

# Kleene Theorem for Labelled Free Choice Nets without Distributed Choice

Ramchandra Phawade

Indian Institute of Technology Bombay, Powai, Mumbai 400076, India

**Abstract.** For 1-bounded nets, equivalent syntax have been given by Grabowski [8], Garg and Ragnunath [7] and other authors Lodaya [15]. We work with 1-bounded, labelled free choice nets given with an S-cover, where the labels come from a distributed alphabet and S-components of the associated S-cover respect the alphabet distribution. In earlier work [21], we provided expressions for free choice nets having “distributed choice property” which makes the nets “direct product” representable. These expressions have “pairings” of blocks of derivatives—representing automaton states—which specify how synchronizations take place. In this work, we give equivalent syntax for a larger class of free choice nets obtained by dropping “distributed choice property”. Hence we deal with the nets which are not representable by direct products also. Now “pairings” relate the tuples of blocks of derivatives and their effects—representing automaton transitions—in our expressions.

## 1 Introduction

The Kleene theorem for regular expressions enables us to translate from regular expressions to automata and back [13]. Expressions are widely used by programmers to describe languages of software components. They are thought of as a user-friendly alternative to the finite state machines for describing software components [12]. On the other hand they can be used for axiomatizing language equivalence [23, 14] and characterizing subclasses of automata.

Free choice nets form an important subclass of Petri nets, having pleasant theory [24, 5] and from verification point of view some of the advantages of 1-bounded free choice nets are: checking liveness is in PTIME [6, 4], checking deadlock is NP-complete [3] and reachability problem is PSPACE-complete [3]. All these problems are PSPACE-complete for 1-bounded nets [3].

Language equivalent expressions for 1-bounded nets have been given by Grabowski [9], Garg and Ragnunath [7] and other authors [15], where renaming operator has been used in the syntax to disambiguate synchronizations. In earlier [16, 21, 22] works, we have provided syntax for various subclasses of 1-bounded free choice nets, and we have used product systems as intermediary formalism. Since we do not use renaming operator in our syntax, our labellings has to preserve the process structure, i.e. same label can not be used for synchronizations by different sets of processes. This also enables us to compare expressiveness of nets with those of different classes of product automata [18].

A restriction of our earlier work [20], was that of “distributed choice”. Roughly, this says that if two transitions in a cluster of a free choice net are labelled the same, then they cannot be used to discriminate between post-places. In this paper we remove this restriction.

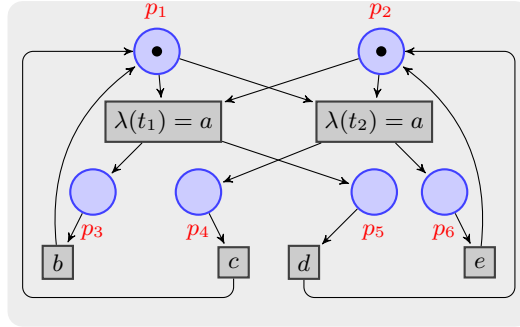


Fig. 1: Live and 1-bounded, labelled free choice net without distributed choice

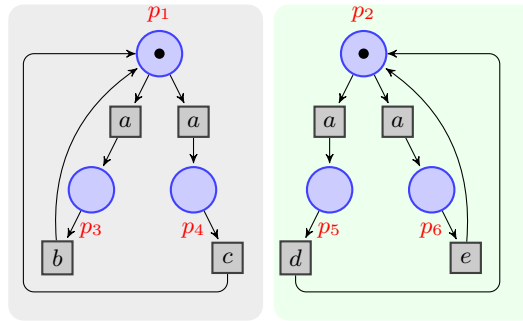


Fig. 2: S-decomposition of the net in Figure 1

The Figure 1 shows an example 1-bounded labelled net which is free choice but does not satisfy this property. A short proof [19] using a characterization of direct product languages [18] shows that the language of this net with final marking  $\{p_1, p_2\}$  is not accepted by a direct product. Its (only possible) S-decomposition is shown in Figure 2. If we used our earlier theorems on this example, we would get the informal expression  $fsync((ab + ac)^*, (ad + ae)^*)$ , that is, the *fsync* (synchronization) of the two regular expressions. But this is not what the net does, perhaps its behavior can be informally described as  $(fsync(ab, ad) + fsync(ac, ae))^*$ , the left choices (moves  $(p_1, a, p_3)$  and  $(p_2, a, p_5)$ )

and the right choices (moves  $(p_1, a, p_4)$  and  $(p_2, a, p_6)$ ) separately pair up to synchronize. In other words, what a net transition does has to be remembered in the product system, and that we capture by “globals” of product system, which are nothing but a set of–global moves–tuples of local moves. Indeed, these product systems with globals are more expressive than direct products and are a restricted version of Zielonka automaton [25].

In Section 2 we formally define the nets without distributed choice property and the product systems with globals and in the subsequent section we prove the equivalence of these two formalisms.

On the expressions side, these global moves are represented by “cables” which are tuples of “ducts” (representing local moves in the product system which in turn are arcs inside S-components). This syntactic machinery along with the syntax of expressions is developed in Section 4. In previous works [21, 22] we have developed the concept of “blocks” –collection of derivatives [1]–which represents a state of an automaton or a place in the net. To represent a local move  $(p, a, q)$  belonging to some component automaton– of a product system–in regular expression corresponding to it, we need to identify the blocks of derivatives corresponding to source place  $p$  and target place  $q$  of the local move. While the former part has been dealt previously [21], using the notion of “bifurcation”; it alone is not sufficient to get a block corresponding to the target place, because of two difficulties: first is the possibility of existence of multiple target states, for a given source state, due to nondeterminism on local actions; and the second difficulty arises from the fact that, in some sense, bifurcation works in the forward direction of outgoing moves and does not help in keeping the identity of target places intact. To handle this, we introduce the notion of “funneling” in Section 4, and the task of formally establishing the correspondence between a local move and a duct is achieved in Lemma 4 of Section 5. In the same section we prove equivalence between product systems with globals and expressions with cables. Final section summarizes the work and discusses time complexities involved in the translations.

## 2 Preliminaries

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 (Distributed Alphabet).** Let  $Loc$  denote the set  $\{1, 2, \dots, k\}$ . A distribution of  $\Sigma$  over  $Loc$  is a tuple of nonempty sets  $(\Sigma_1, \Sigma_2, \dots, \Sigma_k)$  with  $\Sigma = \bigcup_{1 \leq i \leq k} \Sigma_i$ . 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*.

For a set  $S$  let  $\wp(S)$  denote the set of all subsets of  $S$ .

## 2.1 Nets

**Definition 2.** 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  $\lambda : T \rightarrow \Sigma$  and  $F \subseteq (T \times S) \cup (S \times T)$  is the flow relation.

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. For all actions  $a$  in  $\Sigma$  let  $T_a = \{t \mid t \in T \text{ and } \lambda(t) = a\}$ .

**Net Systems and their Languages** For our results we are only interested in 1-bounded nets, where a place is either marked or not marked. Hence we define a marking of a net as a subset of its places.

**Definition 3.** A labelled net system is a tuple  $(N, M_0, \mathcal{G})$  where  $N = (S, T, F, \lambda)$  is a labelled net with  $M_0 \subseteq S$  as its initial marking and a set of markings  $\mathcal{G} \subseteq \wp(S)$ .

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  $M' = (M \setminus \bullet t) \cup t \bullet$  and, we write this as  $M[t]M'$  or  $M[\lambda(t)]M'$ .

A firing sequence (finite or infinite)  $\lambda(t_1)\lambda(t_2)\dots$  is defined by composition, from  $M_0[t_1]M_1[t_2]\dots$ . For every  $i \leq j$ , we say that  $M_j$  is reachable from  $M_i$ . We say 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 4.** For a labelled net system  $(N, M_0, \mathcal{G})$ , its language is defined as  $\text{Lang}(N, M_0, \mathcal{G}) = \{\lambda(\sigma) \in \Sigma^* \mid \sigma \in T^* \text{ and } M_0[\sigma]M, \text{ for some } M \in \mathcal{G}\}$ .

### Net Systems and its components

**Definition 5.** 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  $N$ . Subnet  $N'$  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 [5]),
- 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 [5], 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 [11].

**Definition 6.** 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  $S = S_1 \cup S_2 \cup \dots \cup S_k$  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  $T_i$ .

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 component places  $S_1 \times S_2 \times \dots \times S_k$ . If we do not enforce the “direct product” condition on the set of final markings, given below, it is known [25, 18] that we get a larger subclass of languages.

**Definition 7.** 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  $\langle q_1, q_2, \dots, q_k \rangle \in \mathcal{G}$  and  $\langle q'_1, q'_2, \dots, q'_k \rangle \in \mathcal{G}$  then  $\{q_1, q'_1\} \times \{q_2, q'_2\} \times \dots \times \{q_k, q'_k\} \subseteq \mathcal{G}$ .

Let  $t$  be a transition in  $T_a$ . Then by S-decomposability a pre-place and a post-place of  $t$  belongs to each  $S_i$  for all  $i$  in  $loc(a)$ . Let  $t[i]$  denote the tuple  $\langle p, a, p' \rangle$  such that  $(p, t), (t, p') \in F_i$ , for  $p, p' \in P_i$  for all  $i$  in  $loc(a)$ .

### Free choice nets and distributed choice

**Definition 8 (Free choice nets [5]).** Let  $x$  be a node of a net  $N$ . The cluster of  $x$ , denoted by  $[x]$ , is the minimal set of nodes containing  $x$  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 net  $N$ , for a free choice 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 then we associate a post-product  $\pi(t) = \prod_{i \in loc(a)} (t^\bullet \cap S_i)$  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**  $postdecomp(C_a) = \prod_{i \in loc(a)} C_a[i]$ . Clearly  $post(C_a) \subseteq postdecomp(C_a)$ . The following definition generalized from [20, 19] provides the way to direct product representability.

**Definition 9.** An S-decomposable net  $N = (S, T, F, \lambda)$  is said to have **distributed choice** property if, for all  $a$  in  $\Sigma$  and for all free choice clusters  $C$  of  $N$ ,  $postdecomp(C_a) \subseteq post(C_a)$ .

*Example 1 (A net without distributed choice property).* Consider a distributed alphabet  $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$ . Consider the labelled net system  $(N = (S, T, F, \lambda), \{p_1, p_2\}, \{\{p_1, p_2\}\})$  in Figure 1.

Its (only possible) S-cover having two S-components  $N_1$  and  $N_2$  is given in Figure 2. We have  $N_1 = (S_1, T_1, F_1)$  and  $N_2 = (S_2, T_2, F_2)$ , with the set of places  $S_1 = \{p_1, p_3, p_4\}$  and  $S_2 = \{p_2, p_5, p_6\}$ . Because we have net decomposed into these two S-components, as mentioned before we can write the markings as tuples, therefore alternately we can have net described as  $(N, (p_1, p_2), \{\{p_1, p_2\}\})$ . For the cluster  $[p_1]$ , its set of  $a$ -labelled transitions is  $C_a = \{t_1, t_2\}$ . We have sets  $C_a[1] = \{p_3, p_4\}$  and  $C_a[2] = \{p_5, p_6\}$  therefore we get  $postdecomp(C_a) = \{(p_3, p_5), (p_3, p_6), (p_4, p_5), (p_4, p_6)\}$ . Now  $\pi(t_1) = (t_1 \bullet \cap S_1) \times (t_1 \bullet \cap S_2) = \{p_3\} \times \{p_5\} = \{(p_3, p_5)\}$  and  $\pi(t_2) = (t_2 \bullet \cap S_1) \times (t_2 \bullet \cap S_2) = \{p_4\} \times \{p_6\} = \{(p_4, p_6)\}$ , therefore  $Post(C_a) = \{(p_3, p_5), (p_4, p_6)\}$ . Hence  $postdecomp(C_a) \neq Post(C_a)$ . Since this cluster does not have distributed choice property the net system does not have distributed choice property.

*Example 2 (A net with distributed choice property).* Consider a distributed alphabet  $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$ . Consider the labelled net system  $(N, (p_1, p_2), \{\{p_1, p_2\}\})$  in Figure 3. Its (only possible) S-cover having two S-components  $N_1$  and  $N_2$  is given in Figure 4. We have components  $N_1 = (S_1, T_1, F_1)$  and  $N_2 = (S_2, T_2, F_2)$  with sets of places  $S_1 = \{p_1, p_3, p_4\}$  and  $S_2 = \{p_2, p_5\}$ . For cluster  $[p_1]$ , the set of  $a$ -labelled transitions is  $C_a = \{t_1, t_2\}$ . We have  $C_a[1] = \{p_3, p_4\}$  and  $C_a[2] = \{p_5\}$ , hence we get  $postdecomp(C_a) = \{(p_3, p_5), (p_4, p_5)\}$ . Now  $\pi(t_1) = (t_1 \bullet \cap S_1) \times (t_1 \bullet \cap S_2) = \{p_3\} \times \{p_5\} = \{(p_3, p_5)\}$  and  $\pi(t_2) = (t_2 \bullet \cap S_1) \times (t_2 \bullet \cap S_2) = \{p_4\} \times \{p_5\} = \{(p_4, p_5)\}$ , therefore  $Post(C_a) = \{(p_3, p_5), (p_4, p_5)\}$ . Hence  $postdecomp(C_a) = \{(p_3, p_5), (p_4, p_5)\} = Post(C_a)$ . For all other clusters this holds trivially, because each of them have only one transition and only one post-place, hence the net has distributed choice.

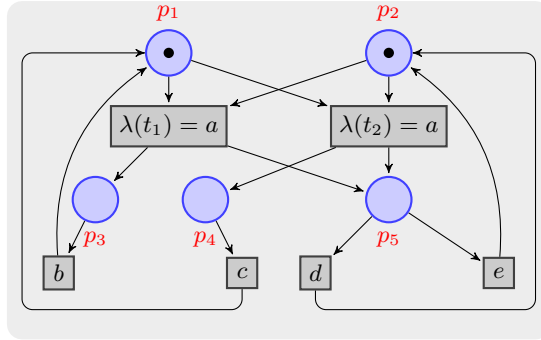


Fig. 3: Live and 1-bounded, labelled free choice net with distributed choice

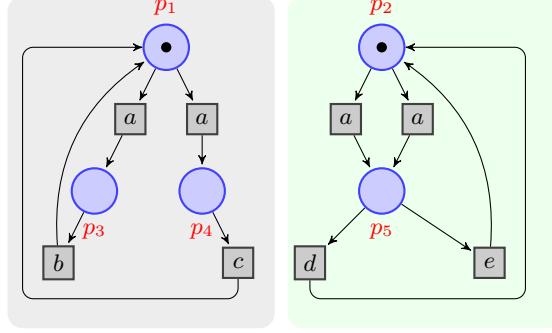


Fig. 4: S-decomposition of the net in Figure 3

## 2.2 Automata and their properties

**Definition 10.** A sequential system over a set of actions  $\Sigma_i$  is a finite state automaton  $A_i = \langle P_i, \rightarrow_i, G_i, p_i^0 \rangle$  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  $\rightarrow_i \subseteq P_i \times \Sigma_i \times P_i$  is a set of **local moves**.

Let  $\rightarrow_a^i$  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  $\rightarrow_i$  place  $p$  is called **pre-place** and  $p'$  is called **post-place** of  $t$ . The language of a sequential system is defined as usual.

A set of places  $X \subseteq P_i$  of component  $A_i$  having language  $L$  we define relativized languages and we will extend this definition to product systems:  $L_a^X = \{xay \mid xay \in L \text{ and } \exists p \text{ in } X \text{ such that } p_0 \xrightarrow{x} p \xrightarrow{ay} G_i\}$ ,  $Pref_a^X(L) = \{x \mid xay \in L \text{ and } xay \in L_a^X\}$ , and  $Suf_a^X(L) = \{y \mid xay \in L \text{ and } xay \in L_a^X\}$ . A set of places  $X \subseteq P_i$  of component  $A_i$   **$a$ -bifurcates**  $L$  if  $L_a^X = Pref_a^X(L) \cdot a \cdot Suf_a^X(L)$ .

These relativized languages and the property of  $a$ -bifurcation has been defined earlier [21] for a place i.e., when set  $X$  is singleton; here we have generalized those notions to a set of places.

*Remark 1.* A place  $p$  of  $A_i$   $a$ -bifurcates  $Lang(A_i)$ .

For a place  $p$  let  $a\text{-targets}(p) = \{q \mid \langle p, a, q \rangle \in \rightarrow_i\}$  denote the set of its target places. For a set  $X$  of places of  $A_i$  define  $a\text{-targets}(X) = \bigcup_{p \in X} a\text{-targets}(p)$ . For sets of places  $X$  and  $Y$  of  $A_i$  with language  $L$ , we define relativized languages:  $L_a^{(X,Y)} = \{xay \mid xay \in L, \exists p \in X \text{ and } \exists q \in Y \text{ such that } p_0 \xrightarrow{x} p \xrightarrow{a} q \xrightarrow{y} G_i\}$ ,  $Pref_a^{(X,Y)}(L) = \{x \mid xay \in L_a^{(X,Y)}\}$  and  $Suf_a^{(X,Y)}(L) = \{y \mid xay \in L_a^{(X,Y)}\}$ . And, we say that  $(X, Y)$   **$a$ -funnels** language  $L$  if  $L_a^{(X,Y)} = Pref_a^{(X,Y)}(L) \cdot a \cdot Suf_a^{(X,Y)}(L)$ .

*Remark 2.* For a set  $X$  of places of  $A_i$  with language  $L$  and a set  $Y \subseteq a\text{-targets}(X)$ , if  $X$   $a$ -bifurcates  $L$  then the tuple  $(X, Y)$   $a$ -funnels  $L$ .

Using Remark 1 and Remark 2 we get the following corollary.

**Corollary 1.** Given a place  $p$  of  $A_i$  and a set  $Y \subseteq a\text{-targets}(p)$  the tuple  $(p, Y)$   $a$ -funnels  $L(A_i)$ .

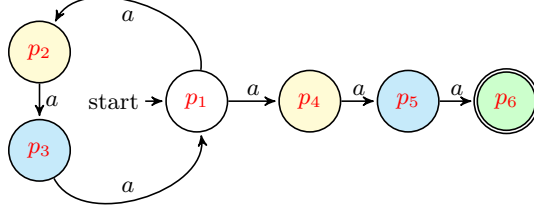


Fig. 5: Automaton of Example 3 accepting language  $L((aaa)^*aaa)$

*Example 3.* Let  $L$  denote the language of automaton shown in Figure 5. The set  $X = \{p_2, p_4\}$   $a$ -bifurcates language  $L$  and the tuple  $(X, \{p_3\})$   $a$ -funnels  $L$ . The set  $Z = \{p_1, p_3\}$  does not  $a$ -bifurcate language  $L$  because, for the word  $aaaaaa$  in  $L$ , we have string  $aaa$  in  $Suf_a^{p_3}(L)$  (and hence in  $Suf_a^Z(L)$ ) and we have string  $\varepsilon$  in  $Pref_a^{p_1}(L)$  (and hence in  $Pref_a^Z(L)$ ), but the word  $\varepsilon \cdot a \cdot aaa$  is not in  $L$ . Now let  $Y = \{p_1, p_2\}$ . Considering the same string as above, we get string  $\varepsilon$  in  $Pref_a^{(p_1, p_2)}(L)$  (and hence in  $Pref_a^{(Z, Y)}(L)$ ), and string  $aaa$  in  $Suf_a^{(p_3, p_1)}(L)$  (and hence in  $Suf_a^{(Z, Y)}(L)$ ). But the word  $\varepsilon \cdot a \cdot aaa$  is not in  $L$ , and hence not in  $L_a^{(Z, Y)}$ . Therefore, tuple  $(Z, Y)$  do not  $a$ -funnel  $L$ .

### 2.3 Product Systems

Fix a distribution  $(\Sigma_1, \Sigma_2, \dots, \Sigma_k)$  of  $\Sigma$ . We define product systems over this.

**Definition 11.** Let  $A_i = \langle P_i, \rightarrow_i, G_i, p_i^0 \rangle$  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  $\prod_{i \in Loc} P_i$  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 relativized language  $L^R$  of a language  $L \subseteq \Sigma_i^*$  consider projections to place  $R[i]$  in  $A_i$ .

The initial product state of  $A$  is  $R^0 = (p_1^0, \dots, p_k^0)$ , while  $G = \prod_{i \in Loc} G_i$  denotes the final states of  $A$ .

Let  $\Rightarrow_a = \prod_{i \in loc(a)} \rightarrow_a^i$ . The set of all global moves of  $A$  is  $\Rightarrow = \bigcup_{a \in \Sigma} \Rightarrow_a$ . Then for a global move  $g$  in  $\Rightarrow_a$ , we define its set of pre-places  $\mathbf{pre}(g)$  as the set of pre-places of all its component  $a$ -moves; set of post-places  $\mathbf{post}(g)$  as the set of post-places of all its component  $a$ -moves; and, let  $g[i]$  denote its  $i$ -th component-local  $a$ -move-belonging to  $A_i$ , for all  $i$  in  $loc(a)$ .



**Definition 12 ([16]).** In a product system, we say the local move  $\langle p, a, q_1 \rangle \in \rightarrow_i$  is **conflict-equivalent** to the local move  $\langle p', a, q'_1 \rangle \in \rightarrow_j$ , if for every other local move  $\langle p, b, q_2 \rangle \in \rightarrow_i$ , there is a local move  $\langle p', b, q'_2 \rangle \in \rightarrow_j$  and, conversely, for moves from  $p'$  there are corresponding outgoing moves from  $p$ . We call  $A = \langle A_1, \dots, A_k \rangle$  a **conflict-equivalent product system** if for every global action  $a \in \Sigma$ , and for all  $i, j \in \text{loc}(a)$ , every  $a$ -move in  $A_i$  is conflict-equivalent to every  $a$ -move in  $A_j$ .

**Definition 13 ([22]).** A product system  $A$  is said to have **separation of labels** if for all  $i \in \text{Loc}$ , and for all global actions  $a$ , if  $\langle p, a, p' \rangle, \langle q, a, q' \rangle \in \rightarrow_i$  then  $p = q$ .

*Example 4.* Consider the product system  $A = (A_1, A_2)$  shown in Figure 6 which is defined over distributed alphabet  $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$ . In sequential component  $A_1$ , place  $r_1$  is the only one which has outgoing local  $a$ -moves, and similarly in component  $A_2$ , the only place which has outgoing local  $a$ -moves is  $s_1$ . Therefore,  $A$  has separation of labels property.

Let  $\text{globals}(a)$  be a subset of its global moves  $\Rightarrow_a$ , and  $a$ -**global** denote an element of  $\text{globals}(a)$ .

**Definition 14.** A **product system with globals** is a product system with relations  $\text{globals}(a)$ , for each global action  $a$  in  $\Sigma$ .

Now we describe runs of a product system  $A$  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  $j \in \text{loc}(a)$ ,  $\langle R[j], a, Q[j] \rangle \in \rightarrow_j$ , and for all  $j \notin \text{loc}(a)$ ,  $R[j] = Q[j]$ .

Runs of a product system with globals, are defined in the same way as above where, an additional requirement of  $\prod_{j \in \text{loc}(a)} (\langle R[j], a, Q[j] \rangle \in \text{globals}(a))$ , has to be satisfied. With abuse of notation sometimes we use  $\text{pre}(a)$  to denote the set  $\{R \mid \exists Q, R \xrightarrow{a} Q\}$ .

A run of a product system over word  $w$  is said to be **accepting** if the product state reached after  $w$  is in  $G$ . We define the **language**  $\text{Lang}(A)$  of product system  $A$  (with globals), as the words on which the product system (with globals) has an accepting run.

The following property helps us to capture free choice property of nets.

**Definition 15.** A product system with globals have **same source property** if, for any pair of two global moves sharing a common pre-place have the same set of pre-places.

*Example 5.* Consider the product system  $A = (A_1, A_2)$  shown in Figure 6 defined over a distributed alphabet  $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$ . We have action  $a$  is the only global action. The given relation  $\text{globals}(a)$  is  $\{((r_1 \xrightarrow{a} r_2), (s_1 \xrightarrow{a} s_2)), ((r_1 \xrightarrow{a} r_3), (s_1 \xrightarrow{a} s_3))\}$ . Since both  $a$ -globals in it have same set of pre-places, the product system has same source property.

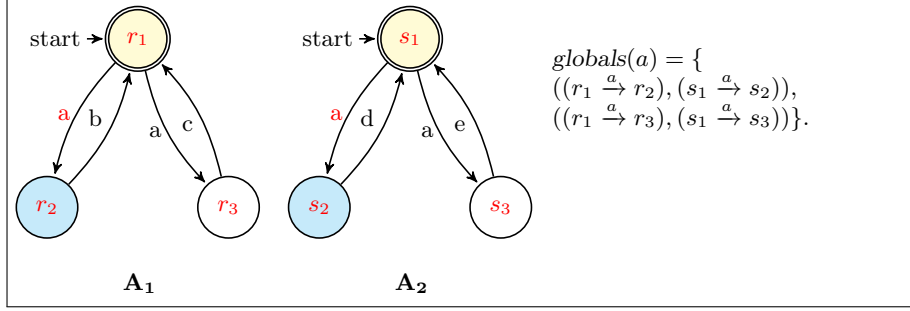


Fig. 6: Product system  $A = (A_1, A_2)$ , used in Example 5 and Example 9

## 2.4 Expressive power of product systems with globals

Product systems considered here are more expressive than direct products and less expressive than Zielonka automaton. An easy but long proof shows that, the class of synchronous products, whose expressive power lies in between direct product and Zielonka automaton, are incomparable with the product systems considered here.

## 3 Nets and Product systems

It is easy to give a generic construction of a 1-bounded S-coverable labelled net system from a product system with globals.

**Definition 16 (Product system with globals to Net).** *Given a product system  $A = \langle A_1, A_2, \dots, A_k \rangle$  over distribution  $\Sigma$ , we can produce a net system  $(N = (S, T, F, \lambda), M_0, \mathcal{G})$  as follows:*

- $S = \bigcup_i P_i$ , the set of places.
- $T = \bigcup_a \text{globals}(a)$ , for all actions  $a$  in  $\Sigma$ . We also let  $T_i = \{\lambda^{-1}(a) \mid a \in \Sigma_i\}$ .
- The labelling function  $\lambda$  labels by action  $a$  the transitions in  $\text{globals}(a)$ .
- The flow relation  $F = \{(p, g), (g, q) \mid g \in T_a, g[i] = \langle p, a, q \rangle, i \in \text{loc}(a)\}$ . Let  $F_i$  be its restriction to the transitions  $T_i$  for  $i \in \text{loc}(a)$ .
- $M_0 = \{p_1^0, \dots, p_k^0\}$ , the initial product state.
- $\mathcal{G} = G$ , the set of final product states.

Since, the set of transitions of resultant net is same as the set of moves in the product system; and construction preserves pre as well as post places, we get one to one correspondence between reachable states of product system and reachable markings of nets.

**Lemma 1.** *In the construction of net system  $N$  in Definition 16,  $N$  is S-coverable, the construction preserves language, i.e.,  $\text{Lang}(N, M_0, \mathcal{G}) = \text{Lang}(A)$ .*

When we apply the generic construction above to product systems, with same source property, we get a free choice net, because any two global moves having same set of pre-places are put into one cluster.

**Theorem 1.** *Let  $(N, M_0, \mathcal{G})$  be the net system constructed from product system  $A$  as in Definition 16. If  $A$  has same source property then  $N$  is a free choice net.*

Even if a net is 1-bounded and S-coverable each component need not have only one token in it, but when we say that a 1-bounded net is S-coverable we assume that each component has one token. For live and 1-bounded free choice nets, such S-covers can be guaranteed [5]. Now we describe a linear-size construction of a product system from a net which is S-coverable.

**Definition 17 (Net to product system with globals).** *Given  $(N, M_0, \mathcal{G})$  a 1-bounded S-coverable labelled net system, 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  $\{1, 2, \dots, k\}$ , we define a product system  $A = \langle A_1, \dots, A_k \rangle$ .*

- $P_i = S_i, p_i^0$  the unique state in  $M_0 \cap P_i$ .
- $\rightarrow_i = \{\langle p, \lambda(t), p' \rangle \mid t \in T_i \text{ and } (p, t), (t, p') \in F_i, \text{ for } p, p' \in P_i\}$ .
- So we get sequential systems  $A_i = \langle P_i, \rightarrow_i, p_i^0 \rangle$  and the product system  $A = \langle A_1, A_2, \dots, A_k \rangle$  over alphabet  $\Sigma$ .
- $\text{globals}(a) = \bigcup_{t \in T_a} (\Pi_{i \in \text{loc}(a)} t[i])$ .
- $G = \mathcal{G}$

Again we can generalize results obtained in the thesis [20].

**Lemma 2.** *For net  $N$ , the construction of the product system  $A$  in Definition 17 above preserves language.*

For each  $a$ -labelled transition of the net we get one global  $a$ -move in the product system having same set of pre-places and post-places. And, by definition for each global  $a$ -move in product system we have an  $a$ -labelled transition in the net having same pre and post-places. This was ensured by “distributed choice” property [21, 20], here now ensured by construction using globals. We get one to one correspondence between reachable states of product system and reachable markings of the net we started with. Therefore, if we begin with the free choice net, we get same source property in the product system obtained.

**Theorem 2.** *Let  $(N, M_0, \mathcal{G})$  be a 1-bounded, S-coverable labelled free choice net system. Then one can construct a product system  $A$  with same source property.*

*Example 6.* From the net in Figure 1 using its S-decomposition shown in Figure 2, we get the product system of Figure 6 for the language of net. In this example, starting with this product system we get the same net back.

## 4 Expressions

### 4.1 Regular expressions and their properties

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 language of constant 0 is  $\emptyset$  and that of 1 is  $\{\varepsilon\}$ . For a symbol  $a \in \Sigma_i$ , its language is  $Lang(a) = \{a\}$ . For regular expressions  $s_1 + s_2$ ,  $s_1 \cdot s_2$  and  $s_1^*$ , its languages are defined inductively as union, concatenation and Kleene star 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—called derivatives—associated with regular expressions which are known since the time of Brzozowski [2], Mirkin [17] and Antimirov [1].

For each regular expression  $s$  over  $\Sigma_i$ , let  $Lang(s)$  be its language and 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. Similarly, we can syntactically check whether the empty word  $\varepsilon \in Lang(s)$ .

Antimirov derivatives [1] are defined as given below.

**Definition 18 ([1]).** *Given regular expression  $s$  and symbol  $a$ , the set of partial derivatives of  $s$  with respect to  $a$ , written  $Der_a(s)$  are defined as follows.*

$$\begin{aligned} Der_a(0) &= \emptyset \\ Der_a(1) &= \emptyset \\ Der_a(b) &= \{1\} \text{ if } b = a \quad \emptyset \text{ otherwise} \\ Der_a(s_1 + s_2) &= Der_a(s_1) \cup Der_a(s_2) \\ Der_a(s_1^*) &= Der_a(s_1) \cdot s_1^* \\ Der_a(s_1 \cdot s_2) &= \begin{cases} Der_a(s_1) \cdot s_2 \cup Der_a(s_2) & \text{if } \varepsilon \in Lang(s_1) \\ Der_a(s_1) \cdot s_2 & \text{otherwise} \end{cases} \\ \text{Inductively } Der_{aw}(s) &= Der_w(Der_a(s)). \end{aligned}$$

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

We have the Antimirov derivatives  $Der_a(ab + ac) = \{b, c\}$  and  $Der_a(a(b + c)) = \{b + c\}$ , whereas the Brzozowski  $a$ -derivative [2] (which is used for constructing deterministic automata, but which we do not use in this paper) for both expressions would be  $\{b + c\}$ .

A derivative  $d$  of  $s$  with global  $a \in Init(d)$  is called an  **$a$ -site** of  $s$ . An expression is said to have **equal choice** if for all  $a$ , its  $a$ -sites have the same set of initial actions. For a set  $D$  of derivatives, we collect all initial actions to form  $Init(D)$ . Two sets of derivatives have equal choice if their  $Init$  sets are same.

As in [21] we put together derivatives which may correspond to the same state in a finite automaton.

**Definition 19 ([21]).** *Let  $s$  be a regular expression and  $L = Lang(s)$ . For a set  $D$  of  $a$ -sites of regular expression  $s$  and an action  $a$ , we define the relativized*

language  $L_a^D = \{xay \mid xay \in L, \exists d \in \text{Der}_x(s) \cap D, \exists d' \in \text{Der}_{ay}(d) \text{ with } \varepsilon \in \text{Lang}(d')\}$ , and the prefixes  $\text{Pref}_a^D(L) = \{x \mid xay \in L_a^D\}$ , and the suffixes  $\text{Suf}_a^D(L) = \{y \mid xay \in L_a^D\}$ . We say that the derivatives in set  $D$  ***a-bifurcate***  $L$  if  $L_a^D = \text{Pref}_a^D(L) \cdot a \cdot \text{Suf}_a^D(L)$ .

We use partitions [21] of the  $a$ -sites of  $s$  into blocks such that each block (that is, element of the partition)  $a$ -bifurcates  $L$ . We have reproduced this definition in Appendix A. For an action  $a$ , let  $\text{Part}_a(s)$  denote such a partition.

**Proposition 1 ([21]).** *Every block  $D$  of the partition  $\text{Part}_a(s)$   $a$ -bifurcates language of a regular expression  $s$ .*

In comparison with our earlier paper [21], the next Definition 20 and Proposition 2 deal with “ $a$ -effects”, which are new. In addition to thinking of blocks of the partition as places of an automaton, we can now think of blocks and their effects as local moves.

**Definition 20.** *Given an action  $a$ , a set of  $a$ -sites  $B$  of regular expression  $s$  and a specified set of  $a$ -effects  $E \subseteq \text{Der}_a(B)$ , we define the relativized languages*

$$L_a^{(B,E)} = \{xay \in L \mid \exists d \in \text{Der}_x(s) \cap B, \exists d' \in \text{Der}_a(d) \cap E, \text{ and } \exists d'' \in \text{Der}_y(d') \text{ with } \varepsilon \in \text{Lang}(d'')\}.$$

Also define the prefixes  $\text{Pref}_a^{(B,E)}(L) = \{x \mid xay \in L_a^{(B,E)}\}$  and the suffixes  $\text{Suf}_a^{(B,E)}(L) = \{y \mid xay \in L_a^{(B,E)}\}$ . We say that a tuple  $(B, E)$  ***a-funnels***  $L$  if  $L_a^{(B,E)} = \text{Pref}_a^{(B,E)}(L) \cdot a \cdot \text{Suf}_a^{(B,E)}(L)$ . In such a pair  $(B, E)$ , if  $B$  is a block in the  $\text{Part}_a(s)$  and  $E$  is a nonempty subset of  $a$ -effects of  $B$ , then it is called as an ***a-duct***. For an  $a$ -duct  $(B, E)$ , we define its set of initial actions  $\text{Init}(B, E)$  as  $\text{Init}(B, E) = \text{Init}(B)$ , call  $B$  as its ***pre-block*** and call  $E$  as its ***post-effect***. For all  $i$  in  $\text{loc}(a)$  let  $a$ -***ducts*** $(s_i)$  denote the set of all  $a$ -ducts of regular expression  $s_i$ . For any two  $a$ -ducts  $(B, E)$  and  $(B', E')$  in  $a$ -ducts $(s_i)$ , define  $(B, E) = (B', E')$  if  $B = B'$  and  $E = E'$ .

*Example 7.* Consider a regular expression  $p_1 = (aaa)^*aaa$ . Let  $L$  denote the language of  $p_1$ . The set of all its derivatives is  $\text{Der}(p_1) = \{p_1, p_2 = aa(aaa)^*aaa, p_3 = a(aaa)^*aaa, p_4 = aa, p_5 = a, p_6 = \varepsilon\}$ . All these derivatives are also shown in Figure 5 used in Example 3. The set of  $a$ -sites of  $p_1$  consist of all its derivatives except expression  $p_6$ . A partition of  $a$ -sites of  $p_1$  is  $\text{Part}_a(p_1) = \{D_1 = \{p_1\}, D_2 = \{p_2, p_4\}, D_3 = \{p_3, p_5\}\}$ . See that each block in this partition  $a$ -bifurcates language  $L$ . For tuple  $(D_2, \{p_3\})$  its set of  $a$ -effects is  $\{p_3, p_5\}$ . The tuple  $(D_2, \{p_3\})$   $a$ -funnels  $L$  therefore it is an  $a$ -duct, having  $D_2$  as its pre-block and  $\{p_3\}$  as its post-effect. Another example of an  $a$ -duct is  $(D_3, \{p_1\})$ .

Consider the set  $Z = \{p_1, p_3\}$  of derivatives of  $p_1$ . This set  $Z$  does not  $a$ -bifurcate language  $L$ , because for the word  $aaaaaa$  in  $L$ , we have string  $aaa$  in  $\text{Suf}_a^{p_3}(L)$  (and hence in  $\text{Suf}_a^Z(L)$ ) and we have string  $\varepsilon$  in  $\text{Pref}_a^{p_1}(L)$  (and hence in  $\text{Pref}_a^Z(L)$ ), but the word  $\varepsilon \cdot a \cdot aaa$  is not in  $L$ . For set  $Z$ , we have

$Der_a(Z) = \{p_2, p_4, p_1\}$ . Now consider a set  $Y = \{p_1, p_2\} \subseteq Der_a(Z)$  of  $a$ -effects of  $Z$ . Considering the same string as above, we have  $\varepsilon$  in  $Pref_a^{(p_1, p_2)}(L)$  (and hence in  $Pref_a^{(Z, Y)}(L)$ ), and string  $aaa$  in  $Suf_a^{(p_3, p_1)}(L)$  (and hence in  $Suf_a^{(Z, Y)}(L)$ ). But the word  $\varepsilon \cdot a \cdot aaa$  is not in  $L$ , and hence not in  $L_a^{(Z, Y)}$ . Therefore, tuple  $(Z, Y)$  do not  $a$ -funnel  $L$  and is not an  $a$ -duct.

**Proposition 2.** *Every  $a$ -duct of a regular expression  $s$ ,  $a$ -funnels  $Lang(s)$ .*

We give following remark, using Proposition 2, for singleton set of derivatives.

*Remark 3.* Let  $r$  be an  $a$ -site of a regular expression  $s$ . It  $a$ -bifurcates  $Lang(s)$ , and for any of its  $a$ -effect  $E$ , tuple  $(r, E)$   $a$ -funnels  $Lang(s)$ .

Consider a regular expression  $s$  in the context of a distribution  $(\Sigma_1, \dots, \Sigma_k)$ , so that some of the actions are global. The following property of expressions has been related to an important property of product systems which enables us to identify places coming from a cluster in the free choice net.

**Definition 21 ([21]).** *If for all global actions  $a$  occurring in  $s$ , the partition  $Part_a(s)$  consists of a single block, then we say  $s$  has **unique sites**.*

## 4.2 Connected expressions

**Connected expressions over a distributed alphabet** This is the syntax of connected expressions defined over a distribution  $(\Sigma_1, \Sigma_2, \dots, \Sigma_k)$  of alphabet  $\Sigma$ .

$$e ::= 0 \mid \text{fsync}(s_1, s_2, \dots, s_k), \text{ where } s_i \text{ is a regular expression over } \Sigma_i$$

When  $e = \text{fsync}(s_1, s_2, \dots, s_k)$  and  $I \subseteq Loc$ , let the projection  $e \downarrow I = \prod_{i \in I} s_i$ .

**Definition 22.** *A connected expression  $e = \text{fsync}(s_1, s_2, \dots, s_k)$  over  $\Sigma$ , is said to have **equal choice** if, for all global actions  $a$  in  $\Sigma$  and for any  $i, j$  in  $loc(a)$ , any  $a$ -site of  $s_i$  have same  $Init$  set as of any  $a$ -site of  $s_j$ .*

*A connected expression  $e = \text{fsync}(s_1, s_2, \dots, s_k)$  over  $\Sigma$ , is said to have **unique sites** if, each component regular expression  $s_i$  have unique sites property.*

For a connected expression defined over distributed alphabet its derivatives and semantics were given in [21], are given below.

For the connected expression  $0$ , we have  $Lang(0) = \emptyset$ . For the connected expression  $e = \text{fsync}(s_1, s_2, \dots, s_k)$ , its language is given by

$$Lang(e) = Lang(s_1) \parallel Lang(s_2) \parallel \dots \parallel Lang(s_k),$$

where the synchronized shuffle  $L = L_1 \parallel \dots \parallel L_k$  is defined by

$$w \in L \text{ iff for all } i \in \{1, \dots, k\}, w \downarrow_{\Sigma_i} \in L_i.$$

The definitions of derivatives extended to connected expressions [21] is given below. The expression  $0$  has no derivatives on any action. Given an expression  $e = \text{fsync}(s_1, s_2, \dots, s_k)$ , its derivatives are defined by induction using the derivatives of the  $s_i$  on action  $a$ :

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

**Connected expressions with cables** We now define some properties of connected expressions over a distribution. These extend the notion of pairing given earlier, and will ultimately lead us to construct free choice nets without distributed choice.

**Definition 23.** Let  $e = \text{fsync}(s_1, s_2, \dots, s_k)$  be a connected expression over  $\Sigma$ . For each action  $a$  in  $\Sigma$ , we define the set  $a\text{-cables}(e) = \prod_{i \in \text{loc}(a)} a\text{-ducts}(s_i)$ . For an action  $a$ , an  $a\text{-cable}$  is an element of the set  $a\text{-cables}(e)$ . We say that a block  $B$  of  $\text{Part}_a(s_i)$  **appears** in an  $a\text{-cable}$   $D$  if there exists  $j$  in  $\text{loc}(a)$  and there exists  $Y \subseteq Der_a(B)$  such that  $D[j] = (B, Y)$ , i.e. if  $B$  is a pre-block of a component  $a\text{-duct}$  of  $D$ . For any  $a\text{-cable}$   $D$ , its set of pre-blocks  $\bullet D = \cup_{i \in \text{loc}(a)} \{B_i \mid B_i \text{ appears in } D\}$ , i.e. the set of pre-blocks of all the of its component  $a\text{-ducts}$ .

For expression  $e$ , let  $\mathbf{cables}(a) \subseteq a\text{-cables}(e)$ , such that for all  $i$  in  $\text{loc}(a)$

1. for all  $(B, E)$  in  $a\text{-ducts}(s_i)$ , there exists  $D$  in  $\mathbf{cables}(a)$ , such that there exists  $j$  in  $\text{loc}(a)$ , such that  $D[j] = (B, E)$ , i.e. each  $a\text{-duct}$  of  $s_i$  appears in at least one  $a\text{-cable}$  of it.
2. for all  $(B, E)$  and  $(B', E')$  in  $a\text{-ducts}(s_i)$  with  $(B, E) \neq (B', E')$ , if  $B = B' \implies E \cap E' = \emptyset$ , i.e. if any two distinct  $a\text{-ducts}$  of  $s_i$  appearing in it have same pre-block then, they must have disjoint post-effects.

A **connected expression with cables** is a connected expression with relations  $\mathbf{cables}(a)$  of it, for each global action  $a$  in  $\Sigma$ .

For connected expressions with cables, its derivatives are defined as:

**Definition 24.** The connected expression  $0$  has no derivatives on any action. For expression  $e = \text{fsync}(s_1, s_2, \dots, s_k)$ , we define its derivatives on action  $a$ , by induction, using  $a\text{-ducts}$  and the derivatives of  $s_j$  as:

$$Der_a(e) = \{\text{fsync}(r_1, r_2, \dots, r_k) \mid r_j \in Der_a(s_j) \text{ if there exists an } a\text{-cable } D \text{ in } \mathbf{cables}(a) \text{ such that, for all } j \text{ in } \text{loc}(a), s_j \text{ is in pre-block } B_j \text{ and } r_j \text{ is in } X_j \text{ of } a\text{-duct } D[j] = (B_j, X_j) \text{ of } s_j, \text{ otherwise } r_j = s_j\}.$$

We use the word **derivative** for expressions such as  $d = \text{fsync}(r_1, \dots, r_k)$  above (essentially tuples of derivatives of regular expressions), and  $d[i]$  for  $r_i$ . Define  $\text{Init}(d)$  to be those actions  $a$  such that  $Der_a(d)$  is nonempty. If  $a \in \text{Init}(d)$  we call  $d$  an  $a\text{-site}$ . The **reachable derivatives** are  $Der(e) = \{d \mid d \in Der_x(e), x \in \Sigma^*\}$ .

Language of  $e$  is the set of words over  $\Sigma$  defined using derivatives as below.

$$\text{Lang}(e) = \{w \in \Sigma^* \mid \exists e' \in Der_w(e) \text{ such that } \varepsilon \in \text{Lang}(r_i), \text{ where } e'[i] = r_i\}.$$

So we can have next derivative on action  $a$ , if it is allowed by the  $cables(a)$  relation. This is different from derivatives defined previously [21], when it is necessary to take derivatives of all component regular expressions having  $a$  in its alphabet. The number of derivatives can be exponential in  $k$ .

**Definition 25.** *A connected expression is said to have **equal source property** if for any pair of two cables sharing a common pre-block have same set of pre-blocks.*

*Example 8.* Let  $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$  be a distributed alphabet. Let  $e = fsync((ab + ac)^*, (ad + ae)^*)$  be a connected expression defined over  $\Sigma$ . Here,  $r_1 = (ab + ac)^*$  is a regular expression defined over component alphabet  $\Sigma_1$ , with language  $L_1 = Lang(r_1)$  and,  $s_1 = (ad + ae)^*$  is defined over  $\Sigma_2$  with language  $L_2 = Lang(s_1)$ . The set of derivatives of  $r_1$  is  $Der(r_1) = \{r_1, r_2 = br_1, r_3 = cr_1\}$ . Its set of  $a$ -sites consist of  $r_1$  itself. The partition of  $a$ -sites is  $Part_a(r_1) = \{D = \{r_1\}\}$ . The set of  $a$ -effects of  $D$  is  $Der_a(D) = \{r_2, r_3\}$ . For expression  $r_1$  its set of  $a$ -ducts is  $\{(D, \{r_2\}), (D, \{r_3\}), (D, \{r_2, r_3\})\}$ . The set of derivatives of  $s_1$  is  $Der(s_1) = \{s_1, s_2 = ds_1, s_3 = es_1\}$ . Its set of  $a$ -sites consist of  $s_1$  only. The partition of  $a$ -sites is  $Part_a(s_1) = \{D' = \{s_1\}\}$ . The set of  $a$ -effects of  $D'$  is  $Der_a(D') = \{s_2, s_3\}$ . And we get  $a$ -ducts  $\{(D', \{s_2\}), (D', \{s_3\}), (D', \{s_2, s_3\})\}$ . For global action  $a$ , set of partitions  $Part_a(r_1)$  and  $Part_a(s_1)$  has only one block, hence both component expressions have unique sites property and therefore connected expression  $e$  also has it.

For global action  $a$ , there are many possible  $cables(a)$  relations for expression  $e$ . One such relation is  $\{((D, \{r_2\}), (D', \{s_2\})), ((D, \{r_3\}), (D', \{s_3\}))\}$ . See that each  $a$ -duct appears at least once in this relation, and for two  $a$ -ducts of  $r_1$  with the same pre-block  $D$  we have that their set of post-effects  $\{r_2\}$  and  $\{r_3\}$  are disjoint. Similarly this condition holds for  $a$ -ducts of  $s_1$ . For both  $a$ -cables set of pre-blocks is identical, therefore  $cables(a)$  satisfies equal source property.

Since the only global action is  $a$ , we have only one cables relation associated with expression  $e$ . Also see that since each component of expression  $e$  have only one block in its partition of  $a$ -sites, hence it satisfies unique sites property. Now with respect to the  $cables(a)$  relation given above,  $Der_a(e) = \{fsync(r_2, s_2), fsync(r_3, s_3)\}$ , but expression  $fsync(r_2, s_3)$  is not in  $Der_a(e)$ , because only post-effect of  $D$  containing  $r_2$  is the set  $\{r_2\}$  and similarly, only post-effect of  $D'$  containing  $s_3$  is the set  $\{s_3\}$  and there does not exist an  $a$ -cable with  $(D, \{r_2\})$  and  $(D', \{s_3\})$  as its components. The set of derivatives of expression  $e$  is  $Der(e) = \{e, (r_2, s_2), (r_3, s_3), (r_1, s_2), (r_1, s_3), (r_2, s_1), (r_3, s_1)\}$ .

## 5 Connected Expressions and Product Systems

In this section we prove two main theorems of the paper. For obtaining a connected expression from a product system with globals defined over a distributed alphabet, we go through an intermediate product system defined over a new distributed alphabet, get a language equivalent intermediate expression defined over this new alphabet using results from [22] and then finally rename this to



get an expression for language of original product system. To obtain a product system from a connected expression we use above steps in the reverse direction. All proofs missing from this section are relegated to Appendix B.

### 5.1 Analysis of Expressions from Systems

In this section we produce expressions for our systems using a result from [22].

**Lemma 3 ([22]).** *Let  $A$  be a product system with separation of labels. Then we can compute a connected expression for the language of  $A$ , where every regular expression has unique sites. If the product was conflict-equivalent, the constructed expression has equal choice.*

The Lemma 4, allows us to relate product systems with globals having separation of labels property defined over a distribution to connected expressions with cables having unique sites property.

Here we present the key idea behind it, formal proof is in the Appendix B, Let  $A_i$  be a sequential automaton having an  $a$ -move  $(p, a, q)$  where  $a$  is a global action. Let  $s$  be a regular expression for the language of  $A_i$ . Our aim is to get an  $a$ -duct  $(B_p, a, B_q)$  where  $B_p$  is the set of derivatives of expression  $s$ , which corresponds to place  $p$  and, similarly  $B_q$  is the set of derivatives which corresponds to place  $q$ , such that any string  $xay$  passing through place  $p$  and  $q$  should also pass through derivatives belonging to  $B_p$  and  $B_q$ . If  $A_i$  has separation of labels property, then we know that  $p$  is the unique place having outgoing  $a$ -moves in the automaton. Hence, we can apply Lemma 3 and get the set  $B_p$  as required. Now we turn to get  $B_q$  for place  $q$ . If  $q$  has an outgoing  $c$ -move for some global action  $c$ , then our task becomes easy and we can apply Lemma 3 to get  $B_q$ . If  $c$  is not a global action then we can not apply Lemma 3 directly because there might be many other places like  $q$  which may have outgoing  $c$ -moves. Therefore, in order to make use of Lemma 3 we need to make  $q$  as a unique place. This we do so by changing alphabet of  $A_i$  in the following way. If  $\langle q, c, r \rangle$  was a  $c$ -move in  $A_i$  then we rename  $c$  by  $c_q$  and we have local move  $\langle q, c_q, r \rangle$  in this new automaton  $A'_i$  defined over a new alphabet and let expression  $s'$  denote language of  $A'_i$ . Now  $q$  is a unique place having outgoing  $c_q$ -moves and we can apply Lemma 3 to get a set of derivatives  $B'_q$  of expression  $s'$  which corresponds to place  $q$  in  $A'_i$ . Now we rename back these derivatives in  $B'_q$  to get the set  $B_q$  which is a subset of  $Der(s)$ , and corresponds to the place  $q$  in the original automaton  $A_i$ . Now to show that  $a$ -move  $(p, a, q)$  is indeed captured by  $(B_p, a, B_q)$  we need prove that any word accepted by automaton and passing through place  $p$  and  $q$  should also pass through derivatives belonging to these sets, and the reverse of this should also hold.

**Lemma 4.** *Let  $\Sigma$  be a distributed alphabet. Let  $A$  be a product system over  $\Sigma$  with globals, having separation of labels and same source property. Then we can compute a connected expression  $e$  with cables for the language of  $A$ , which has unique sites and equal source property. Moreover, for each global action  $a$  in  $\Sigma$ , we produce an  $a$ -duct for each local  $a$ -move; and, one  $a$ -cable of expression  $e$  for each global  $a$ -move of product  $A$ .*

Now we use Lemma 4 to get expressions for the product systems with globals.

**Theorem 3.** *Let  $A$  be a product system with globals, having same source property, defined over  $\Sigma$ . Then for the language of  $A$ , we can compute a connected expression with cables, having equal source property.*

## 5.2 Synthesis of Systems from Expressions

Using the construction of Antimirov [1], which in polynomial time gives us a finite automaton of size  $O(wd(s))$  for a regular expression  $s$  using partial derivatives as states, we produce product automata for our expressions.

**Lemma 5 ([22]).** *Let  $e$  be a connected expression with unique sites. Then there exists a product system  $A$  with separation of labels accepting  $\text{Lang}(e)$  as its language. If  $e$  has equal choice, then  $A$  is conflict-equivalent.*

We use Lemma 5 to prove Lemma 6, which allows us to interrelate connected expressions with cables defined over a distribution and connected expression defined over a distributed alphabet.

**Lemma 6.** *Let  $\Sigma$  be a distributed alphabet. Let  $e$  be a connected expression with cables, having equal source property and unique sites. Then we can compute a product system with globals for  $\text{Lang}(e)$ , and having separation of labels along with same source property. Moreover, for each  $a$  in  $\Sigma$ , for each  $a$ -duct we produce one local  $a$ -move. And, for each  $a$ -cable we get one global  $a$ -move of  $A$ .*

Next we present construction of products with globals having same source property from connected expressions with cables, having equal source property.

**Theorem 4.** *Let  $e$  be a connected expression with cables, having equal source property, defined over  $\Sigma$ . Then for the language of  $e$ , we can compute a product system with globals having same source property.*

*Proof.* Given a connected expression  $e$  with cables, defined over distributed alphabet  $\Sigma$ , may not have unique sites property. So in the first part of proof, we use cables of  $e$  to do the appropriate renaming to get an expression  $e'$  with cables, over a new alphabet  $\Sigma'$ , and having unique sites property. So we can apply Lemma 6 on  $e'$  to get a language equivalent product system  $A'$  with globals and having separation of labels property. In the last part of proof we rename back the local moves in  $A'$  to get a product system  $A$  with globals for the language of original expression  $e$  with cables.

Since expression  $e$  has equal source property, the set of  $\text{cables}(a)$  can be partitioned into buckets such that two  $a$ -cables belong to a bucket iff they have same set of pre-blocks. Because of same source property of expression  $e$ , an  $a$ -duct can appear as a component of many  $a$ -cables belonging to same bucket. But it can not be a component of two  $a$ -cables belonging to two different buckets. Without loss of generality assume that  $\text{loc}(a) = \{1, \dots, l\}$ . Let  $(B_1, \dots, B_l)$  be the pre-tuple of  $a$ -cables of bucket  $Z$  of the partition of  $\text{cables}(a)$ . For each  $i$

in  $loc(a)$ , replace each occurrence of letter  $a$  in expression  $s_i$ , corresponding to  $Init(B_i)$  by  $a^Z$ , to get an expression  $s'_i$  over alphabet  $\Sigma'_i$ .

So we get connected expression  $e' = (s'_1, \dots, s'_k)$  over distributed alphabet  $\Sigma' = (\Sigma'_1, \dots, \Sigma'_k)$ . This expression has unique sites property, and continues to have equal source property. Each  $a$ -duct of  $s_i$ , corresponds to some  $a^Z$ -duct of  $s'_i$ , and each  $a$ -cable to some  $a^Z$ -cable of  $e'$ . Note that there might be many  $a^Z$ -cables .

For language of  $e'$ , we get product system  $A' = \langle A'_1, \dots, A'_k \rangle$  by applying Lemma 6 over distributed alphabet  $\Sigma'$  having separation of labels. For each  $a^Z$ -duct we get an  $a^Z$ -move and for each  $a^Z$ -cable we get an  $a^Z$ -global in  $A'$ . It has same source property since  $e'$  had equal source property.

Replacing each local action  $\langle p, a^Z, q \rangle$  of  $A'_i$  by  $\langle p, a, q \rangle$  we get  $A_i$ , for each  $i$  in  $loc(a)$ . We repeat this for all global actions, to get product system  $A$  over  $\Sigma$ . Hence, for each  $a^Z$ -global of system  $H$ , we get an  $a$ -global for system  $A$ , which continues to have same source property. Since renamings are well defined in first and last part of the proof we get that  $Lang(e) = Lang(A)$ .  $\square$

*Example 9.* Let  $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$  be a distributed alphabet. Consider a connected expression  $e = fsync(r_1, s_1)$  from Example 8 defined over  $\Sigma$ , where  $r_1 = (ab + ac)^*$  and  $s_1 = (ad + ae)^*$ . Considered  $cables(a)$  relation for the connected expression  $e$  is  $\{((D, \{r_2\}), (D', \{s_2\})), ((D, \{r_3\}), (D', \{s_3\}))\}$ . As shown in Example 8 expression  $e$  has both unique sites and equal source property. Therefore, first step of construction in the proof of Theorem 4, is trivially done. After which, we apply Lemma 6 to get a product system which has separation of labels and same source property. Also see that for  $a$ -duct  $(D, \{r_2\})$  we get local  $a$ -move  $(D, a, \{r_2\})$  in  $\rightarrow_1$  and for  $a$ -duct  $(D', \{s_3\})$  we get local  $a$ -move  $(D', a, \{s_3\})$  in  $\rightarrow_2$ . For  $a$ -cable  $((D, \{r_2\}), (D', \{s_2\}))$  of expression  $e$  we get an  $a$ -global  $((D, \{r_2\}), a, (D', \{s_2\}))$ . Similarly for  $a$ -cable  $((D, \{r_3\}), (D', \{s_3\}))$ ,  $a$ -global is  $((D, \{r_3\}), a, (D', \{s_3\}))$ . This product system with globals, for the expression  $e$  with cables is shown in Figure 6.

*Example 10.* Let  $e = fsync(r_1, s_1)$  be a connected expression defined over distributed alphabet  $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$ , where,  $r_1 = ((ab + ac)(ab + ac))^*$  and  $s_1 = (((ad + ae)(ad + ae))^*$ . Derivatives of components are  $Der(r_1) = \{r_1, r_2 = b(ab + ac)r_1, r_3 = c(ab + ac)r_1, r_4 = (ab + ac)r_1, r_5 = br_1, r_6 = cr_1, \}$  and  $Der(s_1) = \{s_1, s_2 = d(ad + ae)s_1, s_3 = e(ad + ae)s_1, s_4 = (ad + ae)s_1, s_5 = ds_1, s_6 = es_1, \}$ . This expression do not have unique sites, as componets do not have it. Partitions are  $Part_a(r_1) = \{r_1, r_4\}$  and  $Part_a(s_1) = \{s_1, s_4\}$ . A set  $cables(a) = \{d_1 = ((r_1, r_2), (s_1, s_2)), d_2 = ((r_1, r_3), (s_1, s_3)), d_3 = ((r_4, r_5), (s_4, s_5)), d_4 = ((r_4, r_6), (s_4, s_6))\}$ . See that each  $a$ -duct appears at least once in this relation, and for two  $a$ -ducts  $d_1[1]$  and  $d_2[1]$  of  $r_1$  with the same pre-block  $r_1$  we have that their set of post-effects  $\{r_2\}$  and  $\{r_3\}$  are disjoint and similarly it holds for  $d_3[1]$  and  $d_4[1]$  of  $r_1$  with the same pre-block  $r_4$  we have that their set of post-effects  $\{r_5\}$  and  $\{r_6\}$  are disjoint. Similarly this condition holds for  $a$ -ducts of  $s_1$ . It has equal source property as  $\bullet d_1 = \bullet d_2 = \{r_1, s_1\}$  and  $\bullet d_3 = \bullet d_4 = \{r_4, s_4\}$ .  $Der_a(e) = \{fsync(r_2, s_2), fsync(r_3, s_3)\}$ , but  $fsync(r_2, s_3)$  is not

in  $Der_a(e)$ . For simplifying notation, dropping  $fsync$  operator, the set of reachable derivatives is  $Der(e) = \{(r_1, s_1), (r_2, s_2), (r_4, s_2), (r_2, s_4), (r_4, s_4), (r_3, s_3), (r_4, s_3), (r_3, s_4), (r_5, s_5), (r_6, s_6), (r_5, s_1), (r_1, s_5), (r_6, s_1), (r_1, s_6)\}$ .

A  $cables(a)$  relation for  $e$  is A set  $cables(a) = \{d_1 = ((r_1, r_2), (s_1, s_2)), d_2 = ((r_1, r_3), (s_1, s_3)), d_3 = ((r_4, r_5), (s_4, s_5)), d_4 = ((r_4, r_6), (s_4, s_6))\}$ .

In the construction,  $a$ -cables  $d_1$  and  $d_2$  are put into one bucket  $z_1$  and  $d_3$  and  $d_4$  are put into bucket  $z_2$ . So the renamed expression is  $e' = fsync(((a^1b + a^1c)(a^2b + a^2c))^*, ((a^1d + a^1e)(a^2d + a^2e))^*)$ . This expression has unique sites. See that each  $a$ -duct also gets renamed for example for  $a$ -duct  $(r_1, r_2)$  we have an  $a^1$ -duct  $(r'_1, r'_2)$  and and for  $a$ -duct  $(s_4, s_5)$  we get an  $a^2$ -duct  $(s'_4, s'_5)$ , so on. So we have two sets of cables now for two global actions  $cables(a^1) = \{d'_1 = ((r'_1, r'_2), (s'_1, s'_2)), d'_2 = ((r'_1, r'_3), (s'_1, s'_3))\}$  and  $cables(a^2) = \{d'_3 = ((r'_4, r'_5), (s'_4, s'_5)), d'_4 = ((r'_4, r'_6), (s'_4, s'_6))\}$ . Now applying Lemma 6 we get a product system which has separation of labels and same source property. So for the  $a^1$ -duct  $(r'_1, r'_2)$  we get an  $a^1$ -move  $(r'_1, a^1, r'_2)$  and for  $a^2$ -duct  $(s'_4, s'_5)$  we get an  $a^2$ -move  $(s'_4, a^2, s'_5)$ , and so forth, in the constructed  $A'_i$ , where  $i$  in  $\{1, 2\}$ . This intermediate product system with globals is shown in Figure 7. Now renaming back  $a^1$  and  $a^2$  by action  $a$  in  $A'_i$  we get product system  $A = \langle A_1, A_2 \rangle$  for language of  $e$ . In this example starting with this product system  $A$ , using Theorem 3, we get the same expression back.

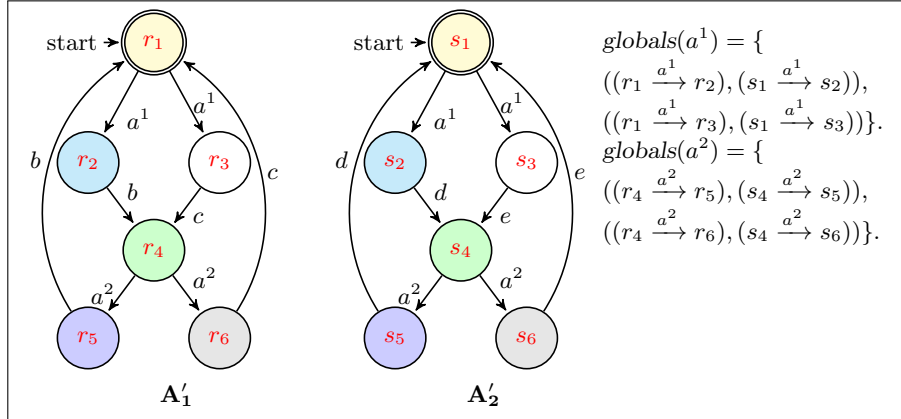


Fig. 7: Product system  $A' = \langle A'_1, A'_2 \rangle$  used in Example 10

## 6 Conclusion

We use Theorem 2 and Theorem 3 to get an expression for a labelled 1-bounded and S-coverable free choice net. The size of the intermediate product system is

linear in the size of net and size of the expression can be exponential in the size of product system. This exponential blow up in the size is expected because a component regular expression obtained from the corresponding component automaton can be of exponential size [10].

Using Theorem 4 and Theorem 1 we obtain labelled free choice nets from expression with cables and equal source property. The size of the intermediate product system is linear in the size of connected expression, as size of each component automaton obtained using Antimirov derivatives is linear in the size of component regular expression [1]. The size of net obtained is linear in the size of product system with globals. Previously [21], resultant net can have exponential number of transitions in the size of product system.

As a next step in this direction, we want to work with free choice nets which are labelled, where labels may not come from a distributed alphabet.

## References

1. Antimirov, V.: Partial derivatives of regular expressions and finite automaton constructions. *Theoret. Comp. Sci.* 155(2), 291–319 (1996)
2. Brzozowski, J.A.: Derivatives of regular expressions. *J. ACM* 11(4), 481–494 (1964)
3. Cheng, A., Esparza, J., Palsberg, J.: Complexity results for 1-safe nets. *Theor. Comput. Sci.* 147(1&2), 117–136 (1995)
4. Desel, J.: A proof of the rank theorem for extended free choice nets. In: *Application and Theory of Petri Nets*. pp. 134–153 (1992)
5. Desel, J., Esparza, J.: *Free choice Petri nets*. Cambridge University Press, New York, USA (1995)
6. Esparza, J., Silva, M.: A polynomial-time algorithm to decide liveness of bounded free choice nets. *Theoret. Comp. Sci.* 102(1), 185–205 (1992)
7. Garg, V.K., Ragunath, M.: Concurrent regular expressions and their relationship to petri nets. *Theoret. Comp. Sci.* 96(2), 285–304 (1992)
8. Grabowski, J.: On partial languages. *Fund. Inform.* IV(2), 427–498 (1981)
9. Grabowski, J.: On partial languages. *Fundam. Inform.* 4(2), 427–498 (1981)
10. Gruber, H., Holzer, M.: Finite automata, digraph connectivity, and regular expression size. In: *ICALP(2)*. pp. 39–50 (2008)
11. Hack, M.H.T.: *Analysis of production schemata by Petri nets*. Project Mac Report TR-94, MIT (1972)
12. Hopcroft, J.E., Motwani, R., Ullman, J.D.: *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley (2003)
13. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: *Automata studies*. pp. 3–41. Princeton (1956)
14. Kozen, D.: A completeness theorem for kleene algebras and the algebra of regular events. *Inf. Comput.* 110(2), 366–390 (1994)
15. Lodaya, K.: Product automata and process algebra. In: *SEFM*. pp. 128–136. IEEE (2006)
16. Lodaya, K., Mukund, M., Phawade, R.: Kleene theorems for product systems. In: *DCFS, Proceedings. LNCS, vol. 6808*. Springer (2011)
17. Mirkin, B.G.: An algorithm for constructing a base in a language of regular expressions. *Engg. Cybern.* 5, 110–116 (1966)

18. Mukund, M.: Automata on distributed alphabets. In: D'Souza, D., Shankar, P. (eds.) *Modern Applications of Automata Theory*. World Scientific (2011)
19. Phawade, R.: Direct product representation of labelled free choice nets. *Int. J. Comp. Appl.* 99(16), 1–8 (2014)
20. Phawade, R.: *Labelled Free Choice Nets, finite Product Automata, and Expressions*. Ph.D. thesis, Homi Bhabha National Institute (2015)
21. Phawade, R., Lodaya, K.: Kleene theorems for labelled free choice nets. In: Moldt, D., Rölke, H. (eds.) *Proc. PNSE. CEUR Workshop Proceedings*, vol. 1160, pp. 75–89. CEUR-WS.org (2014)
22. Phawade, R., Lodaya, K.: Kleene theorems for synchronous products with matching. *Trans. on Petri nets and other models of concurrency X*, 84–108 (2015)
23. Salomaa, A.: Two complete axiom systems for the algebra of regular events. *J. ACM* 13(1), 158–169 (1966)
24. Thiagarajan, P., Voss, K.: In praise of free choice nets. In: *European Workshop on Applications and Theory in Petri Nets*. pp. 438–454 (1984)
25. Zielonka, W.: Notes on finite asynchronous automata. *Inform. Theor. Appl.* 21(2), 99–135 (1987)

## Appendices

### A Partitions of Derivatives of Regular Expressions

Here we give below a syntactic scheme to generate a partition  $Part_a(s)$  of regular expression  $s$ , with respect to a symbol  $a$ .

**Definition 26 ([21]).** Let  $X_1$  be a partition of  $a$ -sites of  $s_1$  and  $X_2$  be a partition of  $a$ -sites of  $s_2$ , where regular expression  $s = s_1 \cdot s_2$  or  $s = s_1 + s_2$ . For partitions  $X_1, X_2$  with blocks  $D_1, D_2$  containing elements  $d_1, d_2$  respectively, we use the notation  $(X_1 \cup X_2)[d/d_1, d_2]$  for the modified partition  $((X_1 \setminus \{D_1\}) \cup (X_2 \setminus \{D_2\})) \cup \{(D_1 \cup D_2 \cup \{d\}) \setminus \{d_1, d_2\}\}$ . And, for partition  $X$  with block  $D_1$  in it, having  $d_1$  in it,  $X[d/d_1]$  is the modified partition  $X \setminus \{D_1\} \cup \{(D_1 \setminus \{d_1\}) \cup \{d\}\}$ .

$$\begin{aligned}
Part_a(b) &= \emptyset \text{ if } a \neq b \\
Part_a(a) &= \{\{a\}\} \\
Part_a(s_1^*) &= (Part_a(s_1) \cdot s_1^*)[s_1^*/s_1 \cdot s_1^*] \\
Part_a(s_1 + s_2) &= Z_1 \cup Z_2 \cup \{s_1 + s_2\} \text{ if } a \in \text{Init}(s_1 + s_2) \\
Part_a(s_1 \cdot s_2) &= \begin{cases} Part_a(s_1) \cdot s_2 \cup Part_a(s_2)[s_1 \cdot s_2/s_2] & \text{if } \varepsilon \in \text{Lang}(s_1) \\ & \text{and } \varepsilon \notin \text{Lang}(s_2) \\ Part_a(s_1) \cdot s_2 \cup Part_a(s_2) & \text{otherwise} \end{cases} \\
&\quad \text{where ,} \\
Z_1 &= Part_a(s_1) \setminus \{s_1\} \text{ if } s_1 \notin \text{Der}_a(s_1 + s_2), Part_a(s_1) \text{ otherwise} \\
Z_2 &= Part_a(s_2) \setminus \{s_2\} \text{ if } s_2 \notin \text{Der}_a(s_1 + s_2), Part_a(s_2) \text{ otherwise.}
\end{aligned}$$

### B Proofs from the Section 5

*Proof (Lemma 4).* Let  $A = \langle A_1, \dots, A_k \rangle$  be the given product system. The separation of labels property of  $A$  ensures that for a global action  $a$ , and for each  $i$  in  $loc(a)$ , we have a unique place in  $A_i$  having outgoing local  $a$ -moves; but it may have multiple places having outgoing  $c$ -moves for some local action  $c$  of  $\Sigma_i$ . And, same source property of  $A$  ensures that these places can not have any outgoing  $a$ -move for any global action  $a$ .

*(Stage 1 of proof):* Now, for all local actions  $c$  in  $\Sigma_i$ , we replace each local  $c$ -move  $\langle q, c, r \rangle$  of component  $A_i$  by  $c_q$ -move  $\langle q, c_q, r \rangle$  to get new alphabet  $\Gamma_i$  and new sequential system  $H_i$  over it. Repeating this for each sequential system  $A_i$  we get a new product system  $H = (H_1, \dots, H_k)$  over distributed alphabet  $\Gamma = (\Gamma_1, \dots, \Gamma_k)$ . For each global action  $a$ , the relation  $global(a)$  of product  $A$ , remains same for product  $H$ .

Since a local action is renamed on basis of the source place of the local move of which it is part of, this renaming is well defined. For any word  $w'$  in  $Lang(H_i)$  we get a unique word  $w$  in  $Lang(A_i)$  by renaming back local action  $c_q$  in  $w'$  by  $c$ . And in the reverse direction, for any  $w$  in  $Lang(A_i)$  we can replace a local action  $c$  by  $c_q$  using the run of  $A_i$  on  $w$  (in the case that  $A_i$  is at place  $q$  and some local  $c$ -move  $\langle q, c, r \rangle$  is used), so this word  $w'$  is unique. See that  $\Gamma_i$  and  $\Sigma_i$  have same set of global actions and they differ only in local actions they have.

Therefore, for any word  $u'$  in  $Lang(H)$  we get a unique word  $u$  in  $Lang(A)$  by renaming local actions  $c_q$  of  $u'$  to  $u$ .

For each local action  $c_q$  of  $\Gamma_i$  we have a unique place  $q$  in  $H_i$  having outgoing  $c_q$ -moves; thus, for action  $c$  (action  $c$  could be global or local) of  $\Gamma_i$  we have one unique place in  $H_i$  having outgoing  $c$ -moves. Also, for each place of  $H_i$  we have an action –local or global– such that the place has outgoing local moves on that action. The product  $H$  has separation of labels property in a more general way, as even local actions have unique places for them now; and, it has same source property since product  $A$  has it. Since product system  $H$  has both these properties, it is conflict-equivalent.

(*Stage 2 of proof*): Now we can apply Lemma 3, to get a connected expression  $e' = (s'_1, \dots, s'_k)$  over alphabet  $\Gamma$  for language of product  $H$  and it has unique sites property. Hence, for each (local or global) action  $c$  in  $\Gamma_i$  we have only one block in  $Part_c(s'_i)$ . If place  $p$  has outgoing local  $a$ -moves  $\langle p, a, q \rangle$  and  $\langle p, a, r \rangle$  in  $H_i$ , with place  $q$  having outgoing local  $c$ -moves and place  $r$  having outgoing local  $d$ -moves; then by application of Lemma 3, we would get unique blocks  $B'_p, B'_q$ , and  $B'_r$  in  $Part_a(s'_i), Part_c(s'_i)$  and  $Part_d(s'_i)$  respectively. And each of these blocks bifurcate  $Lang(s'_i)$ . We know that the sets  $B'_q$  and  $B'_r$  of  $a$ -effects of block  $B'_p$  are disjoint and their *Init* sets are also disjoint. Therefore, by Proposition 2 each  $a$ -duct  $(B'_p, B'_q)$  of  $s'_i$   $a$ -funnels  $Lang(s'_i)$ .

To show the correspondence between local  $a$ -move  $\langle p, a, q \rangle$  and the  $a$ -duct  $(B'_p, B'_q)$  obtained as above, we need to prove that  $L_a^{(p,q)} = L_a^{(B'_p, B'_q)}$ . As  $(p, q)$  and  $(B'_p, B'_q)$  both  $a$ -funnel language  $L$ , hence  $L_a^{(p,q)} = Pref_a^p(L) a Suf_a^{(p,q)}(L)$  and  $L_a^{(B'_p, B'_q)} = Pref_a^{B'_p}(L) a Suf_a^{(B'_p, B'_q)}(L)$ . Since place  $p$ , as well as block  $B'_p$ ,  $a$ -bifurcates  $L$ , we have  $Pref_a^p(L) = Pref_a^{B'_p}(L)$ . Therefore, it is sufficient to prove the equivalence  $Suf_a^{(p,q)}(L) = Suf_a^{(B'_p, B'_q)}(L)$ . Consider a word  $y$  in  $Suf_a^{(p,q)}(L)$  but we have  $y = c_q y'$  as  $q$  is the only place having outgoing  $c_q$  moves. Therefore  $y'$  is in  $Suf_a^q(L)$  and since  $q$ , as well as block  $B'_q$ ,  $c_q$ -bifurcates  $L$  we have  $y'$  in  $Suf_a^{B'_q}(L)$  also. But we also have word  $y = cy'$  in  $Suf_a^p(L)$  and, hence  $y$  in  $Suf_a^{B'_p}(L)$ . Therefore, we have  $cy'$  in  $Suf_a^{(B'_p, B'_q)}(L)$  as required.

Hence for each local  $a$ -move  $(p, a, q)$  of sequential system  $H_i$  we have an  $a$ -duct  $(B'_p, B'_q)$  of regular expression  $s'_i$ . Therefore, for each  $a$ -global of product  $H$  we get an  $a$ -cable of connected expression  $e'$  by taking tuple of  $a$ -ducts corresponding to its component  $a$ -moves. Since product  $H$  has same source property expression  $e'$  has equal source property.

(*Stage 3 of proof*): Now we rename back each local action  $c_q$  by  $c$  in each of  $s'_i$  to get regular expression  $s_i$  over alphabet  $\Sigma_i$ . See that we had not renamed any global action of  $\Sigma$ , and for such local action  $c_q$  its set of locations  $loc(c_q)$  is singleton. Hence we continue to have unique sites property for all global actions  $a$ . Repeating this for each regular expression  $s'_i$  in expression  $e'$ , we get a connected expression  $e = (s_1, \dots, s_k)$  over distributed alphabet  $\Sigma$ , having unique sites property. Since expression  $e'$  has equal source property expression  $e$  continues to have it.



Now we prove the claim that renaming a derivative of  $s'_i$  in this fashion, we get derivative of  $s_i$ . For the base case  $s_i$  is obtained by renaming  $s'_i$  (the fact that  $\varepsilon$  is in  $Lang(s'_i)$  iff  $\varepsilon$  is in  $Lang(s_i)$ , can be proved by structural induction on  $s_i$ ). Consider a word  $w' = x'ay'$  in  $Lang(s'_i)$ . So there exist derivatives  $d'$  in  $Der_{x'}(s'_i)$ , and  $r'$  in  $Der_{x'a}(s'_i)$ , and  $s'_f$  in  $Der_{y'}(r')$  such that  $\varepsilon$  is in  $Lang(s'_f)$ . By induction hypothesis we have derivative  $d$  of  $s_i$  obtained by renaming  $d'$ . If action  $a$  is global then  $Init(d) = Init(d')$  and if it is local action then  $Init(d)$  is obtained by renaming  $Init(d')$ . To complete the induction step, we need to prove that  $r$  is in  $Der_{xa}(s_i)$ , where  $x$  is obtained by renaming word  $x'$ . By induction hypothesis, derivative  $d$  is in  $Der_x(s_i)$ , therefore it suffices to prove that derivative  $r$ , the word obtained by renaming back  $r'$ , is in  $Der_a(d)$ . We proceed by doing structural induction on  $d'$ .

- The base cases of  $d$  being 0 or 1 or  $b \neq a$  are straightforward, as it does not involve renaming and  $r' = r$ .
- For  $d' = d'_1 \cdot d'_2$ , we have  $d = d_1 \cdot d_2$  where  $d_1$  (respectively  $d_2$ ) is obtained by renaming  $d'_1$  (respectively  $d'_2$ ). Consider the case where  $\varepsilon \notin Lang(d'_1)$ . By (structural) induction hypothesis (since  $d'_1$  being smaller than  $d'$ ), expression  $d_1$  also does not have  $\varepsilon$  in it. Let  $r'$  is in  $Der_a(d'_1) \cdot d'_2$ , that is  $r = r'_1 \cdot d'_2$  for some  $r'_1$  in  $Der_a(d'_1)$ . Now  $d'_1$  being smaller, by induction hypothesis we have that  $r_1$  (obtained by renaming  $r'_1$ ) is in  $Der_a(d_1)$ , similarly we have  $d_2$ . Hence we have  $r = r_1 \cdot d_2$  which belongs to  $Der_a(d_1)d_2$ , as required. In the case that  $\varepsilon$  is in  $Lang(d'_1)$ , by (structural) induction hypothesis, we have  $\varepsilon$  in  $Lang(d_1)$ . So  $r'$  is in  $Der_a(d'_1)d'_2$  or  $r'$  is in  $Der_a(d'_2)$ . In the latter case,  $d'_2$  being smaller than  $d'$  proof follows by using structural induction. In the former case we have  $r' = r'_1 d'_2$  where  $r'_1$  is in  $Der_a(d'_1)$ . Since  $d'_1$  smaller than  $d'$  using structural induction hypothesis we have  $r_1$  (renamed expression  $r'_1$ ) in  $Der_a(d_1)$  and similarly  $d_2$  is obtained by renaming  $d'_2$ . Therefore  $r = r_1 d_2$  is in  $Der_a(d)$ .
- $d' = d_1^*$  we have  $r'$  in  $Der_a(d_1^*)d'_1$ . So  $r' = r'_1 \cdot d'_1$ . By structural induction hypothesis we have  $r_1$  (obtained by renaming  $r'_1$ ) in  $Lang(d_1)$ . Therefore  $r = r_1 d_1$  is in  $Der_a(d_1^*)$ , as required.
- For  $d' = d'_1 + d'_2$ , we have  $d = d_1 + d_2$  and we want to prove that derivative  $r$  is in  $Der_a(d)$ , is in  $Der_a(d)$ . By definition of derivatives we have  $r'$  in  $Der_a(d'_1)$  or in  $Der_a(d'_2)$ . In the former case, by (structural) induction hypothesis we get  $r$  (renamed  $r'$ ) in  $Der_a(d_1)$  and in the latter case also expression  $r$  is in  $Der_a(d_2)$ .

Because of above claim, for any word  $w'$  in  $Lang(s'_i)$  we have a unique word  $w$  in  $Lang(s_i)$  obtained by renaming back. In the reverse direction, the claim that for any  $w$  in  $Lang(s_i)$  we get a unique word  $w'$  in  $Lang(s'_i)$  is proved in the similar manner. Using above claim, we can rename a set of derivatives; so for a set of derivatives  $B' \subseteq Der(s'_i)$  we get the set  $B \subseteq Der(s_i)$ , and vice versa.

The set  $X'_q$  of  $a$ -effects of block  $B'_p$ , might have been a unique block of  $Part_{c_q}(s'_i)$ , but after renaming, it might not  $c$ -bifurcate  $Lang(s_i)$ . On the other hand, since  $B_p$   $a$ -bifurcates  $Lang(s_i)$ , pair  $(B_p, X_q)$  of  $s_i$  continues to  $a$ -funnel

$Lang(s_i)$  and therefore is an  $a$ -duct of  $s_i$ . Hence, for each  $a$ -duct  $(B'_p, X'_q)$  of  $s'_i$  we get an  $a$ -duct  $(B_p, X_q)$  of  $s_i$ . As a consequence, for each  $a$ -cable of  $e'$  we get an  $a$ -cable of expression  $e$ .

So, for each action  $\langle p, a, q \rangle$  we get an  $a$ -duct  $(B_p, X_q)$ , and for each  $a$ -global we get an  $a$ -cable obtained by taking product of  $a$ -ducts corresponding to its component local  $a$ -moves. Therefore, for any word  $u'$  in  $Lang(e')$  we get a unique word  $u$  in  $Lang(e)$  and for any word  $v$  in  $Lang(e)$  we get a unique word  $v'$  in  $Lang(e')$ . Now using equivalence  $Lang(H) = Lang(e')$  along with the fact that, the renaming used in the first step of getting product system  $H$  from  $A$ , and the renaming used in the last step of getting expression  $e$  from  $e'$  is well defined, we get language equivalence of expression  $e$  and product system  $A$ .  $\square$

*Proof (Theorem 3).* Let  $A = (A_1, \dots, A_K)$  be a product system with globals defined over  $\Sigma$ . Product system  $A$  may not have separation of labels property. Consider a component  $A_i$  of such product system  $A$ . It may have more than one place having outgoing local  $a$ -moves, for a global action  $a$ . Using globals we rename local  $a$ -moves appropriately across all such component  $A_i$ 's, we get a new product system  $H$  having separation of labels property over a new distributed alphabet. This consists of the first part of the proof. Then we apply Lemma 4 to get an expression for the language of  $H$ , which is renamed back to get an expression for the language of original product system  $A$ .

Since, product system  $A$  satisfies same source property; for each action  $a$ , set  $globals(a)$  can be partitioned into buckets such that any two  $a$ -globals belong to same bucket iff their sets of pre-places are equal.

In each  $A_i$ , where  $i$  in  $loc(a)$ , replace local  $a$ -move  $\langle p, a, q \rangle$  by  $\langle p, a^Z, q \rangle$  if local  $a$ -move  $\langle p, a, q \rangle$  is part of some  $a$ -global belonging to some partition  $Z$  of  $cables(a)$ . Each local  $a$ -move can be part of many  $a$ -globals belonging to same such bucket; but can not be a component of two  $a$ -globals belonging to two different buckets, because of same source property of  $A$ . Hence, this renaming is well defined. Repeat this for each global action  $a$ , to get a product system  $A' = (A'_1, \dots, A'_k)$  over new distributed alphabet  $\Sigma' = (\Sigma'_1, \dots, \Sigma'_k)$ . For each  $a$ -global we have corresponding  $a^Z$ -global. And, we have set  $globals(a^Z)$  for any action  $a^Z$ . Renaming preserves separation of labels property of product  $A$  so product  $A'$  has it and, it continues to have same source property. Since renaming is well defined, for any word  $w'$  in  $Lang(A')$  we get a unique word in  $Lang(A)$  and vice versa.

So we can apply Lemma 4 to get a connected expression  $e' = (s'_1, \dots, s'_k)$ , defined over  $\Sigma'$ , having unique sites property. For an  $a^{Z_i}$ -move of  $A'_i$  we have an  $a^{Z_i}$ -duct corresponding to it and for each  $a^{Z_i}$ -global of  $A'$  we have an  $a^{Z_i}$ -cable in  $e'$ . For each action  $a^{Z_i}$  we have set  $cables(a^{Z_i})$ . Expression  $e'$  has equal source property, since  $A'$  had same source property.

Now renaming back each action  $a^Z$  in  $s'_i$  by action  $a$ , we get regular expression  $s_i$ , for  $Lang(A_i)$ , where  $i$  is in  $loc(a)$ . So we get connected expression  $e = (s_1, \dots, s_k)$ , for language of  $A$ . Renaming gives us an  $a$ -duct of  $s_i$  from an  $a^Z$ -duct of  $s'_i$  and an  $a$ -cable of  $e$  from an  $a^Z$ -cable of  $e'$ . Expression  $e$  continues to have equal source property.  $\square$

*Proof (Lemma 6).* Let  $e = fsync(s_1, s_2, \dots, s_k)$  be a connected expression with cables and having unique sites property.

(Stage 1 of proof):

For all  $a$ -cables set of pre-blocks are same. Consider a global action  $a$  and a location  $i$  in  $loc(a)$ . All  $a$ -ducts of regular expression  $s_i$  have same pre-block but their post-effects are pairwise disjoint. Let  $(B, X)$  be an  $a$ -duct of expression  $s_i$ . Consider actions in  $Init(X)$ . Either all of them are global or local; we do not have a mix of global or local actions because of equal source property of expression  $e$ . If some local action  $c$  is present in  $Init(X)$  means we have a derivative  $d$  of  $s_i$  in  $X$  with  $c$  in  $Init(d)$ . If there is an other post-effect  $Y$  of block  $B$  having local action  $c$  in  $Init(Y)$ , then we have a derivative  $d'$  of  $s_i$  in  $Y$  with action  $c$  in  $Init(d')$ . Since  $X$  and  $Y$  are disjoint, derivatives  $d$  and  $d'$  are distinct. Therefore two distinct occurrences of  $c$  in  $s_i$  correspond to two distinct post-effects of pre-block  $B$ . Therefore, each occurrence of local action  $c$  in  $s_i$  corresponds to only one set of  $a$ -effects of pre-block  $B$ . Now we rename each occurrence of local action  $c$  in expression  $s_i$ , by new local action  $c^X$  if it corresponds to  $a$ -effects  $X$ . Since all  $a$ -effects are pairwise disjoint, this renaming is well-defined. Therefore we get an expression  $s'_i$  over new alphabet  $\Gamma_i$ , having unique sites property. Now we claim that for a derivative  $d$  of  $s_i$ , renaming  $d'$  obtained by renaming  $d$  is also a derivative of  $s'_i$  and in the reverse direction for any derivative  $d'$  in  $Der(s'_i)$ , expression  $d$  obtained by renaming back, is a derivative of  $s_i$ . Proof of this claim is similar to the proof given in Lemma 4. Therefore, for any word  $w'$  in  $Lang(s'_i)$  we get a unique word  $w$  in  $Lang(s_i)$ . We can extend this renaming to sets of derivatives. Repeating this for all component regular expressions, we get a connected expression  $e' = (s'_1, \dots, s'_k)$  over distributed alphabet  $\Gamma = (\Gamma_1, \dots, \Gamma_k)$ . Notice that global actions are not renamed, so they remain unchanged in  $\Gamma$ ; but only local actions are changed from  $\Sigma$ . Hence, for an  $a$ -duct  $(B, X)$  of  $s_i$  we get an  $a$ -duct  $(B', X')$  of  $s'_i$ . Because renaming is well defined and extends to derivatives of expressions, we have that for any word  $w$  in  $Lang(e)$  we have a unique word  $w'$  in  $Lang(e')$  and renaming back we get a unique word in  $Lang(e)$  for any word in  $Lang(e')$ . Since each component of  $e'$  have unique sites, this expression has unique sites, and continues to have equal source property. Therefore, it has equal choice property.

(Stage 2 of proof):

We apply Lemma 5 on expression  $e'$  to get product system  $H = (H_1, \dots, H_k)$  for  $Lang(e')$ , having separation of labels. It has same source property because  $e'$  had equal source property. To explain how we associate an  $a$ -move in  $H_i$  to an  $a$ -duct of  $s'_i$ , for a global action  $a$ , consider an  $a$ -duct  $(B', X')$  of  $s'_i$ . We know that either all actions in  $Init(X')$  are global or all of them are local. First assume that it has only global actions in it. Let  $b$  be a global action in  $Init(X')$ . By application of Lemma 5, we get a unique place having outgoing local  $a$ -moves, say place  $p$ ; and a unique place for having outgoing local  $b$ -moves, say place  $q$ . So for the  $a$ -duct  $(B', X')$  we produce an  $a$ -move  $\langle p, a, q \rangle$  in  $H_i$ . Now assume that all actions of  $Init(X')$  are local. Let local action  $c^X$  belongs to  $Init(X')$ . We know that because of renaming there is only one  $c^X$ -site in  $Der(s'_i)$ . And,

expression  $s'_i$  has unique sites property not only for global actions but also for local actions in  $\Gamma'_i$ . Hence application of Lemma 5 gives us a unique place  $r$  in  $H_i$  which has outgoing  $c^X$ -moves. Therefore, we produce  $\langle p, c^X, r \rangle$  in  $H_i$ , for the  $c^X$ -duct  $(B', X')$  of  $s'_i$ . For two distinct  $a$ -ducts of  $s'_i$ , it is possible that in sequential system  $H_i$  we may have only one local  $a$ -move. So it is possible that for two distinct  $a$ -cables we might get only one  $a$ -global in system  $H$ .

*(Stage 3 of proof):*

Now we rename back, each local move  $\langle p, c^X, r \rangle$  in  $H_i$ , by  $\langle p, c, r \rangle$  to get  $A_i$  over  $\Sigma_i$ . Each sequential system  $A_i$  continues to have separation of labels property. Therefore, we have product system  $A = (A_1, \dots, A_k)$  over distributed alphabet  $\Sigma$ , having separation of labels. The relation  $cables(a)$  remains same. Product system  $A$  continues to have same source property. Since renamings used in the first and last stage of the proof are well defined, combined with the fact the  $Lang(e') = Lang(H)$ , we get that  $Lang(A) = Lang(e)$ .  $\square$