\Sigma$ over Loc is a tuple of nonempty sets $(\Sigma_1, \Sigma_2, \ldots, \Sigma_k)$ with $\Sigma = \bigcup_{1 \le i \le k} \Sigma_i$. For each action $a \in \Sigma$, its locations are the set $loc(a) = \{i \mid a \in \Sigma_i\}$. Actions $a \in \Sigma$ such that $|loc(a)| = 1$ are called local, otherwise they are called global.*

A regular expression over alphabet $\Sigma_i$ such that constants 0 and 1 are not in $\Sigma_i$ is given by:

$$s ::= 0 \mid 1 \mid a \in \Sigma_i \mid s_1 \cdot s_2 \mid s_1 + s_2 \mid s_1^*$$

The languages defined are $Lang(0) = \emptyset$, $Lang(1) = \{\varepsilon\}$ and $Lang(a) = \{a\}$. For regular expressions $s_1 + s_2$, $s_1 \cdot s_2$ and $s_1^*$, the languages are defined inductively as union, concatenation and Kleene star of of the component languages respectively.

As a measure of the size of an expression we will use $wd(s)$ for its alphabetic width—the total number of occurrences of letters of $\Sigma$ in $s$. We will use syntactic entities associated with regular expressions which are known since the time of Brzozowski [Brz64], Mirkin [Mir66] and Antimirov [Ant96]. For each regular expression $s$ over $\Sigma_i$, its initial actions form the set $Init(s) = \{a \mid \exists v \in \Sigma_i^* \text{ and } av \in Lang(s)\ \}$ which can be defined syntactically. (For a set of expressions $D$, $Init(D)$ collects the initial actions of members of $D$.) Similarly, we can syntactically check whether the empty word $\varepsilon \in Lang(s)$. Below we inductively define Antimirov derivatives [Ant96].

**Definition 2.** *Given regular expression $s$ and symbol $a$, the set of partial derivatives of $s$ wrt $a$, written $\mathrm{Der}_a(s)$ are defined as follows.*

$$\begin{aligned}
\mathrm{Der}_a(0) &= \emptyset \\
\mathrm{Der}_a(1) &= \emptyset \\
\mathrm{Der}_a(b) &= \{1\} \text{ if } b = a, \quad \emptyset \text{ otherwise} \\
\mathrm{Der}_a(s_1 + s_2) &= \mathrm{Der}_a(s_1) \cup \mathrm{Der}_a(s_2) \\
\mathrm{Der}_a(s_1^*) &= \mathrm{Der}_a(s_1) \cdot s_1^* \\
\mathrm{Der}_a(s_1 \cdot s_2) &= \begin{cases} \mathrm{Der}_a(s_1) \cdot s_2 \cup \mathrm{Der}_a(s_2) & \text{if } \varepsilon \in \mathrm{Lang}(s_1) \\ \mathrm{Der}_a(s_1) \cdot s_2 & \text{otherwise} \end{cases}
\end{aligned}$$

*Inductively* $\mathrm{Der}_{aw}(s) = \mathrm{Der}_w(\mathrm{Der}_a(s))$.

*The set of all partial derivatives* $\mathrm{Der}(s) = \bigcup_{w \in \Sigma_i^*} \mathrm{Der}_w(s)$, *where* $\mathrm{Der}_\varepsilon(s) = \{s\}$.

*A derivative $d$ of $s$ with global $a \in \mathrm{Init}(d)$ is called an $a$-**site** of $s$. Expression $s$ is said to have **equal choice** if for all $a$, all its $a$-sites have the same set of initial actions.*

The Antimirov derivatives are $Der_a(ab + ac) = \{b, c\}$ and $Der_a(a(b + c)) = \{b + c\}$, whereas the Brzozowski $a$-derivative [Brz64] (which is used for constructing deterministic automata, but which we do not use in this paper) for both expressions would be $\{b + c\}$.

*Example 1.* Consider a regular expression $r = a(b + c)d(b + c)^*$. The set of its derivatives is $Der(r) = \{r, (b + c)d(b + c)^*, d(b + c)^*, (b + c)^*\}$. For derivative $(b + c)d(b + c)^*$ of $r$, its set of initial actions is $Init((b + c)d(b + c)^*) = \{b, c\}$. Therefore, derivative $(b+c)d(b+c)^*$ is a $b$-site and a $c$-site but it is not an $a$-site. For $b$-site $(b + c)^*$ of $r$, its set of initial actions is $Init((b + c)^*) = \{b, c\}$. Sets of initial actions for all $b$-sites of $r$ are equal, and this is true for all $c$-sites and $a$-sites. Therefore, expression $r$ has equal choice property.

Now consider another regular expression $r' = a(b + c)d(b + e)^*$. The set of its $b$-sites is $\{(b+c)d(b+e)^*, (b+e)^*\}$. For $b$-site $(b+c)d(b+e)^*$ of $r'$, its set of initial actions is $Init((b+c)d(b+e)^*) = \{b, c\}$. For $b$-site $(b+e)^*$ of $r'$, its set of initial actions is $Init((b+e)^*) = \{b, e\}$. Since sets of initial actions are not equal for these two $b$-sites, expression $r'$ does not have equal choice property.

We wish to club together derivatives which may correspond to the same state in a finite automaton. For this we use partitions of the set of derivatives of expression $s$. If for every global action $a$, the partition of $a$-sites of $s$ consists of a single block, then we say $s$ has **unique sites**. We syntactically determine a partition of the $a$-sites of $s$. This kind of idea appears in the work of Lombardy and Sakarovitch [LS05,LS10]. We go on to define a semantic property, which first appeared in the conference version of this paper [PL14]. Later in this paper we will consider coarsenings of this partition.

**Definition 3.** *We define an equivalence relation $\sim_a$ between a-sites, given by $s_1 \sim_a s_1 + s_2$, $s_2 \sim_a s_1 + s_2$, $s_1 \cdot s_2 \sim_a s_2$ in case $\varepsilon \in \mathrm{Lang}(s_1)$, $s_1 \cdot s_1^* \sim_a s_1^*$ and one of the sides is not an a-derivative of the other, and also $s_1^* \cdot s_1^* \sim_a s_1^*$. The partition defined on the a-sites of s is denoted $\mathrm{Part}_a(s)$.*

*Example 2.* For expression $aa$ the partition of $a$-sites is: $Part_a(aa) = \{\{aa\}, \{a\}\}$. For expression $b$ it is $Part_a(b) = \emptyset$. The $a$-sites of expression $aa + b$ can be partitioned by this representation: $Part_a(aa + b) = \{\{aa + b\}, \{a\}\}$. The $a$-sites of expression $(aa + b)^*aa$ are: $Part_a((aa + b)^*aa) = \{\{(aa + b)^*aa\}, \{a(aa + b)^*aa\}, \{a\}\}$. Finally, the $a$-sites of $a^*(aa+b)^*aa$ are described by the partition: $Part_a(a^*(aa + b)^*aa) = \{\{a^*(aa + b)^*aa\}, \{a(aa + b)^*aa\}, \{(aa + b)^*aa\}, \{a\}\}$. Here we do not have $a^*(aa + b)^*aa \sim_a (aa + b)^*aa$ even though $\varepsilon \in Lang(a^*)$ because the right hand expression is an $a$-derivative of the left hand one.

**Definition 4.** *Given a set D of a-sites of regular expression s, an action a and a language L, we define the relativized language $L^D = \{xay \mid xay \in L, \exists d \in \mathrm{Der}_x(s) \cap D, \exists d' \in \mathrm{Der}_{ay}(d)$ with $\varepsilon \in \mathrm{Lang}(d')\}$, and the prefixes $\mathrm{Pref}_a^D(L) = \{x \mid xay \in L^D\}$, and the suffixes $\mathrm{Suf}_a^D(L) = \{y \mid xay \in L^D\}$. We say that the derivatives in set D a-**bifurcate** L if $L^D = \mathrm{Pref}_a^D(L)\ a\ \mathrm{Suf}_a^D(L)$. (The left to right direction always holds.)*

*Example 3.* Let $L = Lang((aa)^*) = \{(aa)^k \mid k \geq 0\}$. Then $L^{(aa)^*} = L^{a(aa)^*} = \{(aa)^k \mid k \geq 1\}$. Hence we have, $Pref_a^{a(aa)^*}(L) = \{a^{2k} \mid k \geq 0\} = Suf_a^{(aa)^*}(L)$ and $Suf_a^{a(aa)^*}(L) = \{a^{2k+1} \mid k \geq 0\} = Pref_a^{(aa)^*}(L)$. The derivatives $(aa)^*$ and $a(aa)^*$ both $a$-bifurcate $L$, but the set $D = \{(aa)^*, a(aa)^*\}$ does not, as $a^2 \in Pref_a^{a(aa)^*}(L)$, and $a^2 \in Suf_a^{(aa)^*}(L)$, but $a^2aa^2 \notin L^D$.

**Proposition 1.** *Every block $D$ of the partition $Part_a(s)$ $a$-bifurcates $Lang(s)$.*

*Proof.* Let $L = Lang(s), x \in Pref_a^D(L), y \in Suf_a^D(L)$. We have to show that $xay \in L$, for which we use induction on $s$. The base case $s = a$ is easy as there is only one $a$-derivative. Below we use $D[s/r]$ to mean the derivatives in $D$ where the expression $r$ is replaced by the expression $s$.

**(Case** $s = s_1 + s_2$**):** In the case that $D[s_1/s_1 + s_2]$ was from $Part_a(s_1)$ or $D[s_2/s_1 + s_2]$ was from $Part_a(s_2)$, then from the induction hypothesis $D$ $a$-bifurcates $Lang(s_1)$ or $Lang(s_2)$ and hence also $Lang(s_1 + s_2)$. The remaining case is where $s \in D$, but since $s_1 + s_2$ cannot be an $a$-derivative of itself, this requires that either $s_1 \sim_a s$ or $s_2 \sim_a s$ and we can again apply the induction hypothesis.

**(Case** $s = s_1 \cdot s_2$**):** If $D = D_1 \cdot s_2 \in Part_a(s_1) \cdot s_2$, then $y$ factorizes as $y_1 y_2$ with $y_2 \in Lang(s_2)$ and we use the induction hypothesis to show $xay_1$ in $Lang(s_1)$. If $D \in Part_a(s_2)$ then $x$ factorizes as $x_1 x_2$ with $x_1 \in Lang(s_1)$ and we use the induction hypothesis to show $x_2 ay$ in $Lang(s_2)$. With $s_1 s_2 \in D$ we can have both the conditions

- $x \in Pref_a^{s_1 s_2}(L) \setminus Pref_a^{s_1}(L)$, this implies $x \in Lang(s_1)$, and
- $y \in Suf_a^D(L) \cap Suf_a^{s_2}(Lang(s_2))$, this implies $\varepsilon \in Pref_a^{s_2}(Lang(s_2))$.

Induction hypothesis, applied to the block $D[s_2/s_1 s_2]$ of $Part_a(s_2)$ (because under these conditions $s_2 \sim_a s_1 s_2$), gives $ay$ in $Lang(s_2)$. Since $\varepsilon \in Lang(s_1)$, $ay$ is in $Lang(s_1 s_2)$. So $xay \in L^{s_1 s_2} \subseteq L^D$.

**(Case** $s = s_1^*$**):** If $D = D_1 \cdot s_1^* \in Part_a(s_1) \cdot s_1^*$, then $xay$ factorizes as $x_1 x_2 a y_1 y_2$ with $x_1, y_2 \in Lang(s_1^*), x_2 \in Pref_a^{D_1}(Lang(s_1)), y_1 \in Suf_a^{D_1}(Lang(s_1))$ and we use the induction hypothesis to show $x_2 a y_1$ is in $Lang(s_1)$. The remaining case is where $x \in Lang(s_1^*)$ and $ay \in Lang(s_1^*)$, we deal with this as in the previous case using the conditions $s_1 \cdot s_1^* \sim_a s_1^*$ and $s_1^* \cdot s_1^* \sim s_1^*$. ☐

## 3 Connected Expressions over a Distribution

We have a simple syntax of connected expressions. The component expression $s_i$ can be any regular expression (of any star-height), which is different from our earlier paper [LMP11]. A connected expression is given in the form:

$$e ::= fsync(s_1, s_2, \ldots, s_k), \quad s_i \text{ defined over } \Sigma_i$$

When $e = fsync(s_1, s_2, \ldots, s_k)$ and $I \subseteq \Sigma$, let the projection $e{\downarrow}I = \Pi_{i \in I} s_i$.

For the connected expression $e = fsync(s_1, s_2, \ldots, s_k)$, its language is given by

$$Lang(e) = Lang(s_1) \| Lang(s_2) \| \ldots \| Lang(s_k),$$

where the synchronized shuffle $L = L_1 \| \ldots \| L_k$ is defined by

$$w \in L \text{ iff for all } i \in \{1, \ldots, k\}, w{\downarrow}_{\Sigma_i} \in L_i.$$

The definitions of derivatives can be easily extended to connected expressions. Given $e = fsync(s_1, s_2, \ldots, s_k)$, its derivatives are defined using the derivatives of the $s_i$ on action $a$:

$$Der_a(e) = \{fsync(r_1, r_2, \ldots, r_k) \mid \forall i \in loc(a),\ r_i \in Der_a(s_i);\ \text{otherwise } r_j = s_j\}.$$

We will use the word derivative for expressions such as $d = fsync(r_1, r_2, \ldots, r_k)$ above (essentially tuples of derivatives of regular expressions), and $d[i]$ for $r_i$. The number of derivatives can be exponential in $k$. Define $Init(d)$ to be those actions $a$ such that $Der_a(d)$ is nonempty. If $a \in Init(d)$ we call $d$ an **$a$-site**. A connected expression has **unique sites** if each of its component regular expressions has the unique sites property. The reachable derivatives are $Der(e) = \{d \mid d \in Der_x(e), x \in \Sigma^*\}$. For example, $fsync(ab, ba)$ does not have reachable derivatives other than itself.

### 3.1 Pairings of Connected Expressions

In our syntax, along with connected expressions we are supplied with "pairings" which specify which part of a component in a connected expression will intersect with which part of another component in the expression. This is done using derivatives, more precisely partitions of derivatives, since being in a state of an automaton corresponds to any one of the blocks in the partition. This machinery is to be set up now.

**Definition 5.** *Let $e = fsync(s_1, s_2, \ldots, s_k)$ be a connected expression over $\Sigma$. For a global action $a$ and given partitions of derivatives of each $s_i$,* ***pairing(a)*** *is a subset of tuples of $\Pi_{i \in loc(a)} Der(s_i)$ which respects the partitions. (For instance, if the partitions are $Part_a(s_i)$ for $\Sigma_i$, pairing$(a)$ is a subset of tuples such that the projection of these tuples includes all the blocks of $Part_a(s_i)$, and if a block of $Part_a(s_j), j \in loc(a)$ appears in one tuple of the pairing, it does not appear in another tuple.) We call pairing$(a)$ equal choice if for every tuple in the pairing, the blocks of derivatives in the tuple have equal choice.*
*We extend the definition to connected expressions. A derivative $fsync(r_1, \ldots, r_k)$ is in pairing(a) if there is a tuple $D \in$ pairing$(a)$ such that $r_i \in D[i]$ for all $i \in loc(a)$. For convenience we may write a derivative as an element of pairing$(a)$. Expression $e$ is said to have* ***(equal choice) pairing of actions*** *if for all global actions $a$, there exists an (equal choice) pairing$(a)$. Expression $e$ is said to be* ***consistent with a pairing of actions*** *if every reachable $a$-site $d \in Der(e)$ is in pairing$(a)$.*

*Example 4.* Consider a distribution $\Sigma_1 = \Sigma_2 = \{a\}$ and a connected expression $fsync(aa, a)$ defined over it. The partition for $aa$ over $\Sigma_1$ is $Part_a(aa) = \{\{aa\}, \{a\}\}$ and for the expression $a$ over $\Sigma_2$ is $Part_a(a) = \{\{a\}\}$. Since two blocks of $Part_a(aa)$ cannot be paired with one block of $Part_a(a)$, expression $fsync(aa, a)$ does not have a pairing. Since there are two blocks in the partition $Part_a(aa)$, expression $aa$ does not have unique sites property, neither does $fsync(aa, a)$.

*Example 5.* Consider a connected expression $e = fsync(aa, bad + caf)$ over the distribution $(\Sigma_1 = \{a\}, \Sigma_2 = \{a, b, c, d, f\})$. For action $a$, The partition over $\Sigma_1$ is $Part_a(aa) = \{\{aa\}, \{a\}\}$ and the partition over $\Sigma_2$ is $Part_a(bad + caf) = \{\{ad\}, \{af\}\}$. The $a$-sites of expression $e$ are $\{fsync(aa, ad), fsync(aa, af)\}$. There are two possible pairings for action $a$: one is $\{(\{aa\}, \{af\}), (\{a\}, \{ad\})\}$ and another is $\{(\{aa\}, \{ad\}), (\{a\}, \{af\})\}$. The derivative $aa$ on the left appears in the pairing with two different reachable $a$-sites of the right hand side, which belong to two different blocks of $Part_a(bad+caf)$. Hence $e$ is not consistent with respect to any of the above pairings.

*Example 6.* Consider the connected expression $fsync(r_1, r_2, r_3)$ with $r_1 = (ac)^*$, $r_2 = (bc)^*$ and $r_3 = (a(b + c))^*$ over the distribution $(\Sigma_1 = \{a, c\}, \Sigma_2 = \{b, c\}, \Sigma_3 = \{a, b, c\})$. Now we have $r'_1 = Der_a(r_1) = c(ac)^*$ and $Init(r'_1) = \{c\}$. For $r_3$ we have, $r'_3 = Der_a(r_3) = (b + c)(a(b + c))^*$ and $Init(r'_3) = \{b, c\}$. Expressions $r'_1$ and $r'_3$ are $c$-sites of expressions $r_1$ and $r_3$ respectively. With sets of derivatives $D_1 = \{r'_1\}$ and $D_3 = \{r'_3\}$ as the only blocks in the respective partitions of $c$-sites i.e., $Part_c(r_1) = \{D_1\}$ and $Part_c(r_3) = \{D_3\}$. As $Init(D_1) = Init(r'_1) = \{c\}$, $Init(D_3) = Init(r'_3) = \{b, c\}$ and $pairing(c) = \{(D_1, D_3)\}$, $pairing(c)$ is not equal choice. Therefore, connected expression $e$ does not have equal choice. However one can see that $e$ has unique sites property.

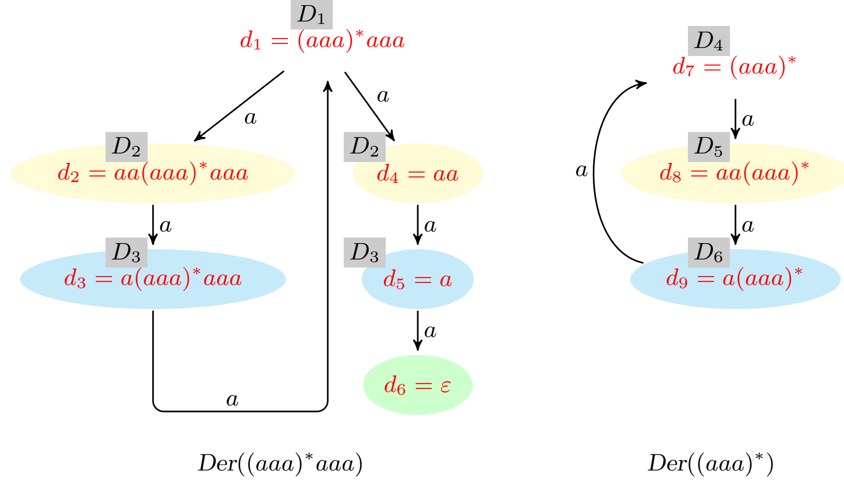


**Fig. 1.** Derivatives of $d_1$ and $d_7$ of expression $e = fsync(d_1, d_7)$ with $pairing(a) = \{(D_1, D_4), (D_2, D_5), (D_3, D_6)\}$.

*Example 7.* Consider a connected expression $e = fsync((aaa)^*aaa, (aaa)^*)$. The $a$-derivatives are $Der_a(e) = \{fsync(aa(aaa)^*aaa, aa(aaa)^*), fsync(aa, aa(aaa)^*)\}$.

With respect to word $aa$, $Der_{aa}(e) = \{fsync(a(aaa)^*aaa, a(aaa)^*), fsync(a, a(aaa)^*)\}$. With respect to word $aaa$, $Der_{aaa}(e) = \{fsync((aaa)^*aaa, (aaa)^*), fsync(\varepsilon, (aaa)^*)\}$. The language of connected expression $e$ is $Lang(e) = \{(aaa)^k \mid k \geq 1\}$. See Figure 1 where derivatives of $d_1 = (aaa)^*aaa$ and $d_7 = (aaa)^*$ are shown. the set of derivatives of $e = fsync(d_1, d_7)$, with respect to all words $w \in \Sigma^*$: $Der(e) = \{(d_1, d_7), (d_2, d_8), (d_4, d_8), (d_3, d_9), (d_5, d_9), (d_6, d_7)\}$ and, its set of $a$-sites is $\{(d_1, d_7), (d_2, d_8), (d_4, d_8), (d_3, d_9), (d_5, d_9)\}$.

Let $D_1, D_2, D_3$ be sets of $a$-sites for expressions $d_1$ where, $D_1 = \{d_1\}$, $D_2 = \{d_2, d_4\}$, and $D_3 = \{d_3, d_5\}$. And let $D_4, D_5, D_6$ be sets of $a$-sites for expressions $d_2$ where, $D_4 = \{d_7\}$, $D_5 = \{d_8\}$ and $D_6 = \{d_9\}$. For expression $d_1$, $Part_a(d_1) = \{D_1, D_2, D_3\}$ and for $d_2$, $Part_a(d_2) = \{D_4, D_5, D_6\}$. For action $a$, we have a pairing relation $pairing(a) = \{(D_1, D_4), (D_2, D_5), (D_3, D_6)\}$. We can see that expression has equal choice property and it is consistent with pairing of actions.

**Proposition 2.** *For a connected expression checking existence of a pairing of actions and checking whether it is equal choice can be done in polynomial time, checking consistency with a pairing of actions is in* PSPACE.

*Proof.* We have to visit each derivative of all the regular expressions to construct the $a$-partitions for every $a$. We can record their initial actions. Maximum number of Antimirov derivatives of any regular expression $s$ is at most $wd(s) + 1$ [Ant96]. If the number of blocks in two $a$-partitions is not the same, there cannot be a $pairing(a)$, otherwise there always exists a pairing($a$). For an equal choice pairing, we have to count blocks whose sets of initial actions are the same, this can be done in cubic time. On the other hand, to check consistency with a pairing of actions, we have to visit each reachable derivative, this can be done in PSPACE. □

# 4 Product Systems over a Distribution

Fix a distribution $(\Sigma_1, \Sigma_2, \ldots, \Sigma_k)$ of $\Sigma$. We now define an automaton over some alphabet $\Sigma_i$.

**Definition 6.** *A* sequential system *over a set of actions $\Sigma_i$ is a tuple $A_i = \langle P_i, \to_i, G_i, p_i^0 \rangle$ where $P_i$ are called* ***places****, $G_i \subseteq P_i$ are final places, $p_i^0 \in P_i$ is the initial place, and $\to_i \subseteq P_i \times \Sigma_i \times P_i$ is a set of* ***local moves****.*

Let $\to_a^i$ denote the set of all $a$-labelled moves in the sequential system $A_i$.

For a local move $t = \langle p, a, p' \rangle$ of $\to_i$, $p$ is called pre-place and $p'$ is called post-place of $t$. A run of the sequential system $A_i$ on word $w$ is a sequence $p_0 a_1 p_1 a_2, \ldots, a_n p_n$, from set $(P_i \times \Sigma_i)^* P_i$, such that $p_0 = p_i^0$ and for each $j \in \{1, \ldots, n\}$, $p_{j-1} \xrightarrow{a_j} p_j$. This run is said to be accepting if $p_n \in G_i$. The sequential system $A_i$ accepts word $w$, if there is at least one accepting run of $A_i$ on $w$. The language $L = Lang(A_i)$ of sequential system $A_i$ is defined as $L = \{w \in {\Sigma_i}^* | w \text{ is accepted by } A_i\}$.

Given a place $p$ of $A_i$, we also define relativized languages and we will extend this definition to product systems: $Pref^p_a(L) = \{x \mid xay \in L, p_0 \xrightarrow{x} p \xrightarrow{ay} G_i\}$, similarly $Suf^p_a(L)$, $L^p = \{xay \mid xay \in L, p_0 \xrightarrow{x} p \xrightarrow{ay} G_i\}$. Say the place $p$ **$a$-bifurcates** $L$ if $L^p = Pref^p_a(L)\ a\ Suf^p_a(L)$.

We now define products of automaton.

**Definition 7.** *Let $A_i = \langle P_i, \rightarrow_i, G_i, p^0_i \rangle$ be a sequential system over alphabet $\Sigma_i$ for $1 \leq i \leq k$. A* ***product system*** *$A$ over the distribution $\Sigma = (\Sigma_1, \dots, \Sigma_k)$ is a tuple $\langle A_1, \dots, A_k \rangle$.*

Let $\Pi_{i \in Loc} P_i$ be the set of product states of $A$. We use $R[i]$ for the projection of a product state $R$ in $A_i$, and $R{\downarrow}I$ for the projection to $I \subseteq Loc$. The relativizations $L^R$ of a language $L \subseteq \Sigma_i^*$ consider projections to place $R[i]$ in $A_i$.

The initial product state of $A$ is $R^0 = (p^0_1, \dots, p^0_k)$, while $G = \Pi_{i \in Loc} G_i$ denotes the final states of $A$.

Let $\Rightarrow_a = \Pi_{i \in loc(a)} \rightarrow^i_a$. The set of global moves of $A$ is $\Rightarrow = \bigcup_{a \in \Sigma} \Rightarrow_a$. Then for a global move

$$g = \langle \langle p_{l_1}, a, p'_{l_1} \rangle, \langle p_{l_2}, a, p'_{l_2} \rangle, \dots \langle p_{l_m}, a, p'_{l_m} \rangle \rangle \in \Rightarrow_a,\ loc(a) = \{l_1, l_2, \dots, l_m\},$$

we write $g[i]$ for $\langle p_i, a, p'_i \rangle$, the projection to $A_i$, $i \in loc(a)$ and $pre(a)$ for the product states where such a move is enabled.

Please note that the set of product states as well as the global moves are not explicitly provided when a product system is given as input to some algorithm.

### 4.1 Matchings of Product Systems

Analogously to what we did for expressions, we now set up "matching" relations between places in different components of a product system which correspond to pre-places of a global move. ("Separation" below is a stronger property.) Thus matchings restrict the possible synchronizations in a product, an idea developed for transition systems by Arnold and Nivat [Arn94]. That the restriction involves places is the key to translation into clusters of labelled free choice nets, which is outlined in Section 6.

**Definition 8.** *For global $a \in \Sigma$, matching(a) is a subset of tuples $\Pi_{i \in loc(a)} P_i$ such that for all $i$ in $loc(a)$, projection of these tuples is the set of all pre-places of $a$-moves in $\rightarrow^a_i$, and if a place $p \in P_i$ appears in one tuple, it does not appear in another tuple. We say a product state $R$ is in matching(a) if its projection $R{\downarrow}loc(a)$ is in the matching.*

*A product system is said to have* ***matching of labels*** *if for all global $a \in \Sigma$, there is a suitable* matching(a)*. A product system $A$ is said to have* ***separation of labels*** *if for all $i \in Loc$, and for all global actions $a$, if $\langle p, a, p' \rangle, \langle q, a, q' \rangle \in \rightarrow_i$ then $p = q$.*

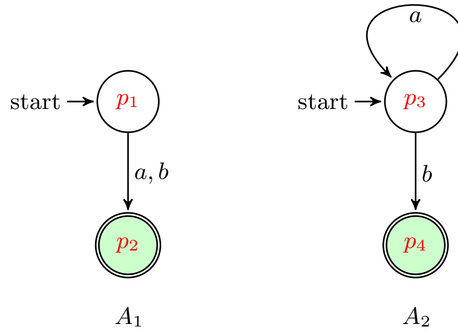


**Fig. 2.** Product system $A = (A_1, A_2)$ with separation of labels

*Example 8.* Let $\Sigma = \{a, b\}$ be a distributed alphabet with distribution ($\Sigma_1 = \Sigma_2 = \Sigma$). Consider the product system $A = (A_1, A_2)$ shown in Figure 2. For global action $a$, place $p_1$ is the only place in $A_1$ having outgoing $a$-moves and, place $p_3$ is the only place in $A_2$ having outgoing $a$-moves.

Similarly these are the only places, in respective sequential systems, which have outgoing local $b$-moves. Therefore, product system $A$ has separation of labels property.

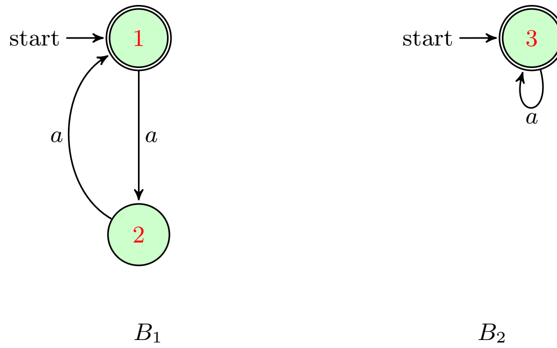


**Fig. 3.** Product system $B = (B_1, B_2)$ without separation of labels

On the other hand, consider product system $B = (B_1, B_2)$ shown in Figure 3, and defined over the distributed alphabet $\Sigma' = \{a\}$ having distribution $\Sigma'_1 = \Sigma'_2 = \Sigma'$. Since sequential system $B_1$ has more than one place having outgoing $a$-moves, product system $B$ does not have the separation of labels property.

**Proposition 3.** *Let $A = \langle A_1, \ldots, A_k \rangle$ be a product system over distribution $\Sigma = (\Sigma_1, \ldots, \Sigma_k)$. If $A$ has separation of labels, then for every $i$ and every global action $a$, $L_i = \text{Lang}(A_i)$ is $a$-bifurcated. If $A$ has matching of labels, then for every $i$ and every global action $a$,*

$$L_i \cap \Sigma_i^* a \Sigma_i^* = \bigcup_{R \downarrow loc(a) \in matching(a)} Pref_a^{R[i]}(L_i)\ a\ Suf_a^{R[i]}(L_i).$$

*Proof.* Let $A$ be a product system as above with separation of labels. Let $L(q)$ be the set of words accepted starting from any place $q$ in $A_i$. If $Pref_a(L(q))$ is nonempty then $L(q)$ is $a$-bifurcated, because the words containing $a$ have to pass through a unique place. When $A$ has a matching of labels, since the places $R[i]$ appear in unique tuples, one can separately consider the places $a$-bifurcating $L(q)$ and the required property follows. ☐

The next property is necessary for product systems to represent free choice in equivalent nets. In our earlier paper [LMP11] we used the definition of an FC-product. The definition of FC-matching product is a generalization since conflict-equivalence is not required for all $a$-moves uniformly but refined into smaller equivalence classes depending on the matching.

**Definition 9.** *In a product system, we say the local move $\langle p, a, q_1 \rangle \in \to_i$ is **conflict-equivalent** to the local move $\langle p', a, q_1' \rangle \in \to_j$, if for every other local move $\langle p, b, q_2 \rangle \in \to_i$, there is a local move $\langle p', b, q_2' \rangle \in \to_j$ and, conversely, for moves from $p'$ there are corresponding outgoing moves from $p$. For global action $a$, its matching($a$) is called **conflict-equivalent matching**, if whenever $p, p'$ are related by the matching($a$), their outgoing local $a$-moves are conflict-equivalent.*

*We call $A = \langle A_1, \ldots, A_k \rangle$ an **FC-product** if for every global action $a \in \Sigma$, and for all $i, j \in loc(a)$, every $a$-move in $A_i$ is conflict-equivalent to every $a$-move in $A_j$ and we call $A$ an **FC-matching product** if it has a conflict-equivalent matching($a$).*

A system having a conflict-equivalent matching is a weaker condition than the system being FC-product.

*Example 9.* Let $\Sigma = \{a, b, c\}$ be a distributed alphabet with distribution ($\Sigma_1 = \Sigma_2 = \Sigma$). Consider the product system $A = (A_1, A_2)$ shown in Figure 4. The matching relations are: $matching(b) = \{(1,4)\}$, $matching(a) = \{(2,5),(1,4)\}$ and $matching(c) = \{(2,5)\}$.

The local move $\langle p_1, a, p_2 \rangle \in \to_1$ in $A_1$ is conflict-equivalent with local move $\langle p_4, a, p_5 \rangle \in \to_2$, but it is not conflict-equivalent with local move $\langle p_5, a, p_7 \rangle \in \to_2$.

For global action $a$, consider places $p_1$ and $p_4$ which appear in a tuple of $matching(a)$, they have all their outgoing moves conflict-equivalent with each other. This is true for places $p_2$ and $p_5$ as well. Hence, $matching(a)$ is conflict-equivalent. In fact, $matching(b)$ and $matching(c)$ are also conflict-equivalent.

Since local move $\langle p_1, a, p_2 \rangle \in \to_1$ is not conflict equivalent with local move $\langle p_5, a, p_7 \rangle \in \to_2$, for global action $a$, not all local $a$-moves are conflict-equivalent to each other. Therefore, product system $A$ is not an FC-product.

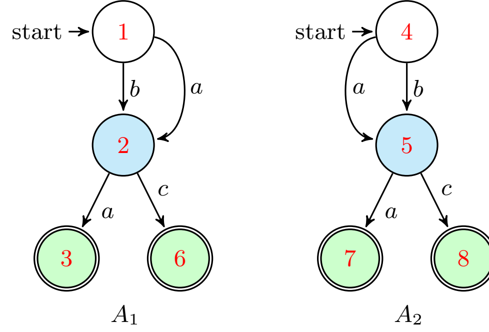


**Fig. 4.** Product system with matching of labels

**Proposition 4.** *For product system $A$, checking if it has separation of labels, and if it has matching of labels, can be done in* PTIME.

*Proof.* To check if $A$ has separation of labels, we visit all local moves of each $A_i$ once for all $i$ in $\{1, \ldots, k\}$, to make sure that for each global action $a$, all $a$-moves of $\rightarrow_i$ have same pre-place. This takes time linear in the size of $A$.

For a global action $a \in \Sigma$, to check if $matching(a)$ exists, we need to visit each place $p$ of $A_i$, for all $i$ in $loc(a)$, to count how many places have outgoing $a$-moves. If this count is same for each i in $loc(a)$, then a $matching(a)$ exists. We repeat above step for each global $a$. So we need to visit each place of $A$ at most $|\Sigma|$ times. Therefore, the total time needed to check if matching of labels exist for product system $A$ is $O(|\Sigma||A|)$.

**Proposition 5.** *Let $A$ be an FC-matching product system. For any $i$, if there exist local moves $\langle p, a, p' \rangle, \langle p, b, p'' \rangle$ in $\rightarrow_i$, then $loc(a) = loc(b)$.*

*Proof.* Since $p$ has an outgoing $a$-move, $p$ belongs to some tuple of $matching(a)$. If $j \in loc(a)$, then in this tuple there exists a place $q \in P_j$, which has an outgoing $a$-move. Since $A$ is an FC-matching product, $matching(a)$ is conflict-equivalent. And, as places $p$ and $q$ appear in a tuple of $matching(a)$, $a$-moves outgoing from $p$ and $q$ are conflict-equivalent. Therefore there exists a local move $(q, b, q') \in \rightarrow_j$. This implies that $j \in loc(b)$. □

### 4.2 Language of a Product System

Now we describe runs of $A$ over some word $w$ by associating product states with prefixes of $w$: the empty word is assigned initial product state $R^0$, and for every prefix $va$ of $w$, if $R$ is the product state reached after $v$ and $Q$ is reached after $va$ where, for all $j \in loc(a), \langle R[j], a, Q[j] \rangle \in \rightarrow_j$ and for all $j \notin loc(a), R[j] = Q[j]$. Let $pre(a) = \{R \mid \exists Q, R \xrightarrow{a} Q\}$.

A run is said to be accepting if the product state reached after $w$ is in $G$. We define the language $Lang(A)$ of product system $A$, as the words on which the product system has an accepting run.

We use the following characterization of direct product languages, which appears in [MR02,Muk11].

**Proposition 6.** *$L = Lang(A)$ is the language of product system $A = \langle A_1, \dots, A_k \rangle$ defined over distributed alphabet $\Sigma$ iff*

$$L = \{w \in \Sigma^* \mid \forall i \in \{1, \dots, k\},\ \exists u_i \in L \text{ such that } w{\downarrow_{\Sigma_i}} = u_i{\downarrow_{\Sigma_i}}\}.$$

*Further $L = Lang(A_1) \| \dots \| Lang(A_k)$.*

The next definition is semantic and not easy to check (we do it in PSPACE). If a system has separation of labels, the property obviously holds.

**Definition 10.** *A run of $A$ is said to be* ***consistent with a matching of labels*** *if for all global actions $a$ and every prefix of the run $R^0 \overset{v}{\Rightarrow} R \overset{a}{\Rightarrow} Q$, the pre-places $R{\downarrow}loc(a)$ are in the matching.*

**Proposition 7.** *For product system $A$ with matching of labels, checking if $A$ is FC-matching product can be done in* PTIME*, and checking if all runs of $A$ are consistent with given matching of labels can be done in* PSPACE*.*

*Proof.* To check if $A$ is FC-product we have to check for each global action $a$, whether $matching(a)$ is conflict-equivalent. Let $(p_1, p_2, \dots, p_m)$ be a tuple in $matching(a)$. For any two places $p_i$ and $p_j$ of this tuple, we have to check if their sets of labels of outgoing local moves are same. This comparison between two sets takes $O(k|\Sigma|)$ time. We need to carry out this step for all tuples in $matching(a)$. This can be done by visiting all local moves of $A_i$, for all $i$ in $loc(a)$ at most once. Therefore, for each global action $a$ in $\Sigma$, we need to visit all local moves of $A$ at most $|\Sigma|$ times. Hence, the total time required is polynomial in the size of $\Sigma$ and $A$.

To check if all runs of $A$ are consistent with given matching of labels we need to visit each reachable global state of $A$ at most once, which can be done in PSPACE.

## 5 Connected Expressions and Product Systems

In this section we prove two main theorems of the paper. To place them in context of our earlier paper [LMP11], there we used a “structural cyclicity” condition which allowed a run to be split into finite parts from the initial product state to itself, since it was guaranteed to be repeated. The new idea in this paper is that runs are split up using matchings which correspond to synchronizations; what happens in between is not relevant for the connections across sequential systems. Hence extending our syntax to allow full regular expressions for the sequential systems does not affect the synchronization properties which are the main issue we are addressing. In Section 6 we outline the connections to labelled free choice nets.

### 5.1 Synthesis of Systems from Expressions

We begin by constructing products of automata for our syntactic entities. For regular expressions, this is well known. We follow the construction of Antimirov, which in polynomial time gives us a finite automaton of size $O(wd(s))$, using partial derivatives as states. Now for connected expressions we need to construct a product of automata.

**Lemma 1.** *Let $e$ be a connected expression with partitions which give unique sites (for every global action). Then there exists a product system $A$ with separation of labels accepting $\mathrm{Lang}(e)$ as its language. If $e$ had equal choice, then $A$ is FC-product.*

*Proof.* Let $e = \mathit{fsync}(s_1, s_2, \ldots, s_k)$. Then for each $s_i$, which is a regular expression defined over some alphabet $\Sigma_i$, we produce a sequential system $A_i$ over $\Sigma_i$, using Antimirov's derivatives, such that $\mathit{Lang}(s_i) = \mathit{Lang}(A_i)$, $\forall\, i \in \{1, \ldots, k\}$. Next we trim it—remove places not reachable from the initial place $p_i^0$ and places from where a final place is not reachable. Now, for each global action $a$, we quotient $A_i$ by merging all derivatives $d$ such that $a \in \mathit{Init}(d)$ into a single place.

Call the resulting automaton $A'_i$. Let $p$ be the merged place in $A'_i$ which is now the source of all $a$-moves. Clearly $\mathit{Lang}(A_i) \subseteq \mathit{Lang}(A'_i)$ since no paths are removed, we show next that the inclusion in the other direction also holds, using the unique sites condition.

Let $a$ be a global action. Consider a word $w = x_1 a x_2 \ldots a x_n$ in $\mathit{Lang}(A'_i)$, where the factors $x_1, x_2, \ldots, x_n$ do not contain the letter $a$. We wish to find derivatives $d_0, d_1, \ldots, d_n$ of $A_i$ such that $d_n$ is a final place and for every $j$ there is a run $d_j \xrightarrow{ax_{j+1}} \ldots \xrightarrow{ax_n} d_n$ of $A_i$ when $j > 0$, and $d_0 \xrightarrow{x_1}\xrightarrow{ax_2} \ldots \xrightarrow{ax_n} d_n$ when $j = 0$, which will show the desired inclusion.

We proceed from $n$ downwards. For any place $d_n$ in $G$ there is a run from $d_n$ on $\varepsilon \in \mathit{Lang}(d_n)$ in $A_i$. Inductively assume we have $d_j$ such that there is a run $d_j \xrightarrow{ax_{j+1}} \ldots \xrightarrow{ax_n} d_n$ of $A_i$, so $x_{j+1} a x_{j+2} \ldots a x_n$ is in $\mathit{Suf}_a(\mathit{Lang}(s_i))$ since $d_j$ is reachable from the initial place. Since there is a run $p \xrightarrow{ax_j} p$ in $A'_i$ there are derivatives $d_{j-1}, c_j$ of $s_j$, such that there is a run $d_{j-1} \xrightarrow{ax_j} c_j$ in $A_i$ (when $j = 1$ we get $d_0 \xrightarrow{x_1} c_1$ by this argument). Since $c_j$ quotients to $p$, it has an $a$-derivative $c$ such that $c$ is in $\mathit{Der}_{ax_j a}(d_{j-1})$ ($\mathit{Der}_{x_0 a}(d_0)$ when $j = 1$). Because $d_{j-1}$ is reachable from the initial place by some $v$ and because some final place is reachable from $c$, $vx_j \in \mathit{Pref}_a(\mathit{Lang}(s_i))$ which is nonempty. By the unique sites condition and Proposition 1, since $x_{j+1} \ldots a x_n$ is in $\mathit{Suf}_a(\mathit{Lang}(s_i))$, $v a x_j a x_{j+1} \ldots a x_n$ is in $\mathit{Lang}(s_i)$ and so $x_j a x_{j+1} \ldots a x_n$ is in $\mathit{Suf}_a(\mathit{Lang}(s_i))$. This means that there is a run from some $d_{j-1}$ on $a x_j a x_{j+1} \ldots a x_n$ ending in a final place $d_n$ of $A_i$. So we have the induction hypothesis restored. If $j = 1$ we get $d_0$ which quotients to $p_0$ and has a run on $w$ to $d_n$ in $G$.

So we get a product system $A' = \langle A'_1, A'_2, \ldots, A'_k \rangle$ defined over $\Sigma$. Because of the quotienting $A'$ has separation of labels. That means for a global action $a$, for $i, j$ in $\mathit{loc}(a)$, sequential machines $A'_i, A'_j$ has only one place which has outgoing local $a$-moves. Let $p_i^a$ be that place in $A'_i$ and let $p_j^a$ be that place

in $A'_j$. On the other hand, since $e$ had unique sites, for a global action $a$ and for $i, j$ in $loc(a)$, expression $s_i$ has only one block $D_i$ in the partition of $a$-sites of $s_i$ and expression $s_j$ has only one block $D_j$ in the partition of $a$-sites of $s_j$. Therefore, all $a$-sites of $s_i$ are in this block $D_i$, and all $a$-sites of $s_j$ are in block $D_j$. Therefore $pairing(a)$ has only one tuple which have $D_i$ and $D_j$ appearing in it. Since $e$ has equal choice property, we have $Init(D_i) = Init(D_j)$. Because of quotienting construction, block $D_i$ corresponds to the place $p_i^a$ in $A'_i$ and block $D_j$ corresponds to the place $p_j^a$ in $A'_j$. So each outgoing local $a$-move of $p_i^a$ is conflict-equivalent to each outgoing local $a$-move of place $p_j^a$.

Now we prove language equivalence of expression $e$ and product system $A'$ constructed from it.

$$\begin{aligned} w \in Lang(e) &\text{ iff } \forall i, w{\downarrow_{\Sigma_i}} \in Lang(s_i), \text{ by definition of synchronized shuffle} \\ &\text{ iff } \forall i, w{\downarrow_{\Sigma_i}} \in Lang(A'_i) \\ &\text{ iff } w \in Lang(A'), \text{ by Proposition 6.} \end{aligned}$$

□

**Theorem 1.** *Let $e = fsync(s_1, \ldots, s_k)$ be a connected expression over a distribution $\Sigma$ with a pairing of actions. Then there exists an FC-matching product system $A$ over $\Sigma$, accepting* Lang$(e)$*. If the pairing was equal choice, the matching is conflict-equivalent. If the expression is consistent with the pairing, all runs of $A$ will be consistent with the matching.*

*Proof.* We first rewrite $e$ to another expression $e'$, construct an automaton $A'$ for $Lang(e')$, and then change it to recover an automaton for $Lang(e)$.

Consider global action $a$ and tuple of blocks $D = \Pi_{i \in loc(a)} D_i$ in $pairing(a)$. By Proposition 1 $D_i$ $a$-bifurcates $Lang(s_i)$. We rename for all $i$ in $loc(a)$, the occurrences of $a$ in $s_i$ which correspond to an $a$ in $Init(D_i)$, by the new letter $a^{D_i}$. This is done for all global actions to obtain from $e$ a new expression $e' = fsync(s'_1, \ldots, s'_k)$ over a distribution $\Sigma'$, where every $s'_i$ now has the unique sites property. For any word $w \in Lang(e)$, there is a well-defined word $w' \in Lang(e')$.

By Lemma 1 we obtain a product system $A'$ with separation of labels for $Lang(e')$. Say $pre(a^D)$ is the pre-place for action $a^D$ in $A'_i$. We change all the $\langle pre(a^D), a^D, q\rangle$ moves to $\langle pre(a^D), a, q\rangle$ in all the $A'_i$ to obtain a product system $A$ over the alphabet $\Sigma$. As $w' \in Lang(e') = Lang(A')$ is well-defined from $w$ and, as the renaming of labels of moves does not remove any paths, $w$ is in $Lang(A)$. Conversely, for every run on $w$ accepted by $A$, because of the separation of labels property, there is a well-defined run on $w'$ with the label of a move appropriately renamed depending on the source state, which is accepted by $A'$, hence $w'$ is in $Lang(e')$. So renaming $w'$ to $w$ gives a word in $Lang(e)$.

Now we refer to the pairing of actions in $e$. This defines for each global action $a$ and tuple of blocks of $a$-sites $D$, a relation between pre-places of $a^D$-moves in different components in the product $A'$. By the separation of labels property of $A'$, the tuples in the relation are disjoint, that is, the relation is functional. So for pre-places of $a$-moves in the product $A$ we have a matching. If the pairing was equal choice, the matching is conflict-equivalent.

If the expression $e$ is consistent with the pairing, all reachable $a$-sites are in the pairing, so we can partition $Lang(e) \cap \Sigma^* a \Sigma^*$ using the partitions in $Part_a(e)$. Letting $D$ range over blocks of connected expressions, each block $D$ contributes a global action $a^D$ in the renaming, so we get an expression $e'$ such that for every global action $a^D$, we have the unique $a$-sites property. Applying Lemma 1, we have the product system $A'$ with separation of labels. By Proposition 3, every $Lang(A'_i)$ is $a^D$-bifurcated, and using the characterization of Proposition 6, $Lang(A') \cap (\Sigma')^* a^D (\Sigma')^* = Pref_{a^D}(Lang(A')) a^D Suf_{a^D}(Lang(A'))$. Since several actions $a^D$ are renamed to $a$ and the corresponding tuples of pre-places are recorded in the matching, by Proposition 3 and Proposition 6:

$$\bigcup_{R \in matching(a)} Pref_a^R(Lang(A))\ a\ Suf_a^R(Lang(A)) \subseteq Lang(A) \cap \Sigma^* a \Sigma^*.$$

But this means that all runs of $A$ are consistent with the matching. □

As an illustration of constructing product system with matching from expression with pairing, using Theorem 1 which employs Lemma 1 in its proof, consider the expression in Example 7, for which we produce a product system as was shown in Example 10.

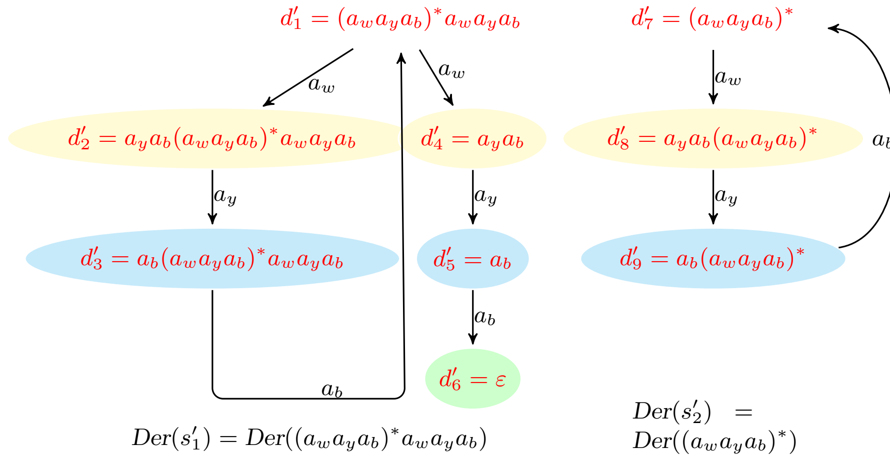


**Fig. 5.** Derivatives of $s'_1$ and $s'_2$ of $e' = fsync(s'_1, s'_2)$ with unique sites property

*Example 10.* As we have seen in Example 7, the pairing relation for expression $e = fsync((aaa)^*aaa, (aaa)^*))$, $pairing(a) = \{(D_1, D_4), (D_2, D_5), (D_3, D_6)\}$. Let $w = (D_1, D_4), y = (D_2, D_5)$ and $b = (D_3, D_6)$.

Then using these tuples, we get a new alphabet $\Sigma' = \{a_w, a_y, a_b\}$ with distribution $\Sigma'_1 = \{a_w, a_y, a_b\}$ and $\Sigma'_2 = \{a_w, a_y, a_b\}$. Each $a$ in $s_i$ belong to

only one block in $Part_a(s_i)$ and that block belong to only one tuple in the $pairing(a)$. Therefore, by renaming each $a$ in $s_i$ by its corresponding tuple in $pairing(a)$, we get $s'_1 = (a_w a_y a_b)^* a_w a_y a_b$ and $s'_2 = (a_w a_y a_b)^* a_w a_r a_b$ over alphabet $\Sigma'_1$ and $\Sigma'_2$ respectively. Hence, we have a connected expression $e'$ over $\Sigma'$ as, $e' = fsync((a_w a_y a_b)^* a_w a_y a_b, (a_w a_y a_b)^*)$.

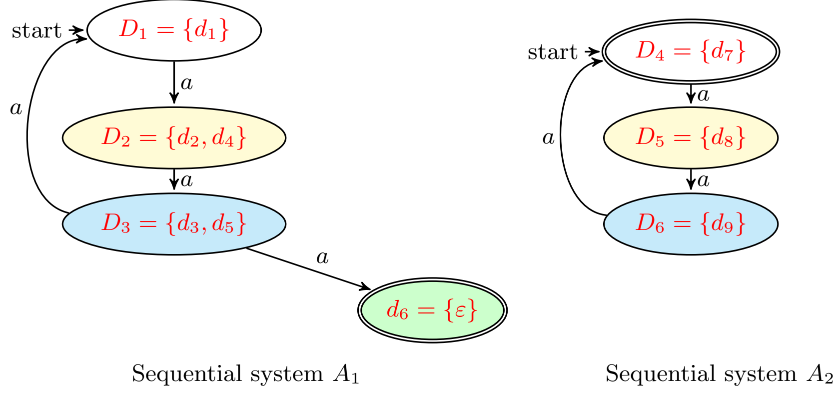


**Fig. 6.** Product system $A = (A_1, A_2)$ with separation of labels

Expressions $s'_1$ and $s'_2$ have unique sites property. In Figure 5, derivatives of $s'_1$ and $s'_2$ are shown. The blocks in the partitions of their respective $a_x$-sites, where $x \in \{w, y, b\}$ are: $D'_1 = \{d'_1\}, D'_2 = \{d'_2, d'_4\}, D'_3 = \{d'_3, d'_5\}, D'_4 = \{d'_7\}, D'_5 = \{d'_5\}, D'_6 = \{d'_6\}$. Now by Lemma 1 we can fuse derivatives in the respective blocks to get product system $A' = (A'_1, A'_2)$, having separation of labels property, and which is language equivalent to expression $e'$. The set of places of sequential system $A'_1$, is $\{D'_1, D'_2, D'_3, d'_6\}$, and of sequential system $A'_2$, is $\{D'_4, D'_5, D'_6\}$. In each $A'_i$ we have only one place which has outgoing $a_x$-moves. So each $a_x$ contributes only one tuple of places in $matching(a)$. Therefore, $matching(a) = \{(D_1, D_4), (D_2, D_5), (D_3, D_6)\}$. The final product system over $\Sigma$ is shown in Figure 6.

### 5.2 Analysis of Expressions from Systems

**Lemma 2.** *Let $A$ be a conflict-equivalent product system with separation of labels. Then we can compute a connected expression for the language of $A$ with partitions of the regular expressions which have unique sites and specified pairings which have equal choice.*

*Proof.* Let $A = \langle A_1, \dots, A_k \rangle$ be a product system with separation of labels, where $A_i$ is a sequential system of $A$ with places $P$, initial place $p_0$ and final places $G$. Kleene's theorem gives us expressions for the words which have runs

from a given state to another using a specified set of states [MY60] and these are put together. Let us suppose that all the states which do not have any global actions enabled are dealt with first. After that we add the states with global actions, we do an induction on the number of these states.

Now we consider a global action $a$. By separation of labels there is a single place $p$ in $A_i$ enabling $a$. Let $Q$ be the states which have already been dealt with and $R = Q \cup \{P\}$. Let $T$ be the set of moves outgoing from $p$ and which are not $a$-moves. Depending on whether we have an $a$-move $p \xrightarrow{a} p$, or $a$-moves $p \xrightarrow{a} p_j,\ p_j \neq p$, or a combination of these two types, we obtain the expression below (where the expressions on the right hand side have already been computed):

$$e^R_{p_0,f} = e^Q_{p_0,f} + e^Q_{p_0,p}(e^Q_{p,p})^* e^Q_{p,f},$$

where the expression $e^Q_{p,p}$ is given by one of the following refinements, for the three cases considered above respectively:

$$(a + e^T_{p,p}),\ \text{or}\ ((\sum_j a e^T_{p_j,p}) + e^T_{p,p}),\ \text{or}\ (a + (\sum_j a e^T_{p_j,p}) + e^T_{p,p}).$$

The superscripts $Q$ and $T$ indicates that these expressions are derived, as in the McNaughton-Yamada construction [MY60], for runs which only use the places $Q$ and, respectively, runs which only use the places $Q$ and moves $T$ (these expressions have already been computed). Whichever be the case, we note that we have an expression with $D^a(e^R_{p_0,f}) = \{(e^Q_{p,p})^* e^Q_{p,f}\}$ as its singleton set of $a$-sites. Therefore, expression $e^R_{p_0,f}$ has the unique $a$-sites property. Since the product system was conflict-equivalent, this argument extends if there are other global actions enabled at state $p$, and the expression obtained is equal choice.

Now consider a global action $c$ enabled at a state $q$ in $Q$. The $c$-sites are obtained from several parts of the expression:

$$D^c(e^R_{p_0,f}) = D^c(e^Q_{p_0,f}) \cup D^c(e^Q_{p_0,p}) \cdot (e^Q_{p,p})^* \cdot e^Q_{p,f} \cup D^c(e^T_{p,p}) \cdot (e^Q_{p,p})^* \cdot e^Q_{p,f} \cup D^c(e^Q_{p,f}).$$

By induction the right hand expressions had the unique $c$-sites property, the $c$-partition collapses all the derivatives above into a single block. We claim the derivatives in this four-way union $c$-bifurcate the language $Lang(e^R_{p_0,f})$. If the state $q$ was visited in only one of the four cases there is nothing to prove. The interesting case is when there is a path from $p$ to $q$ as well as from $q$ to $p$, and separate paths from $p_0$ to $p$ and from $p_0$ to $q$. In this case the second and the third components of the union will both be nonempty. Suppose $w_1 = x_1 c y_1$ with $x_1 \in Lang(e^Q_{p_0,q})$ and $cy_1 \in Lang(e^T_{q,p}(e^Q_{p,p})^* e^Q_{p,f})$, and $w_2 = x_2 c y_2$ with $x_2 \in Lang(e^Q_{p_0,p} e^T_{p,q})$ and $cy_2 \in Lang(e^T_{q,p}(e^Q_{p,p})^* e^Q_{p,f})$. But then $x_1 c y_2$ is in $Lang(e^Q_{p_0,q} e^T_{q,p}(e^Q_{p,p})^* e^Q_{p,f})$ and hence in $Lang(e^R_{p_0,f})$. Similarly $x_2 c y_1$ is in $Lang(e^Q_{p_0,p} e^T_{p,q} e^T_{q,p}(e^Q_{p,p})^* e^Q_{p,f})$ and also in $Lang(e^R_{p_0,f})$. In both cases the same derivatives, giving the language for the expression $e^T_{q,p}(e^Q_{p,p})^* e^Q_{p,f}$, appear in the set $D^c$. By equal choice, this argument extends if other global actions are also enabled along with $c$. $\square$

**Theorem 2.** *Let $A$ be a product system with a conflict-equivalent matching. Then we can compute a connected expression for the language of $A$ with an equal choice pairing of actions.*

*Proof.* Let $A$ be a product system with a conflict-equivalent matching. Enumerate the global actions $a, b, \ldots$. Say the $\mathit{matching}(a)$ has $n$ tuples.

We construct a new product system $A'$ where, for the places in the $j$'th tuple of the $\mathit{matching}(a)$, we change the label of the outgoing $a$-moves to $a^j$; similarly for the places in tuples of the $\mathit{matching}(b)$; and so on. We now have a new product system where the letter $a$ of the alphabet has been replaced by the set $\{a^1, \ldots, a^n\}$; the letter $b$ has been replaced by another set; and so on, obtaining a new distribution $\Sigma'$. By definition of a matching, the various labels do not interfere with each other, so we have a matching with the new alphabet, conflict-equivalent if the previous one was. Runs which were consistent with the matching continue to be consistent with the new matching. Again by the definition of matching, the new system $A'$ has separation of labels. Hence we can apply Lemma 2.

From the Lemma 2 we get a connected expression $e' = \mathit{fsync}(s_1, \ldots, s_k)$ for the language of $A'$ over $\Sigma'$ where every regular expression has unique sites. From the proof of the Lemma 2 we get for every sequential system $A'_i$ in the product, for the global actions $a^1, \ldots, a^n$, tuples $D'(a^j) = \Pi_{i \in loc(a)} D'_i(a^j)$ which are sites for $a^j$ in the expression $s_i$, for every $j$. Now substitute $a$ for every letter $a^1, \ldots, a^n$ in the expression, each tuple $D'$ is isomorphic to a tuple $D$ of sites for $a$ in $e$ and the sites are disjoint from one another. We let $\mathit{pairing}(a)$ be the partition formed by these tuples. Do the same for $b$ obtaining $\mathit{pairing}(b)$. Repeat this process until all the global actions have been dealt with. The result is an expression $e$ with pairing of actions. If the matching was conflict-equivalent, the pairing has equal choice.

The runs of $A$ have to use product places in $pre(a)$ for global action $a$, define

$$L = \mathit{Lang}(A) \cap \Sigma^* a \Sigma^* = \bigcup_{R \in pre(a)} \mathit{Pref}_a^R(\mathit{Lang}(A))\ a\ \mathit{Suf}_a^R(\mathit{Lang}(A)).$$

The renaming of moves depends on the source place, so $L$ is isomorphic to

$$L' = \mathit{Lang}(A') \cap (\sum_j (\Sigma')^* a^j (\Sigma')^*) = \bigcup_{j=1,n} \mathit{Pref}_{a^j}(\mathit{Lang}(A')) a^j \mathit{Suf}_{a^j}(\mathit{Lang}(A')).$$

Keeping Proposition 6 in our hands, the Lemma 2 ensures that $\mathit{Lang}(A') = \mathit{Lang}(e')$ and the expression $e'$ has unique $a^j$-sites forming a block $D'(j)$. Then $L'$ can be written as $\bigcup_{j=1,n} \mathit{Pref}_{a^j}^{D'(j)}(\mathit{Lang}(e')) a^j \mathit{Suf}_{a^j}^{D'(j)}(\mathit{Lang}(e'))$. When we rename the $a^j$ back to $a$ we have a partition of $\mathit{pairing}(a)$ into sets $D$ such that

$$L = \bigcup_{D \subseteq \mathit{pairing}(a)} \mathit{Pref}_a^D(\mathit{Lang}(e))\ a\ \mathit{Suf}_a^D(\mathit{Lang}(e)).$$

If all runs of $A$ were consistent with the $matching(a)$, the product states in $pre(a)$ would all be in the $matching(a)$, and we obtain that the expression $e$ is consistent with the $pairing(a)$. □

*Example 11.* Let $\Sigma$ be a distributed alphabet and $(\Sigma_1 = \{a\}, \Sigma_2 = \{a\})$ be a distribution of $\Sigma$. Consider a product system $A = (A_1, A_2)$ with matching, defined over $\Sigma$, as shown in Figure 6. A matching relation for global action $a$ is: $matching(a) = \{(D_1, D_4), (D_2, D_5), (D_3, D_6)\}$.

Let $w = (D_1, D_4), y = (D_2, D_5)$ and $b = (D_3, D_6)$. Hence, we have new alphabet $\Sigma' = \{a_w, a_y, a_b\}$ with distribution $\Sigma'_1 = \{a_w, a_y, a_b\}$ and $\Sigma'_2 = \{a_w, a_y, a_b\}$. We now have a new product system $A' = (A'_1, A'_2)$ in which each action labelled $a$ of has been replaced by an action from $\{a_w, a_y, a_b\}$; Again by the definition of matching, the new system $A'$ has separation of labels. Hence we can apply Lemma 2, to get a connected expression $e' = fsync((a_w a_y a_b)^* a_w a_y a_b, (a_w a_y a_b)^*)$ defined over $\Sigma'$, language equivalent to $A'$ and have unique sites. Derivatives for $s'_1 = (a_w a_y a_b)^* a_w a_y a_b$ and $s'_2 = (a_w a_y a_b)^* a_w a_r a_b$ are shown in the Figure 5. Since $e'$ has unique actions, for action $a_w$, there is only one block in the partitions of $a_w$-sites of $s'_1$ and $s'_2$: $Part_{a_w}(s'_1)$ and $Part_{a_w}(s'_2)$, and for remaining global actions $a_y, a_b$ also. For action $a_w$ partition set is: $Part_{a_w}(s'_1) = \{D'_1\}$, $Part_{a_w}(s'_2) = \{D'_4\}$, for action $a_y$: $Part_{a_y}(s'_1) = \{D'_2\}$, $Part_{a_y}(s'_2) = \{D'_5\}$, and, for action $a_b$: $Part_{a_b}(s'_1) = \{D'_3\}$, $Part_{a_b}(s'_2) = \{D'_6\}$.

Now we replace each $a_w, a_y$ and $a_b$ in $e'$ by action a to get expression $e = fsync((aaa)^* aaa, (aaa)^*)$ defined over $\Sigma$. For blocks $D'_i$ we get respective blocks $D_i$, as shown in Figure 1. And, pairing relation for $a$ is: $pairing(a) = \{(D_1, D_4), (D_2, D_5), (D_3, D_6)\}$.

## 6 Applying the Kleene Result to Nets

We now wish to see how the Kleene result between expressions and product systems proved in Theorems 1 and 2 can be applied to net systems. First some definitions.

**Definition 11.** *A labelled net $N$ is a tuple $(S, T, F, \lambda)$, where $S$ is a set of places, $T$ is a set of transitions labelled by the function $\lambda : T \to \Sigma$ and $F \subseteq (T \times S) \cup (S \times T)$ is the flow relation. It will be convenient to define $loc(t) = loc(\lambda(t))$.*

Elements of $S \cup T$ are called nodes of $N$. Given a node $z$ of net $N$, set ${}^\bullet z = \{x \mid (x, z) \in F\}$ is called pre-set of $z$ and $z^\bullet = \{x \mid (z, x) \in F\}$ is called post-set of $z$. Given a set $Z$ of nodes of $N$, let ${}^\bullet Z = \bigcup_{z \in Z} {}^\bullet z$ and $Z^\bullet = \bigcup_{z \in Z} z^\bullet$. We only consider nets in which every transition has nonempty pre- and post-set.

**Definition 12.** *Let $N' = (S \cap X, T \cap X, F \cap (X \times X))$ be a subnet of net $N = (S, T, F)$, generated by a nonempty set $X$ of nodes of $N$. $N'$ is called a* ***component*** *of $N$ if,*

- *For each place $s$ of $X$, ${}^\bullet s, s^\bullet \subseteq X$ (the pre- and post-sets are taken in $N$),*
- *For all transitions $t \in T$, we have $|{}^\bullet t| = 1 = |t^\bullet|$ ($N'$ is an S-net [DE95]),*

– *Under the flow relation, $N'$ is connected.*

*A set $\mathcal{C}$ of components of net $N$ is called **S-cover** for $N$, if every place of the net belongs to some component of $\mathcal{C}$. A net is* covered by components *if it has an $S$-cover.*

Note that our notion of component does not require strong connectedness and so it is different from notion of $S$-component in [DE95], and therefore our notion of $S$-cover also differs from theirs.

Fix a distribution $(\Sigma_1, \Sigma_2, \dots, \Sigma_k)$ of $\Sigma$. The next definition appears in several places for unlabelled nets, starting with [Hac72].

**Definition 13.** *A labelled net $N = (S, T, F, \lambda)$ is called **S-decomposable** if, there exists an S-cover $\mathcal{C}$ for $N$, such that for each $T_i = \{\lambda^{-1}(a) \mid a \in \Sigma_i\}$, there exists $S_i$ such that the induced component $(S_i, T_i, F_i)$ is in $\mathcal{C}$.*

Now from S-decomposability we get an $S$-cover for net $N$, since there exist subsets $S_1, S_2, \dots, S_k$ of places $S$, such that $S = S_1 \cup S_2 \cup \dots S_k$ and ${}^\bullet S_i \cup S_i^\bullet = T_i$, such that the subnet $(S_i, T_i, F_i)$ generated by $S_i$ and $T_i$ is an S-net, where $F_i$ is the induced flow relation from $S_i$ and $T_i$.

### 6.1 Free Choice Nets

**Definition 14 ([DE95]).** *Let $x$ be a node of a net $N$. The* cluster *of $x$, denoted by $[x]$, is the minimal set of nodes contaning $x$ such that*

– *if a place $s \in [x]$ then $s^\bullet$ is included in $[x]$, and*
– *if a transition $t \in [x]$ then ${}^\bullet t$ is included in $[x]$.*

*A cluster $C$ is called* free choice *(FC) if all transitions in $C$ have the same pre-set. A net is called* free choice *if all its clusters are free choice.*

In a labelled $N$, for a cluster $C = (S_C, T_C)$ define the $a$-labelled transitions $C_a = \{t \in T_C \mid \lambda(t) = a\}$. If the net has an S-decomposition generated by $S_i$, we associate a post-product $\pi(t) = \Pi_{i \in loc(a)}(t^\bullet \cap S_i)$ with every such transition $t$. This is well defined since by the S-net condition every transition will have at most one post-place in $S_i$. Let $post(C_a) = \bigcup_{t \in C_a} \pi(t)$. We also define the post-projection of the cluster $C_a[i] = {C_a}^\bullet \cap S_i$ and the **post-decomposition** $postdecomp(C_a) = \Pi_{i \in loc(a)} C_a[i]$.

Clearly $post(C_a) \subseteq postdecomp(C_a)$. The following definition appears in [Pha14b], and provides the way to direct product representability.

**Definition 15 ([Pha14b]).** *An S-decomposable net $N = (S, T, F, \lambda)$ is said to be **distributed choice** if, for all global actions $a$ in $\Sigma$ and for all clusters $C$ of $N$, $postdecomp(C_a) \subseteq post(C_a)$.*

### 6.2 Net Systems and Their Languages

For our results we are only interested in 1-bounded (or condition/event) nets, where a place is either marked or not marked. Hence we define a marking as a function from the states of a net to $\{0, 1\}$.

A transition $t$ is enabled in a marking $M$ if all places in its pre-set are marked by $M$. In such a case, $t$ can be fired to yield the new marking $M' = (M \setminus {}^{\bullet}t) \cup t^{\bullet}$. We write this as $M[t\rangle M'$ or $M[\lambda(t)\rangle M'$.

A firing sequence (finite or infinite) $\lambda(t_1)\lambda(t_2)\ldots$ is defined by composition, from $M_0[t_1\rangle M_1[t_2\rangle \ldots$ For every $i \leq j$, we say that $M_j$ is reachable from $M_i$. A net system $(N, M_0)$ is **live** if, for every reachable marking $M$ and every transition $t$, there exists a marking $M'$ reachable from $M$ which enables $t$.

**Definition 16.** *For a labelled net system $(N, M_0, \mathcal{G})$, its language is defined as $Lang(N, M_0, \mathcal{G}) = \{\lambda(\sigma) \in \Sigma^* \mid \sigma \in T^* \text{ and } M_0[\sigma\rangle M, \text{ for some } M \in \mathcal{G}\}$.*

If a net $(S, T, F, \lambda)$ is 1-bounded and S-decomposable then a marking can be written as a $k$-tuple from its components $S_1 \times S_2 \times \ldots \times S_k$. It is known [Zie87,Muk11] that if we do not enforce the "direct product" condition below we get a larger subclass of languages.

**Definition 17.** *An **S-decomposable labelled net system** $(N, M_0, \mathcal{G})$ is an S-decomposable labelled net $N = (S, T, F, \lambda)$ along with an initial marking $M_0$ and a set of markings $\mathcal{G} \subseteq \wp(S)$, which is a **direct product**: if $\langle q_1, q_2, \ldots q_k\rangle \in \mathcal{G}$ and $\langle q'_1, q'_2, \ldots q'_k\rangle \in \mathcal{G}$ then $\{q_1, q'_1\} \times \{q_2, q'_2\} \times \ldots \times \{q_k, q'_k\} \subseteq \mathcal{G}$.*

### 6.3 From Product Systems to Net Systems

From a product system we can straightforwardly construct a net.

**Definition 18 (Product to net).** *Given a product system $A = \langle A_1, \ldots, A_k\rangle$ over distribution $\Sigma$, we can produce a net system $(N = (S, T, F, \lambda), M_0, \mathcal{G})$ as follows:*

- *$S = \cup_i P_i$, the set of places.*
- *$T = \cup_a T_a$, where $T_a$ is $\Rightarrow_a$, the set of $a$-labelled global moves.*
- *The labelling function $\lambda$ labels by $a$ the transitions in $T_a$.*
- *The flow relation $F = \{(p, g), (g, q) \mid g \in T_a, g[i] = \langle p, a, q\rangle, i \in loc(a)\}$.*
- *$M_0 = \{p_1^0, \ldots, p_k^0\}$, the initial product state.*
- *$\mathcal{G} = G_1 \times \cdots \times G_k$, the set of final product states.*

Since a global action $a$ can be in every component $A_i$ of the product system and there can be an arbitrary number $n_i$ of $a$-labelled choices in each component, the resulting $a$-cluster in the net has $n_1 \times \cdots \times n_k$ transitions which can be exponential in the size of the product system.

Here is what this construction yields.

**Theorem 3 ([Pha14b,Pha14a]).** *In the construction of net system* $(N, M_0, \mathcal{G})$ *in Definition 18, N is S-decomposable, satisfies the distributed choice property, and with* $\mathrm{Lang}(N, M_0, \mathcal{G}) = \mathrm{Lang}(A)$*. Further, if all runs of A are consistent with a conflict-equivalent matching of labels, we can choose* $T' \subseteq T$ *such that the subnet* $N'$ *generated by* $T'$ *is a free choice net and* $(N', M_0, \mathcal{G})$ *accepts the same language.*

*Example 12.* For the product system shown in Figure 6 we construct a net as shown in Figure 7. It is free choice and language equivalent to product system $A$. Set of final markings for this net is $M_f = \{(d_6, D_4)\}$. Hence, we

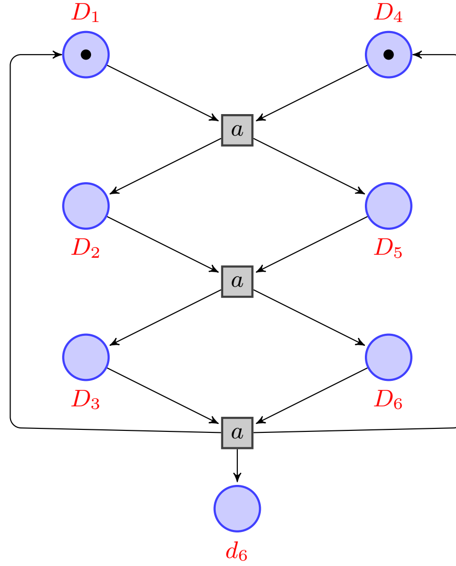


**Fig. 7.** Net system constructed from $A = (A_1, A_2)$ over $\Sigma$ of Figure 6.

get a language equivalent free choice net system for the connected expression $e = \mathit{fsync}((aaa)^*aaa, (aaa)^*)$ of Example 7. This net has distributed choice property.

### 6.4 S-decomposable Net Systems to Product Systems with Matching

For a net which is 1-bounded and S-decomposable it might have many S-covers for it. And, in an S-cover each component of it need not have only one token in it. For live and 1-bounded free choice nets, there exist at least one $S$-cover in which each component have only one token in it [DE95]. In this paper, when we say that a 1-bounded net is S-decomposable we refer to one such S-cover

in which each component has only one token. Taking this S-cover it is easy to construct a product system.

**Definition 19 (Net to product).** *Given a* 1*-bounded and S-decomposable labelled net system* $(N, M_0, \mathcal{G})$*, with* $N = (S, T, F, \lambda)$ *the underlying net and* $N_i = (S_i, T_i, F_i)$ *the components in the S-cover, for* $i$ *in* $\{1, 2, \ldots, k\}$*, we define a product system:*

- $P_i = S_i$, $p_i^0$ *the unique state in* $M_0 \cap P_i$.
- $\rightarrow_i = \{\langle p, \lambda(t), p' \rangle \mid t \in T_i$ *and* $(p, t), (t, p') \in F_i$*, for* $p, p' \in P_i\}$*. For each* $t \in T_i$*, we know that, there exist places* $p, p' \in S_i$ *such that* $(p, t)$ *and* $(t, p')$ *belong to* $F_i$.
- *So we get sequential system* $A_i = \langle P_i, \rightarrow_i, p_i^0 \rangle$ *and the product system* $A = \langle A_1, A_2, \ldots, A_k \rangle$ *over distributed alphabet* $\Sigma$.
- $G = \{(M \cap P_1, \ldots, M \cap P_k) \mid M \in \mathcal{G}\}$*. If* $\mathcal{G}$ *was a direct product set of final markings, we can define* $G_i = \{M \cap P_i \mid M \in \mathcal{G}\}$ *and set* $G$ *to be their product* $G_1 \times \cdots \times G_k$.

The distributed choice property yields the following results. The references below provide counterexamples when the distributed choice condition is not met.

**Theorem 4 ([Pha14b,Pha14a]).** *When the given net* $N$ *satisfies the distributed choice property, the construction of the product system* $A$ *in Definition 19 preserves language, that is, when restricted to runs consistent with a matching,* $\mathrm{Lang}(N, M_0, \mathcal{G}) = \mathrm{Lang}(A)$*. Further, since the net is free choice, the matching is conflict-equivalent, making* $A$ *an FC-matching product.*

*Example 13.* Let $\Sigma$ be a distributed alphabet and $(\Sigma_1 = \{a\}, \Sigma_2 = \{a\})$ be a distribution of $\Sigma$. Consider the free choice net as shown in Figure 7. It has distributed choice property, and an S-decomposition is $\{D_1, D_2, D_3, d_6\}$ and $\{D_4, D_5, D_6\}$. A set of final markings for this net is $M_f = \{(d_6, D_4)\}$. As one may expect this produces the language equivalent product system $A$ shown in Figure 6.

### 6.5 Conclusion

In earlier work [LMP11], we showed that a graph-theoretic condition called “structural cyclicity” enables us to extract syntax from a conflict-equivalent product system. In the present work we have generalized this condition so that we can deal with a larger class of product systems with a conflict-equivalent matching. Using [Pha14b,Pha14a] we obtain a Kleene characterization for the class of labelled free choice nets with the distributed choice property using a pairing condition on connected expressions.

*Acknowledgements.* We thank the referees of the PNSE workshop and the referees of ToPNOC, who urged us to improve the presentation of the proofs of the main theorems and to correct and clarify our results. We would also like to thank Jörg Desel for his patience as editor of the ToPNOC special issue.

## References


[Ant96] Valentin Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoret. Comp. Sci.*, 155(2):291–319, 1996.

[Arn94] André Arnold. *Finite transition systems.* Prentice-Hall, 1994.

[Brz64] Janusz A. Brzozowski. Derivatives of regular expressions. *JACM*, 11(4):481–494, 1964.

[DE95] Jörg Desel and Javier Esparza. *Free choice Petri nets.* Cambridge University Press, New York, USA, 1995.

[GR92] Vijay K. Garg and M.T. Ragunath. Concurrent regular expressions and their relationship to Petri nets. *Theoret. Comp. Sci.*, 96(2):285–304, 1992.

[Gra81] Jan Grabowski. On partial languages. *Fund. Inform.*, IV(2):427–498, 1981.

[Hac72] Michel Henri Théodore Hack. Analysis of production schemata by Petri nets. Project Mac Report TR-94, MIT, 1972.

[LMP11] Kamal Lodaya, Madhavan Mukund, and Ramchandra Phawade. Kleene theorems for product systems. In Markus Holzer, Martin Kutrib, and Giovanni Pighizzini, editors, *Proc. 13th DCFS, Limburg*, volume 6808 of *LNCS*, pages 235–247, 2011.

[LRR03] Kamal Lodaya, D. Ranganayakulu, and K. Rangarajan. Hierarchical structure of 1-safe Petri nets. In Vijay A. Saraswat, editor, *Proc. 8th Asian, Mumbai*, volume 2896 of *LNCS*, pages 173–187, 2003.

[LS05] Sylvain Lombardy and Jacques Sakarovitch. How expressions can code for automata. *RAIRO Theor. Inform. Appl.*, 39(1):217–237, 2005.

[LS10] Sylvain Lombardy and Jacques Sakarovitch. Corrigendum. *RAIRO Theor. Inform. Appl.*, 44(3):339–361, 2010.

[LW00] Kamal Lodaya and Pascal Weil. Series-parallel languages and the bounded width property. *Theoret. Comp. Sci.*, 237(1-2):347–380, 2000.

[Mir66] Boris G. Mirkin. An algorithm for constructing a base in a language of regular expressions. *Engg. Cybern.*, 5:110–116, 1966.

[MR02] Swarup Mohalik and R. Ramanujam. Distributed automata in an assumption-commitment framework. *Sādhanā*, 27, part 2:209–250, April 2002.

[Muk11] Madhavan Mukund. Automata on distributed alphabets. In Deepak D'Souza and Priti Shankar, editors, *Modern applications of automata theory*, pages 257–288. World Scientific, 2011.

[MY60] Robert McNaughton and Hisao Yamada. Regular expressions and state graphs for automata. *IEEE Trans. IRS*, EC-9:39–47, 1960.

[Pha14a] Ramchandra Phawade. Direct product representation of labelled free choice nets. *Int. J. Comp. Appl.*, 99(16), 2014.

[Pha14b] Ramchandra Phawade. *Labelled Free Choice Nets, finite Product Automata, and Expressions.* PhD thesis, Homi Bhabha National Institute, 2014. Submitted.

[PL14] Ramchandra Phawade and Kamal Lodaya. Kleene theorems for labelled free choice nets. In *Proc. 8th PNSE, Tunis*, volume 1160 of *CEUR-WS*, pages 75–89, 2014.

[SH96] Pablo A. Straub and L. Carlos Hurtado. Business process behaviour is (almost) free-choice. In *Proc. CESA, Lille*, pages 9–12. IEEE, 1996.

[TY14] P.S. Thiagarajan and Shaofa Yang. Rabin's theorem in the concurrency setting: a conjecture. *Theoret. Comp. Sci.*, 546:225–236, 2014.

[Zie87] Wiesław Zielonka. Notes on finite asynchronous automata. *Inform. Theor. Appl.*, 21(2):99–135, 1987.