Kleene Theorems for Free Choice Nets Labelled with Distributed Alphabets

Ramchandra Phawade

Indian Institute of Technology Dharwad, Dharwad *580011*, India
Email: prb@iitdh.ac.in

Abstract. We provided [15] expressions for free choice nets having "distributed choice property" which makes the nets "direct product" representable. In a recent work [14], we gave equivalent syntax for a larger class of free choice nets obtained by dropping "distributed choice property".
In both these works, the classes of free choice nets were restricted by a "product condition" on the set of final markings. In this paper we do away with this restriction and give expressions for the resultant classes of nets which correspond to free choice "synchronous products" and "Zielonka automata". For free choice nets with distributed choice property, we give an alternative characterization which uses properties checkable in polynomial time.
Free choice nets we consider are 1-bounded, S-coverable, and are labelled with distributed alphabets, where S-components of the associated S-cover respect the given alphabet distribution.

1 Introduction

There are several different notions of acceptance to define languages for labelled Petri nets, depending on restrictions on labelling and "final" markings [12]. The language of a net with an initial marking and a finite set of final markings, is called L-type language [7]. One goal of this work is to give syntax of expressions for L-type languages for various subclasses of 1-bounded, free choice nets labelled with distributed alphabets. One advantage of using distributed alphabet is that we can see free choice nets as products of automata [11], enabling us to write expressions for the nets using components. This also enables us to compare expressiveness of nets and products of automata. We construct expressions for L-type languages of free choice nets via "free choice Zielonka automata".

Consider the net N of Figure 1 with $G = \{\{r_1, s_1\}, \{r_2, s_2\}\}$ as its set of final markings, with its decomposition into finite state machines in Figure 2. Because this net is decomposable, its markings can be written in tuple form, where each s-component has a place in the tuple: for example $G = \{(r_1, s_1), (r_2, s_2)\}$. For the final marking $\{(r_1, s_1)\}$, its language is expressed as $\mathit{fsync}((ab+ac)^*, (ad+ae)^*)$. Similarly, for the final markings $\{(r_2, s_2)\}$ equivalent expression is $\mathit{fsync}((ab + ac)^*a, (ad + ae)^*a)$. In general, if the places involved in the final markings form a "product ", then its language is specified by taking product of component expressions, using "free choice Zielonka automata with product-acceptance" [15]

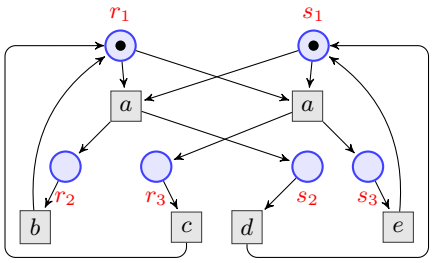

Fig. 1. Free Choice Net without distributed choice

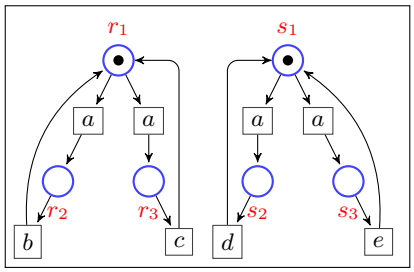

Fig. 2. S-cover of the net in Fig. 1

as intermediary. Even though r_1 and s_2 participate in final markings, marking $\{r_1, s_2\}$ does not belong to G, hence set G do not form a product. The language L of net system (N, G) can be described by, $fsync((ab+ac)^*, (ad+ae)^*)+fsync((ab+ac)^*a, (ad+ae)^*a)$. The key idea is ability to express the language of net as union of language of nets having product condition on final set of markings. This closure under union may not be always possible for restricted classes of languges defined over a distributed alphabet. For example, union of "direct product" languages $L_1 = \{ca, cb\}$ and $L_2 = \{caa, cbb\}$ defined over $\Sigma_1 = \{c, a\}$ and $\Sigma_2 = \{c, b\}$ respectively, is not expressible as a "direct product language". But this language is accepted by "synchronous products": the direct products extended with subset-acceptance.

For the restricted class of direct product representable free choice nets, with its set of final markings having product condition-we gave expressions via product systems with matchings and product-acceptance [15, 16]. As a second goal, we develop syntax for free choice nets with distributed choice, now extended with subset-acceptance. For a net in this class also, its language can be expressed as union of languages accepted by product system with matchings and product-acceptance. This union is accepted by product systems with matching and extended with subset-acceptance ("free choice synchronous products"). As a third contribution, we develop an alternate characterization of this class of nets, via "free choice Zielonka automata with product-moves".

Language equivalent expressions for 1-bounded nets have been given by Grabowski [5], Garg and Ragunath [4] and other authors [8], where renaming operator has been used in the syntax to disambiguate synchronizations. We have chosen to not use this operator and to also exploit the S-decompositions of nets. The syntax for smaller sublclasses of nets such "marked graphs" and "free choice nets with initial markings as feedback vertex set" has been given earlier [9, 13].

Rest of the paper is organized as follows. In the next section, we begin with preliminaries on distributed alphabets and nets. In Section 3 we define product systems with globals and subset-acceptance, and show that their languages can be expressed as union of languages accepted by product systems with globals and product-acceptance. The following section relates these product systems to nets. In Section 5 we develop syntax of expressions for product systems with subset

acceptance, and next section establishes the correspondence between various classes of product systems and expressions. In last section we conclude, with a overview of established correspondences between all three formalisms.

2 Preliminaries

Let Σ be a finite alphabet and Σ^* be the set of all words over alphabet Σ, including the empty word ε. A language over an alphabet Σ 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:

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

Given languages $L_1, L_2, \ldots, L_m$, their synchronized shuffle $L = L_1 \| \ldots \| L_m$ is defined as: $w \in L$ iff for all $i \in \{1, \ldots, m\}, w{\downarrow_{\Sigma_i}} \in L_i$.

Definition 1 (Distributed Alphabet). *Let Loc denote the set $\{1, 2, \ldots, k\}$. A distribution of Σ 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 global action is global in the locations in which it occurs. For a set S let $\wp(S)$ denote the set of all its susbets. For singleton sets like $\{p\}$, sometimes we may write it as p.

2.1 Nets

Definition 2. *A labelled net N is a tuple (S, T, F, λ), 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.*

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 Σ let $T_a = \{t \mid t \in T \text{ and } \lambda(t) = a\}$.

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 produce the new marking $M' = (M \setminus {}^\bullet t) \cup t^\bullet$ and, we write this as $M \xrightarrow{t} M'$ or $M \xrightarrow{\lambda(t)} M'$.

A firing sequence $\lambda(t_1)\lambda(t_2)\dots$ is defined by composition of labels of transition sequence $t_1t_2\dots$ fired from M_0 i.e., $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \dots$. In such a firing sequence, we say that M_j is reachable from M_i, for every $i \leq j$. 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 $Lang(N, M_0, \mathcal{G}) = \{\lambda(\sigma) \in \Sigma^* \mid \sigma \in T^* \text{ and } M_0 \xrightarrow{\sigma} M, \text{ for some } M \in \mathcal{G}\}$.*

Net Systems and its components Here we define components of net. Our notion of component does not require strong connectedness and so it is different from notion of S-component in [3], and therefore our notion of S-cover also differs from theirs.

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 [3]),*
- *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.

Fix a distribution $(\Sigma_1, \Sigma_2, \dots, \Sigma_k)$ of Σ. We define s-decomposition [6] of a net into sequential components.

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 $S = S_1 \cup S_2 \cup \dots S_k$ 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, λ) is 1-bounded and S-decomposable then a marking can be written as a k-tuple from its component places $S_1 \times S_2 \times \dots \times S_k$. We give below the "product" condition on the set of final markings of a net system which is known [17, 11] to restrict classes of languages.

Definition 7. *An S-decomposable labelled net system with product-acceptance $(N, M_0, \mathcal{G})$ is an S-decomposable labelled net $N = (S, T, F, \lambda)$ with an initial marking M_0 and a set of markings $\mathcal{G} \subseteq \wp(S)$, which is a 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 $loc(a)$.

2.2 Free choice nets and their properties

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 $[x]$.

The set of all a-labelled transitions along with places r_1 and s_1 form a cluster of the net shown in Figure 4.

Definition 8 (Free choice nets [3]). *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 $\pi(t) = \Pi_{i \in loc(a)}(t^\bullet \cap S_i)$ 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 from [16, 15] provides the way to direct product representability.

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

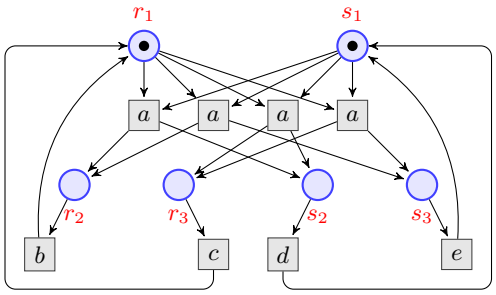

Fig. 3. Labelled Free Choice Net system with distributed choice

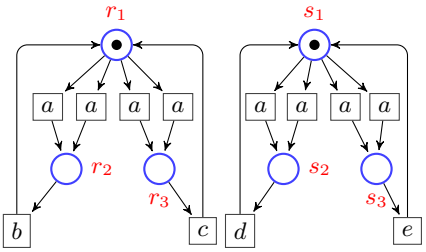

Fig. 4. S-cover of the net in Fig. 3

Example 1 (Free choice net system without distributed choice and having product-acceptance). Consider a distributed alphabet $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$ and the labelled net system N shown in Figure 1 and labelled over Σ. Its (only

possible) S-cover having two S-components with sets of places $S_1 = \{r_1, r_2, r_3\}$ and $S_2 = \{s_1, s_2, s_3\}$ respectively, is given in Figure 2. For the cluster C of r_1, we have the set of a-labelled transitions $C_a = \{t_1, t_2\}$ with its post-projections $C_a[1] = \{r_2, r_3\}$ and $C_a[2] = \{s_2, s_3\}$. So we get
$postdecomp(C_a) = \{(r_2, s_2), (r_2, s_3), (r_3, s_2), (r_3, s_3)\}$.

The post-products are $\pi(t_1) = \{(r_2, s_2)\}$ and $\pi(t_2) = \{(r_3, s_3)\}$ so $post(C_a) = \{(r_2, s_2), (r_3, s_3)\}$. Since $postdecomp(C_a) \nsubseteq post(C_a)$, this cluster does not have distributed choice, so the net system does not have it.

With the set of final markings $\{(r_1, s_1), (r_1, s_2), (r_2, s_1)(r_2, s_2)\}$ satisfying product condition, the language L_p accepted by this net system is $r^*[\varepsilon + a + ab + ad]$ where $r = (a(bd + db) + a(ce + ec))$.

Example 2 (Free choice net system without distributed choice and not satisfying product condition of the set of final markings). Consider the net system of Example 1 whose underlying net is shown in Figure 1. With set of final markings $\{(r_1, s_1), (r_2, s_2)\}$, which do not satisfy product condition, the language L_s accepted by this net system is $r^*[\varepsilon + a]$ where $r = (a(bd + db) + a(ce + ec))$.

Example 3 (A net with distributed choice property and product acceptance condition). Consider the labelled net system $(N, (r_1, s_1), \mathcal{G}))$ of Figure 3, defined over distributed alphabet $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$, and where $\mathcal{G} = \{(r_1, s_1), (r_1, s_2), (r_2, s_1), (r_2, s_2)\}$ is the set of final markings satisfying product condition. Its two S-components with sets of places $S_1 = \{r_1, r_2, r_3\}$ and $S_2 = \{s_1, s_2, s_3\}$, are shown in Figure 4. For cluster C of r_1, we have $C_a = \{t_1, t_2, t_3, t_4\}$, $C_a[1] = \{r_2, r_3\}$ and $C_a[2] = \{s_2, s_3\}$, hence $postdecomp(C_a) = \{(r_2, s_2)(r_2, s_3)(r_3, s_2)(r_3, s_3)\}$. The post-products for transitions are $\pi(t_1) = \{(r_1, s_2)\}$, $\pi(t_2) = \{(r_2, s_3)\}$, $\pi(t_3) = \{(r_3, s_2)\}$, $\pi(t_4) = \{(r_3, s_3)\}$, giving us $post(C_a) = \{(r_2, s_2)(r_2, s_3)(r_3, s_2)(r_3, s_3)\}$. Therefore we have $postdecomp(C_a) = post(C_a)$. 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. Language L_3 accepted by the net system is $r^*[\varepsilon + a + a(b + c) + a(d + e)]$ where $r = (a(bd + db) + a(be + eb) + a(cd + dc) + a(ce + ec))$.

Example 4 (A net with distributed choice and subset-acceptance). Consider the net system of Example 3, with the underlying net shown in Figure 3 with set of final markings $\{(r_1, s_1), (r_2, s_2)\}$. The language L_4 accepted by this net system is $r^*[\varepsilon + a]$ where $r = (a(bd + db) + a(be + eb) + a(cd + dc) + a(ce + ec))$.

3 Product systems

We define product systems over a fixed distribution $(\Sigma_1, \Sigma_2, \ldots, \Sigma_k)$ of Σ. First we define sequential systems.

Definition 10. *A sequential system over a set of actions Σ_i is a finite state automaton $A_i = \langle P_i, \rightarrow_i, G_i, p_i^0 \rangle$ where P_i are called **states**, $G_i \subseteq P_i$ are final states, $p_i^0 \in P_i$ is the initial state, and $\rightarrow_i \subseteq P_i \times \Sigma_i \times P_i$ is a set of **local moves**.*

For a local move $t = \langle p, a, p' \rangle$ of $\rightarrow_i$ state p is called pre-state sometimes denoted by $pre(t)$ and p' is called post-state of t, sometimes denoted by $post(t)$. This notation is extended to a set of local moves as well.

Let $\rightarrow_a^i$ denote the set of all a-labelled moves in the sequential system A_i. The language of a sequential system is defined as usual.

Definition 11. *Let $A_i = \langle P_i, \rightarrow_i, G_i, p_i^0 \rangle$ be a sequential system over alphabet Σ_i for $1 \leq i \leq k$. A **product system** A over the distribution $\Sigma = (\Sigma_1, \ldots, \Sigma_k)$ is a tuple $\langle A_1, \ldots, A_k \rangle$.*

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 initial product state of A is $R^0 = (p_1^0, \ldots, p_k^0)$. The final states of A are denoted by G.

3.1 Direct products

Let $\Rightarrow_a = \Pi_{i \in loc(a)} \rightarrow_a^i$ for the product system A. The set of all global moves of A is $\Rightarrow = \bigcup_{a \in \Sigma} \Rightarrow_a$. This move is *global* within the set of component sequential machines where label a occurs. Then for a global move g in $\Rightarrow_a$, we define its set of pre-states pre(g) as the set of pre-states of all its component a-moves. The set of post-states post(g) as the set of post-states 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 $loc(a)$.

When final states of A are $G = \Pi_{i \in Loc} G_i$ then we say A is product system with product-acceptance. These systems are called direct products in [11]. And, if $G \subseteq \Pi_{i \in Loc} G_i$ then A is a product system with subset-acceptance, these systems are called synchronous products in [11].

The runs of a product system A over some word w are described 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 $j \notin loc(a), R[j] = Q[j]$. 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 $Lang(A)$ of product system A, as the set of words on which the product system has an accepting run. The set of languages accepted by direct (resp. synchronous) products is called direct (resp. synchronous) product languages.

We use a characterization from [11] of languages accepted by direct products.

Proposition 1. *L is a direct product language defined over distributed alphabet Σ iff $L = \{w \in \Sigma^* \mid \forall i \in \{1, \ldots, k\},\ \exists u_i \in L$ such that $w{\downarrow}_{\Sigma_i} = u_i{\downarrow}_{\Sigma_i}\}$.*

If $L = Lang(A)$ for direct product $A = \langle A_1, \ldots, A_k \rangle$ defined over distributed alphabet Σ then $L = Lang(A_1) \| \ldots \| Lang(A_k)$.

We also use a characterization of synchronous product languages [11].

Proposition 2. *A language over distributed alphabet Σ is accepted by a product system with subset-acceptance if and only if it can be expressed as a finite union of direct product languages.*

Definition 12 (*PS-matchings* [15]). *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-states of a-moves in $\rightarrow_i^a$, and if a state $p \in P_i$ 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). Such a system is denoted by PS-matchings.

We have *PS-matchings* with product-acceptance, if the set of final product states of it is a product of final states of component machines, or ***PS-matchings* with subset-acceptance**, if the set of final product states is a subset of product of final states of individual components.

A run of *PS-matchings* A is said to be consistent with a matching of labels [15] if for all global actions a and every prefix of the run $R^0 \stackrel{v}{\Rightarrow} R \stackrel{a}{\Rightarrow} Q$, the pre-states $R{\downarrow}loc(a)$ are in the matching.

Consistency of matchings is a behavioural property and to check if a *PS-matchings* A has it and can be done in PSPACE [15, 16].

Following property from [15] is used to capture free choice property.

Definition 13 (conflict-equivalent matchings for *PS-matchings*). *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. 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.*

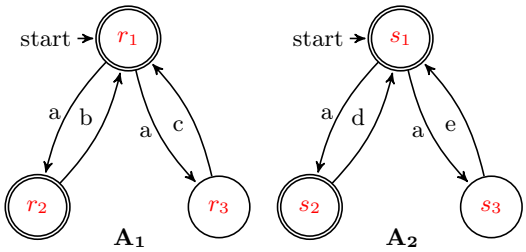

Figure 5, shows a product system defined over a distributed alphabet $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$. It has two components A_1,and A_2 with final states $G_1 = \{r_1, r_2\}$ and, $G_2 = \{s_1, s_2\}$, respectively.

Fig. 5. Product system (A_1, A_2)

Example 5 (Product system with matchings). Consider product system of Figure 5 and relation $matching(a) = \{(r_1, s_1)\}$ relation. This matching is conflict-equivalent and the system is consistent with this matching relation.

We have a *PS-matchings* $\mathcal{A} = (A_1, A_2)$ with product acceptance condition, if its set of final states is $G_1 \times G_2$. With the set of final states as $\{(r_1, s_1), (r_2, s_2)\} \subseteq G_1 \times G_2$, we have a *PS-matchings* $\mathcal{B} = (A_1, A_2)$ having subset-acceptance.

Lemma 1 presents a language not accepted by any direct product.

Lemma 1. *The language* $L_4 = \{(abd + adb + abe + aeb + ace + aec + acd + adc)^*(\varepsilon + a)\}$ *from Example 4 is not accepted by any direct product.*

We know that the class of synchronous product languages is strictly larger than the class of direct product languages [11]. With the matching relations this relationship is preserved. The *PS-matchings* $\mathcal{B}$ of Example 5 accepts language L_4 which by Lemma 1, is not accepted by any direct product. Hence, the class of languages accepted by *PS-matchings* with subset acceptance condition, is strictly larger, than the class of languages accepted by *PS-matchings* with product-acceptance.

However, using Proposition 2 we have the following characterization of *PS-matchings* with subset-acceptance.

Corollary 1. *A language L is accepted by a product system with subset-acceptance and, having conflict-equivalent and consistent matchings if and only if L can be expressed as a finite union of languages accepted by product system with product-acceptance and, having conflict-equivalent and consistent matchings.*

Lemma 2 presents a language not accepted by any synchronous product.

Lemma 2. *The language* $L_s = \{abd, adb, ace, aec\}^*(\varepsilon + a)$ *of Example 2 is not a synchronous product language.*

So we have language L_s which is not accepted by any *PS-matchings* with subset-acceptance. This motivates the bigger class of automata over distributed alphabets, which we discuss next.

3.2 Product systems with globals

Let $A = \langle A_1, \ldots, A_k \rangle$, be a product system over the distribution $\Sigma = (\Sigma_1, \ldots, \Sigma_k)$ and, let *globals*(a) be a subset of its global moves $\Rightarrow_a$, and a-global denote an element of *globals*(a).

Definition 14. *A product system with globals (PS-globals) is a product system with relations globals*(a)*, for each global action a in* Σ*.*

With subset-acceptance condition these systems are called Asynchronous (or Zielonka) automaton [17, 11]. Runs of a product system with globals, are defined in the same way as for the direct products, with an additional requirement of $\Pi_{j \in loc(a)}(\langle R[j], a, Q[j] \rangle \in globals(a)$, to be satisfied when $R \xrightarrow{a} \mathrm{Q}$ is to be taken. With abuse of notation sometimes we use $pre(a)$ to denote the set $\{R \mid \exists Q, R \xrightarrow{a} Q\}$. Following property [14] is used to capture free choiceness, in product systems with globals.

Definition 15 (same source property). *A product system with globals have same source property if, any two global moves share a pre-state then their sets of pre-states are same.*

Example 6 (Product system with globals). Consider the product system of Figure 5. Let $globals(a) = \{ ((r_1 \xrightarrow{a} r_2), (s_1 \xrightarrow{a} s_2)), ((r_1 \xrightarrow{a} r_3), (s_1 \xrightarrow{a} s_3))\}$. This system has same source property.

With the given *globals*(a) we have a *PS-globals* $\mathcal{C} = (A_1, A_2)$ with product acceptance condition, if its set of final states is $G_1 \times G_2$. And, for the set of final states $\{(r_1, s_1), (r_2, s_2)\} \subseteq G_1 \times G_2$, we have a *PS-globals* $\mathcal{D} = (A_1, A_2)$ with subset-acceptance.

The language $L_s = \{abd, adb, ace, aec\}^*(\varepsilon + a)\}$ of Lemma 2, is accepted by product system with globals $\mathcal{D}$ of Example 6 with same source property.

Product systems with globals and product-acceptance are not considered in [11]. We give in Lemma 3, a characterization of class of languages accepted by product systems with globals and having subset-acceptance, in terms of *PS-globals* and product-acceptance.

Lemma 3. *A language is accepted by a PS-globals with subset-acceptance if and only if it can be expressed as a finite union of languages accepted by PS-globals with product-acceptance.*

In the construction, transition structure of local components is preserved, and so the global moves, so we have Corollary 2, which is used to get syntax for product systems with subset acceptance condition.

Corollary 2. *A language L is accepted by a PS-globals with subset-acceptance and having same source property if and only if L can be expressed as a finite union of languages accepted by PS-globals with product-acceptance and same source property.*

In a product system with globals and having same source property, global moves for an action a can be partitioned into different compartments : two global a-moves belong to same compartment if they have the same set of pre-states. For any a-global g of a same source compartment $\Rightarrow_a^{SS}$, we associate a target-configuration $\pi(g) = \Pi_{i \in loc(a)} post(g) \cap P_i$. Let $post(\Rightarrow_a^{SS}) = \{\pi(g) \mid g \in \Rightarrow_a^{SS}\}$. We define post-projection of a compartment as $\Rightarrow_a^{SS}[i] = post(\Rightarrow_a^{SS}) \cap P_i$ and, post-decomposition as $postdecomp(\Rightarrow_a^{SS}) = \Pi_{i \in loc(a)} \Rightarrow_a^{SS}[i]$.

Definition 16. *A product system with globals and having same source property, is said to have* ***product moves property****, if for all a in* Σ*, and for all same source compartments* $\Rightarrow_a^{SS}$ *of a-globals,* $postdecomp(\Rightarrow_a^{SS}) \subseteq post(\Rightarrow_a^{SS})$.

Product systems $\mathcal{C}$ and $\mathcal{D}$ of Example 6 do not have product moves property. Product system of Example 7 has product moves property.

Example 7 (Product system with globals and product moves property). Consider product system $\mathcal{A}$ of Example 5 where the set of final states is $G_1 \times G_2$. With $globals(a) = \{ ((r_1 \xrightarrow{a} r_2), (s_1 \xrightarrow{a} s_2)), ((r_1 \xrightarrow{a} r_2), (s_1 \xrightarrow{a} s_3)), ((r_1 \xrightarrow{a} r_3), (s_1 \xrightarrow{a} s_2)) ((r_1 \xrightarrow{a} r_3), (s_1 \xrightarrow{a} s_3)) \}$ relation, we have *PS-globals* $\mathcal{A}'$ which has product moves property. This also has same source property.

Now consider the product system $\mathcal{B}$ of Example 5, with $\{(r_1, s_1), (r_2, s_2)\} \subseteq G_1 \times G_2$ as its final states, and having the *globals*(a) relation as above, we get a *PS-globals* $\mathcal{B}'$ with subset acceptance condition, having product moves and same source property.

A product system with globals is said to be live, if for any global move g and any reachable product state R, there exists a product state Q such that g is enabled at Q.

3.3 Relating product systems with matchings and globals

First we show in Theorem 1 how to construct a product system with consistent and conflict-equivalent matchings from a *PS-globals* with same source property.

Theorem 1. *Let Σ be a distributed alphabet and A be a product system with globals defined over it.Then we can construct a product system B with matchings, linear in the size of product system A with globals such that,*

1. *if A has same source property then B has conflict-equivalent matchings,*
2. *in addition, if A is live then*
   (a) B is consistent with matchings, and
   (b) Lang(A) = Lang(B).

Now, from a *PS-matchings* with consistent and conflict-equivalent matchings we construct a product system with globals having same source property.

Theorem 2. *Let Σ be a distributed alphabet and let B be a product system with conflict equivalent and consistent matchings. Then for the language of B we can construct a product system A with globals over Σ having same source and product moves property. The constructed product system A with globals is exponential in the size of system B having matching of labels.*

4 Nets and Product systems

We first present a generic construction of a 1-bounded S-coverable labelled net systems, from product systems with globals.

Definition 17 (*PS-globals* to nets). *Given a PS-globals $A = \langle A_1, \ldots, A_k \rangle$ over distribution Σ, a net system $(N = (S, T, F, \lambda), M_0, \mathcal{G})$ is constructed as follows: The set of places is $S = \bigcup_i P_i$, The set of transitions $T = \bigcup_a$ globals(a), for all actions a in Σ. Define $T_i = \{\lambda^{-1}(a) \mid a \in \Sigma_i\}$. The labelling function λ labels by action a the transitions in globals(a). The flow relation is $F = \{(p, g), (g, q) \mid g \in T_a, g[i] = \langle p, a, q \rangle, i \in loc(a)\}$, define F_i as its restriction to the transitions T_i for $i \in loc(a)$. Let $M_0 = \{p_1^0, \ldots, p_k^0\}$, be the initial product state and $\mathcal{G} = G$ as the set of final global states.*

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

Lemma 4. *The constructed net system N from a PS-globals A, as in Definition 17, is S-coverable and $Lang(N, M_0, \mathcal{G}) = Lang(A)$. The size of constructed net is linear in the size of product system.*

Applying 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 3. *Let $(N, M_0, \mathcal{G})$ be the net system constructed from PS-globals A as in Definition 17.*

- *If A has same source property then N is a free choice net,*
- *In addition if A has product moves property, then N has distributed choice.*

In the construction, if the product system has subset-acceptance then we get a net with a set of final markings, which may not have product condition. Since A has subset-acceptance, we generalize the results obtained in [14].

For the product system $\mathcal{D}$ of Example 6, accepting language L_s we can construct the net system of Example 2.

In the case that the product system A has matchings, transitions of net constructed are reachable global moves of system A [15, 16].

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 [3]. Now we describe a linear-size construction of a product system from a net which is S-coverable.

Definition 18 (nets to *PS-globals*). *Given a 1-bounded labelled and an S-coverable 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 $A = \langle A_1, \ldots, A_k \rangle$, as follows. Take $P_i = S_i$, and p_i^0 the unique state in $M_0 \cap P_i$. Define local moves $\rightarrow_i = \{\langle p, \lambda(t), p' \rangle \mid t \in T_i$ and $(p, t), (t, p') \in F_i$, for $p, p' \in P_i\}$. So we get sequential systems $A_i = \langle P_i, \rightarrow_i, p_i^0 \rangle$, and the product system $A = \langle A_1, A_2, \ldots, A_k \rangle$ over alphabet Σ. Global moves are globals$(a) = \{\Pi_{i \in loc(a)} t[i] \mid t \in T_a\}$. And, the set of final states is $G = \mathcal{G}$.*

Lemma 5. *From net system N with a final set of markings, the construction of the PS-globals A in Definition 18 above preserves language. The product system A is linear in the size of net, and product system has subset-acceptance.*

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, for each global a-move in product system we have an a-labelled transition in the net having same

pre and post-places. 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 a free choice net, we get same source property in the product system obtained. And, for each transition in the net we have a global trasition hence, A has product moves property if the net has distributed choice.

Theorem 4. *Let $(N, M_0, \mathcal{G})$ be a 1-bounded, and an S-coverable labelled net with a set of final markings $\mathcal{G}$. Then*

- *if N has free choice property, then constructed product system A with globals, has same source property,*
- *in addition, if the net has distributed choice, then A has product moves.*

In construction of Definition 18, we start with a net having distributed choice and a final set of markings then we get product system with matching with subset acceptance condition. Note that in this case we do not have to construct globals [15, 16].

For the net system of Example 2 and accepting language L_s we can construct the product system $\mathcal{D}$ of Example 6.

Therefore, we generalize the results from [15, 16].

Theorem 5. *For a 1-bounded, S-coverable labelled net having distributed choice and given with a set of final markings. Then one can construct a product system with conflict-equivalent and consistent matchings and having subset-acceptance.*

Given below is the reverse direction.

Theorem 6. *For a product system with conflict-equivalent, consistent matchings, and subset-acceptance, we get language equivalent free choice net with distributed choice and having a set of final markings.*

5 Expressions

First we define regular expressions and its derivatives.

5.1 Regular expressions and their properties

A regular expression over alphabet Σ_i such that constants 0 and 1 are not in Σ_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 Σ in s.

For each regular expression s over Σ_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. We can syntactically check whether the empty word $\varepsilon \in Lang(s)$.

We use derivatives of regular expressions which are known since the time of Brzozowski [2], Mirkin [10] and Antimirov [1].

Definition 19 (Antimirov derivatives [1]). *Given regular expression s and symbol a, the set of partial derivatives of s with respect to a, written $\mathrm{Der}_a(s)$ are defined as follows.*

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

Inductively $Der_{aw}(s) = Der_w(Der_a(s))$.
The set of all partial derivatives $Der(s) = \bigcup_{w \in \Sigma_i^*} Der_w(s)$, where $Der_\varepsilon(s) = \{s\}$.
We have derivatives $Der_a(ab + ac) = \{b, c\}$ and $Der_a(a(b + c)) = \{b + c\}$.

A derivative d of s with action $a \in Init(d)$ 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 [15] we put together derivatives which may correspond to the same state in a finite automaton.

Definition 20 ([15]). *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 $L_a^D = \{xay \mid xay \in L, \exists d \in Der_x(s) \cap D, \exists d' \in Der_{ay}(d) \text{ with } \varepsilon \in Lang(d')\}$, and the prefixes $Pref_a^D(L) = \{x \mid xay \in L_a^D\}$, and the suffixes $Suf_a^D(L) = \{y \mid xay \in L_a^D\}$. We say that the derivatives in set D a-bifurcate L if $L_a^D = Pref_a^D(L)\ a\ Suf_a^D(L)$.*

We use partitions [15] of the a-sites of s into blocks such that each block (that is, element of the partition) a-bifurcates L. For an action a, let $Part_a(s)$ denote such a partition. In addition to thinking of blocks of the partition as places of an automaton, we can think of blocks and their effects as local moves.

Definition 21 ([14]). *Given an action a, a set of a-sites B of regular expression s and a specified set of a-effects $E \subseteq Der_a(B)$, we define the relativized languages*

$$L_a^{(B,E)} = \{xay \in L \mid \exists d \in Der_x(s) \cap B, \exists d' \in Der_a(d) \cap E, \text{ and } \exists d'' \in Der_y(d') \text{ with } \varepsilon \in Lang(d'')\}.$$

We define the prefixes $Pref_a^{(B,E)}(L) = \{x \mid xay \in L_a^{(B,E)}\}$ *and the suffixes* $Suf_a^{(B,E)}(L) = \{y \mid xay \in L_a^{(B,E)}\}$. *We say that a tuple* (B, E) *a-*funnels *L if* $L_a^{(B,E)} = Pref_a^B(L) \cdot a \cdot Suf_a^{(B,E)}(L)$. *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 $Init(B, E)$ as $Init(B, E) = Init(B)$, call B as its pre-block and call E as its post-effect. For all i in $loc(a)$ 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 $E = E'$. Given an a-duct $d = (B, E)$ its post-effect E is sometimes denoted by $d^\bullet$ and its pre-block B can be denoted as ${}^\bullet d$. For a collection of ducts z, the set of all their post-effects (resp. pre-blocks) is denoted as $z^\bullet$ (resp. ${}^\bullet z$). In a similar way, we define the set of post-effects of an a-cable D, as $D^\bullet = \{D[i]^\bullet \mid i \in loc(a)\}$ and its set of pre-blocks as ${}^\bullet D = \{{}^\bullet D[i] \mid i \in loc(a)\}$.

5.2 Connected expressions over a distributed alphabets

The syntax of connected expressions defined over a distribution $(\Sigma_1, \Sigma_2, \ldots, \Sigma_k)$ of alphabet Σ is given below.

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

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

A connected expression $e = fsync(s_1, s_2, \ldots, s_k)$ over Σ, is said to have equal choice if, for all global actions a in Σ 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.

For a connected expression defined over distributed alphabet its derivatives and semantics were given in [15], and are given as follows. For the connected expression 0, we have $Lang(0) = \emptyset$. For the connected expression $e = fsync(s_1, \ldots, s_k)$, its language is $Lang(e) = Lang(s_1) \| Lang(s_2) \| \ldots \| Lang(s_k)$.

The definitions of derivatives extended to connected expressions [15] is as follows. The expression 0 has no derivatives on any action. Given an expression $e = fsync(s_1, s_2, \ldots, s_k)$, its derivatives are defined by induction using the derivatives of the s_i on action a:

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

5.3 Connected expressions with pairings

We recall some properties of connected expressions over a distribution. which were [15], useful in construction of free choice nets.

Definition 22 ([15]). *Let* $e = fsync(s_1, s_2, \ldots, s_k)$ *be a connected expression over* Σ. *For a global action* a, pairing(a) *is a subset of tuples* $\Pi_{i \in loc(a)} \text{Part}_a(s_i)$ *such that the projection of these tuples includes all the blocks of* $\text{Part}_a(s_i)$, *and*

if a block of $\text{Part}_a(s_j), j \in loc(a)$ *appears in one tuple of the pairing, it does not appear in another tuple. (For convenience we also write* pairing(a) *as a subset of* $\Pi_{i \in loc(a)}\text{Der}(s_i)$ *which respects the partition.) We call* pairing(a) *equal choice if for every tuple in the pairing, the blocks of derivatives in the tuple have equal choice.*

Derivatives for connected expressions with *pairing* are defined as follows. 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 $pairing(a)$. Expression e is said to have equal choice property if it has equal choice pairing of actions for all global actions a in Σ.

Given a connected expression e with pairings, checking if it is consistent with pairing of actions can be done in PSPACE [15, 16].

5.4 Connected expression with cables (*CE-cables*)

We give some properties of connected expressions over a distribution, which extend the notion of pairing, and have been related to product systems with globals [14].

Definition 23 ([14]). *Let* $e = fsync(s_1, s_2, \ldots, s_k)$ *be a connected expression over* Σ. *For each action* a *in* Σ, *we define the set* a-cables(e) $= \Pi_{i \in loc(a)} a\text{-ducts}(s_i)$. *For an action* a, *an* a-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-cable D *if there exists* j *in* $loc(a)$ *and there exists* $Y \subseteq \text{Der}_a(B)$ *such that* $D[j] = (B, Y)$, *i.e. if* B *is a pre-block of a component* a-duct *of* D. *For any* a-cable D, *its set of pre-blocks* ${}^\bullet D = \cup_{i \in loc(a)}\{B_i \mid B_i$ *appears in* $D\}$, *i.e. the set of pre-blocks of all the of its component* a-ducts.
For expression e, *let* cables(a) $\subseteq a\text{-cables}(e)$, *such that for all* i *in* $loc(a)$

1. *Each block* B *in* $\text{Part}_a(s_i)$, *appears in at least one* a-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-ducts *of* s_i *appearing in it have same pre-block then, they must have disjoint post-effects.*

A connected expression with cables (CE-cables) *is a connected expression with relations* cables(a) *of it, for each global action* a *in* Σ.

Derivatives of a connected expression with cables are [14] defined as follows. The *CE-cables* 0 has no derivatives on any action. For expression $e = fsync(s_1, s_2, \ldots, s_k)$, we define its derivatives on action a, by induction, using a-ducts and the derivatives of s_j as:

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

We use the word derivative for expressions such as $d = \mathit{fsync}(r_1, \ldots, r_k)$ given above. The reachable derivatives are $Der(e) = \{d \mid d \in Der_x(e), x \in \Sigma^*\}$. A *CE-cables* 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. Language of e is the set of words over Σ defined using derivatives as below.

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

So we can have next derivative on action a, if it is allowed by the *cables*(a) relation. The number of derivatives may be exponential in k.

Example 8 (CE-pairings and CE-cables). Let $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$ be a distributed alphabet. Let $e = \mathit{fsync}((ab + ac)^*, (ad + ae)^*)$ be a connected expression defined over Σ. Here, $r_1 = (ab + ac)^*$ and $s_1 = (ad + ae)^*$. The set of derivatives of r_1 is $Der(r_1) = \{r_1, r_2 = br_1, r_3 = cr_1\}$ and for s_1 it is $Der(s_1) = \{s_1, s_2 = ds_1, s_3 = es_1\}$. We have a-sites$(r_1) = r_1$ and $Part_a(r_1) = \{D = r_1\}$. Similarly, a-sites$(s_1) = r_1$ and $Part_a(s_1) = \{D' = s_1\}$. The only possible pairing relation is $pairing(a) = \{(D, D')\}$. We have $Der_a(e) = \{\mathit{fsync}(r_i, s_j) \mid i, j \in \{2, 3\}\}$. Expression e satisfies equal choice property.

Now we associate a cabling relation with e. The set of a-effects of D is $Der_a(D) = \{r_2, r_3\}$. The set of a-ducts of r_1 is $\{(D, r_2), (D, r_3), (D, \{r_2, r_3\})\}$. The set of a-effects of D' is $Der_a(D') = \{s_2, s_3\}$ and the set of a-ducts of s_1 is $\{(D', s_2), (D', s_3), (D', \{s_2, s_3\})\}$. A possible *cables*$(a)$ relations for expression e is $\{((D, r_2), (D', s_2)), ((D, r_3), (D', s_3))\}$. See that each block in the $Part_a(r_1)$ and $Part_a(s_1)$ appears at least once in the *cables*(a) relation. And two a-ducts of r_1 appearing in this relation, have same pre-block D, we have that their set of post-effects r_2 and r_3 are disjoint. This condition also holds for a-ducts of s_1.

For both a-cables set of pre-blocks is identical, therefore *cables*(a) satisfies equal source property. We have $Der_a(e) = \{\mathit{fsync}(r_2, s_2), \mathit{fsync}(r_3, s_3)\}$, but expression $\mathit{fsync}(r_2, s_3)$ 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. We have $Der(e) = \{e, (r_2, s_2), (r_3, s_3), (r_1, s_2), (r_1, s_3), (r_2, s_1), (r_3, s_1)\}$.

Another such example of connected expression with pairings (resp. cables) is given below.

Example 9 (CE-pairings). Let $e = \mathit{fsync}((ab+ac)^*a, (ad+ae)^*a)$ be a connected expression defined over Σ. Let $p_1 = (ab+ac)^*a$ with language $L_1$and $q_1 = (ad+ae)^*a$ with language L_2. The set of derivatives are $Der(p_1) = \{p_1, p_2 = bp_1, p_3 = cp_1, p_4 = \varepsilon\}$ and $Der(q_1) = \{q_1, q_2 = dq_1, q_3 = eq_1, q_4 = \varepsilon\}$. The partitions of a-sites is $Part_a(p_1) = \{B = \{p_1\}\}$ and $Part_a(q_1) = \{B' = \{q_1\}\}$. A pairing relation is $pairing(a) = \{(B, B')\}$ and wrt to that $Der_a(e) = \{\mathit{fsync}(p_i, q_j) \mid i, j \in \{2, 3, 4\}\}$. Expression e has equal choice property.

Now we associate a cabling relation with e. The set of a-effects of B is $Der_a(B) = \{p_2, p_3, p_4\}$ and $Der_a(B') = \{q_2, q_3, q_4\}$. The set of a-ducts for p_1 is $\{(B, \{p_2\}), (B, \{p_3\}), (B, \{p_2, p_4\}), (B, \{p_3, p_4\})(B, \{p_2, p_4, p_3\})\}$ and for q_1

is $\{(B', q_2), (B', q_3), (B', \{q_2, q_4\}), (B', \{q_3, q_4\})(B', \{q_2, q_4, q_3\})\}$. A *cables*(a) relation is $\{((B, p_2), (B', q_2)), ((B, p_3), (B', q_3)), ((B, p_2), (B', q_2))\}$. Expression e has equal source property and its set of derivatives with respect to letter a is $Der_a(e) = \{fsync(p_2, q_2), fsync(p_3, q_3), fsync(p_4, q_4)\}$.

Now we give two new properties of connected expressions. In a connected expression with cables and having equal source property, cables for an global action a can be partitioned into different compartments : two a-cables belong to same compartment if they have equal source. For any such a-cable D belonging to an equal source compartment ES_a of a-cables, we can associate a set of its post-blocks listed in some order as $\pi(D) = \Pi_{i \in loc(a)}(D^\bullet \cap \chi_i)$, where $\chi_i = \{Part_a(s_i) \mid a \in \Sigma\}$. Let $post(ES_a) = \{\pi(D) \mid D \in ES_a\}$. We define post-projection of compartment ES_a as $ES_a[i] = post(ES_a^i) \cap \chi_i$, and its post-decomposition as $postdecomp(ES_a) = \Pi_{i \in loc(a)} ES_a[i]$.

Following property of connected expressions is related later to product-moves-property of direct products and hence to distributed-choice of nets.

Definition 24 (product derivatives property). *A connected expression with cables and having equal source property, is said to have* ***product derivatives property****, if for all a in Σ, and for all equal source compartments ES_a of a-cables $postdecomp(ES_a) \subseteq post(ES_a)$.*

Example 10. The connected expression $e = fsync((ab + ac)^*, (ad + ae)^*)$ of Example 8 does not have product derivative property with the given cabling relation $\{((D, r_2), (D', s_2)), ((D, r_3), (D', s_3))\}$. If we associate the cabling relation $\{((D, r_2), (D', s_2)), ((D, r_3), (D', s_3), ((D, r_2), (D', s_3)), ((D, r_3), (D', s_2))\}$ with e then it has product derivative property.

Definition 25. *A connected expression e is* ***action-live*** *if for all actions a in Σ, from any reachable derivative of e, we can reach an a-derivative of e.*

5.5 Relating connected-expressions with pairings and with cables

First, we show how connected expressions with equal choice and consistent pairings can be seen as connected expression with cables and having equal source and product-derivatives property.

Theorem 7. *Let Σ be a distributed alphabet and e be a connected expression having equal choice and consistent pairing of actions, defined over Σ. Then for the language of e, we can construct a connected expression e' with cables having equal source and product derivatives property. The constructed expression e' with cables is exponential in the size of expression e with pairings.*

Now we give a language preserving construction of connected expression with equal choice and consistent pairings from a *CE-cables* having equal source and product-derivatives property.

Theorem 8. *Let Σ be a distributed alphabet and e' be a connected expression with cables defined over it.Then we can construct a connected expression e with pairings linear in the size of e'. And, if e' has equal source then e has equal choice property. In addition, if e' is action-live then if e' has product moves property then e has consistency of pairing. $Lang(e) = Lang(e')$.*

5.6 Sum of Connected Expressions (*SCE*)

We give syntax for sum of connected expressions (SCE) defined over a distribution $(\Sigma_1, \Sigma_2, \ldots, \Sigma_k)$ of alphabet Σ.

$$e ::= 0 | e_1 + \cdots + e_m, \text{ where } e_i \text{ is a connected expression } (CE) \text{ over } \Sigma$$

For an *SCE* e its semantics is given as follows: For the *SCE* 0, we have $Lang(0) = \emptyset$. For the *SCE* $e = e_1 + e_2, \cdots, e_m$, its language is given as $Lang(e) = Lang(e_1) \cup Lang(e_2) \cup \cdots \cup Lang(s_m)$. The definitions of derivatives extended to *SCE*s is as given below. The expression 0 has no derivatives on any action. Derivative of an *SCE* with respect to a letter a is defined as: $Der_a(e) = Der_a(e_1) \cup Der_a(e_2) \cup \cdots \cup Der_a(s_m)$. Inductively $Der_{aw}(e) = Der_w(Der_a(e))$. The set of all derivatives $Der(e)$ is union of sets of derivatives of e with respect to all words w in Σ_i^* for all i in $\{1, \ldots, m\}$.

Definition 26 (*SCE* with pairings)). *An SCE $e = e_1 + \cdots + e_m$ where each e_i is a CE-pairings, is called a sum of connected expressions with pairing (SCE-pairings). An SCE-pairings e is said to have equal choice property if each component CE-pairings e_i of the sum also has it.*

Example 11. The expression $e = e_1 + e_2$ where $e_1 = fsync((ab+ac)^*, (ad+ae)^*)$ is the *CE-pairings* of Example 8 and $e_2 = fsync((ab+ac)^*a, (ad+ae)^*a)$ is the *CE-pairings* of Example 9 is an *SCE-pairings* with equal choice property, as both e_1 and e_2 have it.

Definition 27 (*SCE* with cables)). *An SCE $e = e_1 + \cdots + e_m$ where each e_i is a CE-cables, then e is called a sum of connected expressions with cables (SCE-cables). An SCE-cables e is said to have **equal source property** if each component CE-cables e_i of the sum also has it. An SCE-cables e has **product derivatives property** if each component CE-cables e_i of the sum has it.*

Example 12. The expression $e' = e_3 + e_4$ where $e_3 = fsync((ab+ac)^*, (ad+ae)^*)$ is the *CE-cables* of Example 8 and $e_4 = fsync((ab+ac)^*a, (ad+ae)^*a)$ is the *CE-cables* of Example 9.

As an example of how derivatives of *SCE-cables* from derivatives of *SCE-pairings* differ even while having identical components, see that $fsync(r_2, s_3) \in Der_a(e)$ (of Example 11) but it does not belong to $Der_a(e')$.

6 Connected Expressions and Product Systems

To get a product system with globals having subset-acceptance, from a sum of connected expression, we use an earlier result from [14], where construction of *PS-globals* with product-acceptance was given from a *CE-cables*.

For each set of derivatives (pre-blocks and after-effects), of a component regular expression, we constructed an unique state of local component, which gives us product moves property in the constructed product system if we have product derivatives property for given *CE-cables*.

Lemma 6 (*CE-cables* to *PS-globals* with product-acceptance [14]). *Let e be a CE-cables, defined over a distribution Σ. Then for the language of e, we can compute a PS-globals with product-acceptance linear in the size of expression e. Further, if e had equal-source, then system A has same source property; and, if e had product-derivatives then A has product moves property.*

Using Lemma 6, we get *PS-globals* with subset-acceptance, from sum of connected expressions.

Theorem 9 (*SCE*-cables to *PS-globals* with subset-acceptance). *Let e be an sum of connected expression defined over Σ. Then we can construct a PS-globals A with subset-acceptance for the language of e. If e had equal source property, then has same source property. In addition, if e has product derivatives property then A has product moves property.*

Example 13. For *SCE-cables* e' of Example 12, we can construct *PS-globals* with same source property and subset-acceptance $\mathcal{D}$ of Example 6, accepting language L_s, using Theorem 9.

A language preserving, construction of connected expressions with equal source property, from a *PS-globals* with same source property and product-acceptance, was given in [14]. Since each local state–whether a source state or a target state of local move–is mapped uniquely to a set of derivatives of component regular expression, we have product-derivatives property for the expression, if the product system had product-moves property.

Lemma 7 (*PS-globals* with product-acceptance to *CE*-cables [14]). *Let Σ be a distributed alphabet and, A be a product system with globals and product-acceptance, defined over Σ. For the language of A, we can construct a connected expression e with cables, exponential in the size of the given product system. Further, if product system has same source property then connected expression with cables has equal source property, in addition, if it has product-moves property then connected expression with cables has product-derivatives property.*

Now using Lemma 7, we get a sum of connected expressions for *PS-globals* with subset-acceptance.

Theorem 10 (*PS-globals* with subset-acceptance to *SCE*-cables). *Let A be a product system with globals and subset-acceptance, defined over distribution Σ. For the language of A, we can construct a sum of connected expression e with cables. And, if product system has same source property then e has equal source property. Also, in addition, if A has product moves property then e has product derivatives property.*

Example 14. For a *PS-globals* with same source property and subset-acceptance $\mathcal{D}$ of Example 6, accepting language L_s, we can construct an *SCE-cables* e' of Example 12 using Theorem 10.

Using equivalence of *PS-matchings* with product-acceptance and *CE-cables*, from [15, 16], and Corollary 1 we get language equivalent *SCE-pairings* for *PS-matchings* with subset-acceptance, and vice-versa.

Theorem 11 (*PS-matchings* with subset-acceptance to *SCE*-pairings). *Let A be a product system with conflict-equivalent and consistent matchings, having subset-acceptance. For the language of A, we can construct a sum of connected expression e with equal choice and consistent pairings.*

We have the reverse translation given as below.

Theorem 12 (*SCE*-pairings to *PS-matchings* with subset-acceptance). *Let Σ be a distributed alphabet and a sum of connected expression e defined over it, with equal choice and consistent pairings. Then for its language we can construct a product system with conflict-eauivalent and consistent matchings, having subset-acceptance.*

7 Conclusion

In this paper, we have given a language (L_s of Example 2) which can be accepted by "free choice" Zielonka automata. This language is not accepted by any synchronous product or direct product. We have also given a language (L_4 of Example 4) which can be accepted by a "free choice" synchronous product and not by any direct product. A language which can be accepted by "free choice" direct product (L_3 of Example 3) was given in [15]. With this we have a hierarchy of labelled free choice nets similar to automata over distributed alphabets. In addition we have defined Zielonka automata with product acceptance condition and its "free choice" restriction. We have given language (L_p of Example 1) of this class. We used this intermediate automata to obtain Kleene theorem for "free choice" Zielonka automata.

We give below the summary of correspondences established for the nets, automata over distributed alphabets and expressions. To get an expression, for the language of a labelled 1-bounded and S-coverable free choice net (with or without distributed choice) having a finite set of final markings, we use Theorem 4 and Theorem 10. In the reverse direction, we use Theorem 9 to get product sytem

with subset acceptance from expressions and then Theorem 3 to get a language equivalent free choice net system.

If the labelled free choice net had distributed choice and has a finite set of final markings, we have an alternate syntax for it. We first use Theorem 5 to get equivalent product system, and then Theorem 11, to get equivalent expressions for the product system constructed. In the reverse direction, we use Theorem 12 and then Theorem 6.

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. Desel, J., Esparza, J.: Free choice Petri nets. Cambridge University Press, New York, USA (1995)
4. Garg, V.K., Ragunath, M.: Concurrent regular expressions and their relationship to petri nets. Theoret. Comp. Sci. 96(2), 285–304 (1992)
5. Grabowski, J.: On partial languages. Fundam. Inform. 4(2), 427–498 (1981)
6. Hack, M.H.T.: Analysis of production schemata by Petri nets. Project Mac Report TR-94, MIT (1972)
7. Jantzen, M.: Language theory of petri nets. In: ACPN. LNCS, vol. 254 (1987)
8. Lodaya, K.: Product automata and process algebra. In: SEFM. IEEE (2006)
9. Lodaya, K., Mukund, M., Phawade, R.: Kleene theorems for product systems. In: DCFS, Proceedings. LNCS, vol. 6808. Springer (2011)
10. Mirkin, B.G.: An algorithm for constructing a base in a language of regular expressions. Engg. Cybern. 5, 110–116 (1966)
11. Mukund, M.: Automata on distributed alphabets. In: D'Souza, D., Shankar, P. (eds.) Modern Applications of Automata Theory. World Scientific (2011)
12. Petersen, J.L.: Computation sequence sets. Journal of Computing and Systems Science 13(1), 1–24 (1976)
13. Phawade, R.: Labelled Free Choice Nets, finite Product Automata, and Expressions. Ph.D. thesis, Homi Bhabha National Institute (2015)
14. Phawade, R.: Kleene theorems for labelled free choice nets without distributed choice. In: Cabac, L., Kristensen, L.M., Rölke, H. (eds.) Proc. PNSE. CEUR Workshop Proceedings, vol. 1591, pp. 132–152. CEUR-WS.org (2016)
15. 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)
16. Phawade, R., Lodaya, K.: Kleene theorems for synchronous products with matching. Transactions on Petri nets and other models of concurrency X, 84–108 (2015)
17. Zielonka, W.: Notes on finite asynchronous automata. Inform. Theor. Appl. 21(2), 99–135 (1987)