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 Ragunath [7] and other authors Lodaya [14]. 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 [12]. 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 [11]. On the other hand they can be used for axiomatizing language equivalence [23,13] 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 [8], Garg and Ragunath [7] and other authors [14], where renaming operator has been used in the syntax to disambiguate synchronizations. In earlier [15,21,22] 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 stucture, 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 [17].

A restriction of our earlier work [19], 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.

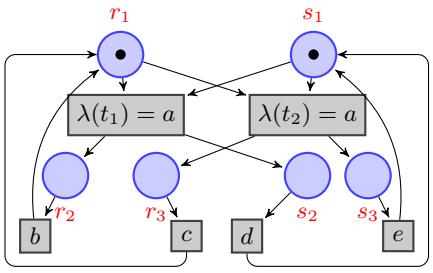

Fig. 1. Free Choice Net without distributed choice

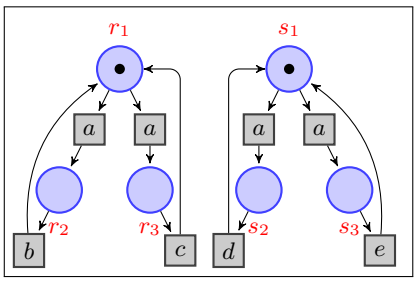

Fig. 2. S-cover of the net in Fig. 1

The Figure 1 shows an example 1-bounded labelled net which is free choice but does not satisfy this property. A short proof [18] using a characterization of direct product languages [17] shows that the language of this net with final marking $\{r_1, s_1\}$ 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 $\mathsf{fsync}((ab + ac)^*, (ad + ae)^*)$, 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 $(\mathsf{fsync}(ab, ad) + \mathsf{fsync}(ac, ae))^*$, the left choices (moves (r_1, a, r_2) and (s_1, a, s_2)) and the right choices (moves (r_1, a, r_3) and (s_1, a, s_3)) 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 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

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

For a set S let $\wp(S)$ denote the set of all subsets of S. To simplify notations sometimes we may denote a singleton set like $\{r\}$ by its element r.

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.*

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

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 $M[t\rangle M'$ or $M[\lambda(t)\rangle M'$. A firing sequence (finite or infinite) $\lambda(t_1)\lambda(t_2)\ldots$ is defined by composition, from $M_0[t_1\rangle M_1[t_2\rangle \ldots$ For every $i \leq j$, we say that M_j is reachable from M_i. 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* $Lang(N, M_0, \mathcal{G}) = \{\lambda(\sigma) \in \Sigma^* \mid \sigma \in T^* \text{ and } M_0[\sigma\rangle M, \text{ for some } M \in \mathcal{G}\}$.

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 $(\Sigma_1, \Sigma_2, \ldots, \Sigma_k)$ of Σ. The next definition appears in several places for unlabelled nets, starting with [10].

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 $S_1, S_2, \ldots, S_k$ of places S, such that $S = S_1 \cup S_2 \cup \ldots S_k$ 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 $S_1 \times S_2 \times \ldots \times S_k$. As before [21], we enforce the "direct product" condition on the set of final markings.

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 $\langle q_1, q_2, \ldots q_k \rangle \in \mathcal{G}$ and $\langle q_1', q_2', \ldots q_k' \rangle \in \mathcal{G}$ then $\{q_1, q_1'\} \times \{q_2, q_2'\} \times \ldots \times \{q_k, q_k'\} \subseteq \mathcal{G}$.*

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 $\pi(t) = \Pi_{i \in loc(a)}(t^\bullet \cap S_i)$ 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 generalized from [19,18] 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 Σ 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 $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$ and the labelled net system $(N = (S, T, F, \lambda), \{r_1, s_1\}, \{\{r_1, s_1\}\})$ shown in Figure 1 defined over it. 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. Since N is S-decomposable, we can write its markings as tuples, therefore it can be alternately described as $(N, (p_1, p_2), \{(p_1, p_2)\})$. 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 $\pi(t_2) = \{(r_3, s_3)\}$. Therefore we have $post(C_a) = \{(r_2, s_2), (r_3, s_3)\}$. Since $postdecomp(C_a) \not\subseteq post(C_a)$, this cluster does not have distributed choice, so the net system does not have it.

Example 2 (A net with distributed choice property). Consider the labelled net system $(N, (p_1, p_2), \{(p_1, p_2)\})$ of Figure 3, defined over distributed alphabet $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$. Its two S-components with sets of places $S_1 = \{p_1, p_3, p_4\}$ and $S_2 = \{p_2, p_5\}$, are shown in Figure 4. For cluster C of p_1, we have $C_a = \{t_1, t_2\}$, $C_a[1] = \{p_3, p_4\}$ and $C_a[2] = \{p_5\}$, hence $postdecomp(C_a) = \{(p_3, p_5), (p_4, p_5)\}$. The post-products $\pi(t_1) = \{(p_3, p_5)\}$ and $\pi(t_2) = \{(p_4, p_5)\}$ give $post(C_a) = \{(p_3, p_5), (p_4, p_5)\}$. Hence $postdecomp(C_a) = post(C_a)$. 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.

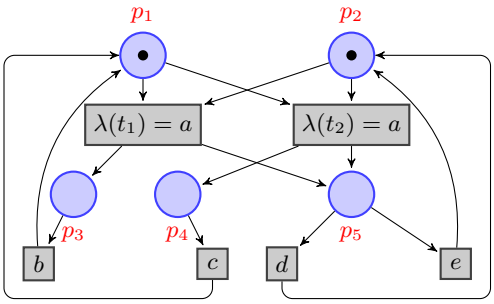

Fig. 3. Labelled Free Choice Net with distributed choice

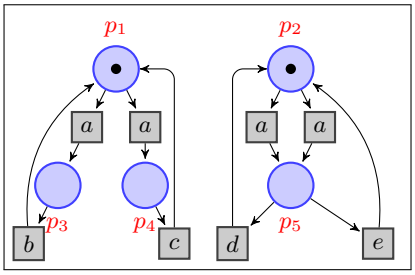

Fig. 4. S-cover of the net in Fig. 3

2.2 Automata and their properties

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* ***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: $L_a^X = \{xay \in L \mid \exists p \text{ in } X \text{ such that } p_0 \xrightarrow{x} p \xrightarrow{ay} G_i\}$, $Pref_a^X(L) = \{x \mid xay \in L_a^X\}$, and $Suf_a^X(L) = \{y \mid xay \in L_a^X\}$. 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 was [21] defined for a single place; 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: $L_a^{(X,Y)} = \{xay \in L \mid \exists p \in X \text{ and } \exists q \in Y \text{ such that } p_0 \xrightarrow{x} p \xrightarrow{a} q \xrightarrow{y} G_i\}$, $Pref_a^{(X,Y)}(L) = \{x \mid xay \in L_a^{(X,Y)}\}$ and $Suf_a^{(X,Y)}(L) = \{y \mid xay \in L_a^{(X,Y)}\}$. 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 $L(A_i)$.*

Let L denote the language of automaton shown in Figure 5. The set $X = \{p_2, p_4\}$ a-bifurcates L and the tuple $(X, \{p_3\})$ a-funnels L. The set $Z = \{p_1, p_3\}$ does not a-bifurcate L because, for the word $aaaaaa$ in L, we have aaa in $Suf_a^{p_3}(L)$ (hence in $Suf_a^{Z}(L)$) and ε in $Pref_a^{p_1}(L)$ (so in $Pref_a^{Z}(L)$), but the word $\varepsilon \cdot a \cdot aaa \notin L$. We can use the same string to see that $(Z, \{p_1, p_2\})$ do not a-funnel L.

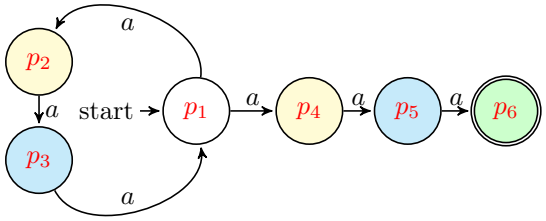

Fig. 5. Automaton with $Lang((aaa)^*aaa)$

2.3 Product Systems

Fix a distribution $(\Sigma_1, \Sigma_2, \ldots, \Sigma_k)$ of Σ. 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 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. The initial product state of A is $R^0 = (p_1^0, \ldots, p_k^0)$, while $G = \Pi_{i \in Loc} G_i$ denotes the final states of A. Let $\Rightarrow_a = \Pi_{i \in loc(a)} \rightarrow_a^i$. The set of all global moves of A is $\Rightarrow = \bigcup_{a \in \Sigma} \Rightarrow_a$. Then for a global move g in $\Rightarrow_a$, let $g[i]$ denote its i-th component–local a-move–belonging to A_i, for all i in $loc(a)$.

Definition 12 ([22]). *A product system A is said to have **separation of labels** if for all $i \in Loc$, and for all global actions a, if $\langle p, a, p' \rangle, \langle q, a, q' \rangle \in \rightarrow_i$ then $p = q$.*

The product system shown in Figure 6, has separation of labels property.

Definition 13 ([15]). *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 $A =$*

$\langle A_1, \ldots, A_k \rangle$ *a* ***conflict-equivalent product system*** *if for every global action* $a \in \Sigma$*, and for all* $i, j \in loc(a)$*, every* a*-move in* A_i *is conflict-equivalent to every* a*-move in* A_j*.*

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*** *is a product system with relations* $globals(a)$*, for each global action* a *in* Σ*.*

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 $j \in loc(a), \langle R[j], a, Q[j] \rangle \in \rightarrow_j$, and for all $j \notin loc(a), R[j] = Q[j]$. Runs of a product system with globals, are defined in the same way where, an additional requirement of $\Pi_{j \in loc(a)}(\langle R[j], a, Q[j] \rangle \in globals(a)$, has to be satisfied. With abuse of notation sometimes we use $pre(a)$ 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 $Lang(A)$ 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.*

The product system A shown in Figure 6 has same source, as both the a-*globals* in the given $globals(a)$ relation have the same set of pre-places.

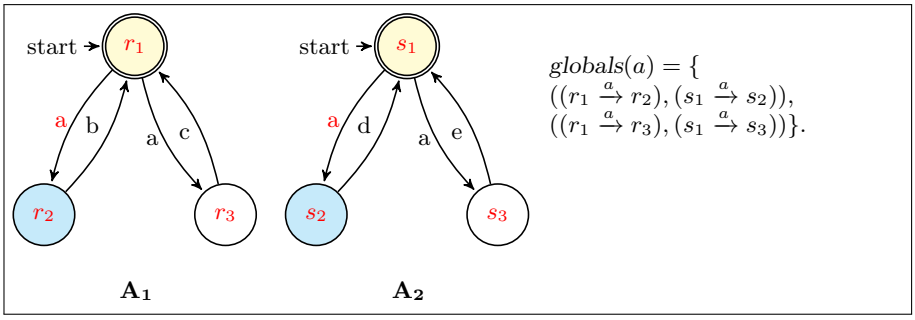

Fig. 6. Product system $A = \langle A_1, A_2 \rangle$ used in Example 3

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 $A = \langle A_1, A_2, \ldots, A_k \rangle$ over distribution Σ, 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.*
- $T = \bigcup_a globals(a)$, *for all actions a in Σ. We also let $T_i = \{\lambda^{-1}(a) \mid a \in \Sigma_i\}$.*
- *The labelling function λ labels by action a the transitions in $globals(a)$.*
- *The flow relation $F = \{(p, g), (g, q) \mid g \in T_a, g[i] = \langle p, a, q \rangle, i \in loc(a)\}$. Let F_i be its restriction to the transitions T_i for $i \in loc(a)$.*
- $M_0 = \{p_1^0, \ldots, p_k^0\}$, *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., $Lang(N, M_0, \mathcal{G}) = Lang(A)$.*

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 $\{1, 2, \ldots, k\}$, we define a product system $A = \langle A_1, \ldots, A_k \rangle$.*

- $P_i = S_i$, *p_i^0 the unique state in $M_0 \cap P_i$.*
- $\rightarrow_i = \{\langle p, \lambda(t), p' \rangle \mid t \in T_i$ *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.*

- $globals(a) = \bigcup_{t \in T_a} (\Pi_{i \in loc(a)} t[i])$.
- $G = \mathcal{G}$

Again we can generalize results obtained in the thesis [19].

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 [21,19], 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 3. 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 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. We will use syntactic entities–called derivatives–associated with regular expressions which are known since the time of Brzozowski [2], Mirkin [16] and Antimirov [1].

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. Similarly, we can syntactically check if $\varepsilon \in Lang(s)$.

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 $\mathsf{Der}_a(s)$ are defined as follows.*

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

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

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

We have the Antimirov derivatives $\mathsf{Der}_a(ab + ac) = \{b, c\}$ and $\mathsf{Der}_a(a(b + c)) = \{b + c\}$, 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 $L_a^D = \{xay \mid xay \in L, \exists d \in \mathsf{Der}_x(s) \cap D, \exists d' \in \mathsf{Der}_{ay}(d)$ 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 $L_a^D = Pref_a^D(L) \cdot a \cdot Suf_a^D(L)$.*

Let $Part_a(s)$ denote a partition of the a-sites of s into blocks such that each block (that is, element of the partition) a-bifurcates L. Earlier [21] we have given a syntactic scheme for obtaining a partition, for which we proved the following.

Proposition 1 ([21]). *Every block D of the partition $Part_a(s)$ 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 $E \subseteq Der_a(B)$, we define the relativized languages*

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

Also 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* $\text{Init}(B, E)$ *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* $loc(a)$ *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 4. Consider a regular expression $p_1 = (aaa)^*aaa$. Let L denote the language of p_1. The set of all its derivatives is $\mathsf{Der}(p_1) = \{p_1, p_2 = aa(aaa)^*aaa, p_3 = a(aaa)^*aaa, p_4 = aa, p_5 = a, p_6 = \varepsilon\}$. All these derivatives are also shown in Figure 5. 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 $Part_a(p_1) = \{D_1 = \{p_1\}, D_2 = \{p_2, p_4\}, D_3 = \{p_3, p_5\}\}$. 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 $Suf_a^{p_3}(L)$ (and hence in $Suf_a^Z(L)$) and we have string ε 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. For set Z, we have $\mathsf{Der}_a(Z) = \{p_2, p_4, p_1\}$. Considering the same string as above, we can see that tuple $(Z, \{p_1, p_2\})$ 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 $(\Sigma_1, \ldots, \Sigma_k)$, 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 $(\Sigma_1, \Sigma_2, \ldots, \Sigma_k)$ of alphabet Σ.

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

Definition 22. *A connected expression $e = \mathsf{fsync}(s_1, s_2, \ldots, s_k)$ over Σ, is said to have* ***equal choice*** *if, for all global actions a in Σ and i, j in $loc(a)$, for an a-site r of s_i and an a-site r' of s_j, $Init(r) = Init(r')$.*

A connected expression $e = \mathsf{fsync}(s_1, s_2, \ldots, s_k)$ over Σ, 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 $e = \mathsf{fsync}(s_1, s_2, \ldots, s_k)$, its language is given as $Lang(e) = Lang(s_1) \| Lang(s_2) \| \ldots \| Lang(s_k)$, where the synchronized shuffle $L = L_1 \| \ldots \| L_k$ is defined as: $w \in L$ iff for all $i \in \{1, \ldots, k\}, w{\downarrow_{\Sigma_i}} \in L_i$. The definition of derivatives extended to connected expressions [21] is given as follows. The expression 0 has no derivatives on any action. Given an expression $e = \mathsf{fsync}(s_1, s_2, \ldots, s_k)$, its derivatives are defined by induction using the derivatives of the s_i on action a as, $Der_a(e) = \{\mathsf{fsync}(r_1, \ldots, r_k) \mid \forall i \in loc(a), r_i \in Der_a(s_i);$ 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 $e = \mathsf{fsync}(s_1, s_2, \ldots, s_k)$ 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-cables(e). A block B of $Part_a(s_i)$ appears in an a-cable D if there exists j in $loc(a)$ and $Y \subseteq \mathsf{Der}_a(B)$, such that $D[j] = (B, Y)$. For any a-cable D its set of pre-blocks is ${}^\bullet D = \cup_{i \in loc(a)} \{B_i \mid B_i$ appears in $D\}$.*
For expression e, let ***cables(a)*** *$\subseteq a$-cables(e), such that for all i in $loc(a)$: (1) for all (B, E) in a-ducts(s_i), there exists an a-cable D in cables(a) and location j in $loc(a)$, such that $D[j] = (B, E)$ (2) for all (B, E) and (B', E') in a-ducts(s_i) with $(B, E) \neq (B', E')$, if $B = B'$ then $E \cap E' = \emptyset$.*
A ***connected expression with cables*** *is a connected expression with relations cables(a) of it, for each global action a in Σ.*

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

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

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

For expressions $d = \mathsf{fsync}(r_1, \ldots, r_k)$ we use $d[i]$ for r_i. Define $Init(d) = \{a \in \Sigma \mid Der_a(d) \neq \emptyset\}$. If $a \in Init(d)$ we call d an ***a-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 Σ defined using derivatives as below. $Lang(e) = \{w \in \Sigma^* \mid \exists e' \in \mathsf{Der}_w(e)$ *such that* $\varepsilon \in Lang(r_i)$, *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 have* ***equal source property*** *if for any pair of two cables sharing a common pre-block have same set of pre-blocks.*

Example 5. Let $e = \mathsf{fsync}(r_1, s_1)$ 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 $\mathsf{Der}(r_1) = \{r_1, r_2 = b(ab+ac)r_1, r_3 = c(ab+ac)r_1, r_4 = (ab+ac)r_1, r_5 = br_1, r_6 = cr_1, \}$ and $\mathsf{Der}(s_1) = \{s_1, s_2 = d(ad + ae)s_1, s_3 = e(ad + ae)s_1, s_4 = (ad + ae)s_1, s_5 = ds_1, s_6 = es_1, \}$. 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, their set of post-effects $\{r_2\}$ and $\{r_3\}$ are disjoint, and similarly for a-ducts $d_3[1]$ and $d_4[1]$ of r_1, with the same pre-block r_4, their sets of post-effects $\{r_5\}$ and $\{r_6\}$ are disjoint. Similarly this condition holds for a-ducts of s_1. This expression do not have unique sites, as its componet expressions do not have it. It has equal source property as ${}^\bullet d_1 = {}^\bullet d_2 = \{r_1, s_1\}$ and ${}^\bullet d_3 = {}^\bullet d_4 = \{r_4, s_4\}$. We have $\mathsf{Der}_a(e) = \{fsync(r_2, s_2), fsync(r_3, s_3)\}$, but $fsync(r_2, s_3)$ is not in $\mathsf{Der}_a(e)$. Simplifying notation, the set of reachable derivatives is $\mathsf{Der}(e) = \{(r_1, s_1), (r_2, s_2), (r_4, s_2), (r_2, s_4), (r_4, s_4), (r_3, s_3), (r_4, s_3), (r_3, s_4), (r_5, s_5), (r_6, s_6), (r_5, s_1), (r_1, s_5), (r_6, s_1), (r_1, s_6)\}$.

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. We refer the reader to [20] for detailed proofs of Lemma 4, Lemma 6, Theorem 3.

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.

Before giving a formal proof, we present the key idea behind it. 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) for a-move (p, a, 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. 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 $\mathsf{Der}(s)$, and corresponds to the place q in the original automaton A_i. To show that a-move (p, a, q) is indeed captured by (B_p, a, B_q), we need prove that a word xay accepted by automaton and passing through p and q should also pass through derivatives belonging to these sets, and the reverse of this should also hold.

Lemma 4. *Let Σ be a distributed alphabet. Let A be a product system over Σ 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 Σ, 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.*

Proof. Let $A = \langle A_1, \ldots, A_k\rangle$ 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 Σ_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 Σ_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 Γ_i

and new sequential system H_i over it. Repeating this for each sequential system A_i we get a new product system $H = \langle H_1, \ldots, H_k \rangle$ over distributed alphabet $\Gamma = (\Gamma_1, \ldots, \Gamma_k)$. 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 Γ_i and Σ_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 Γ_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 Γ_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 $e' = (s'_1, \ldots, s'_k)$ over alphabet Γ for language of product H and it has unique sites property. Hence, for each (local or global) action c in Γ_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 $L_a^{(p,q)} = Pref_a^p(L) \cdot a \cdot Suf_a^{(p,q)}(L)$ and $L_a^{(B'_p, B'_q)} = Pref_a^{B'_p}(L) \cdot a \cdot Suf_a^{(B'_p, B'_q)}(L)$. 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 Σ_i. See that we had not renamed any global action of Σ, 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 $e = (s_1, \ldots, s_k)$ over distributed alphabet Σ, 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 ε is in $Lang(s'_i)$ iff ε 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 $\mathsf{Der}_{x'}(s'_i)$, and r' in $\mathsf{Der}_{x'a}(s'_i)$, and s'_f in $\mathsf{Der}_{y'}(r')$ such that ε 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 $\mathsf{Der}_{xa}(s_i)$, where x is obtained by renaming word x'. By induction hypothesis, derivative d is in $\mathsf{Der}_x(s_i)$, therefore it suffices to prove that derivative r, the word obtained by renaming back r', is in $\mathsf{Der}_a(d)$. This is done by structural induction on d'.

As a consequence 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 also, for any w in $Lang(s'_i)$ we get a unique word w' in $Lang(s'_i)$. We extend this renaming to a set of derivatives; so for a set of derivatives $B' \subseteq \mathsf{Der}(s'_i)$ we get the set $B \subseteq \mathsf{Der}(s_i)$, 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. □

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 Σ. Then for the language of A, we can compute a connected expression with cables, having equal source property.*

Proof Sketch. Given product system $A = \langle A_1, \ldots, A_K \rangle$ may not have separation of labels property, i.e. for a global action a, a component A_i may have many places with outgoing local a-moves. Using globals we rename local a-moves appropriately across all such component A_i's, to get a new product system H having separation of labels property over a new distributed alphabet. Then we apply Lemma 4 to get an expression for product H, which is renamed back to get an expression for the language of A. □

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 $Lang(e)$ 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 Σ 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 $\mathrm{Lang}(e)$, and having separation of labels along with same source property. Moreover, for each a in Σ, for each a-duct we produce one local a-move. And, for each a-cable we get one global a-move of A.*

Proof Sketch. Consider a global action a and a location i in $loc(a)$. For an a-cable d we construct an a-global by giving a local a-move, for each a-duct $d[i] = (B, X)$ of s_i. Since B is a block in $Part_a(s_i)$, we get a unique place p for it using a-bifurcation, but X may not be a block and there may exist another a-effect Y of B (so (B, Y) can be a component of another a-cable d'), so we can not use bifurcation directly to get a place for X. Here we employ the trick of renaming X to X' so that X' is a block. Take a local action c in $Init(X)$. As X and Y are disjoint, each distinct occurrence of c in s_i corresponds to either X or Y, so we rename each occurrence of c by c^X to get s'_i over a new alphabet Γ'_i, and similarly X' from X, and B' from B. This renaming is well-defined. Now X' is a block in the $Part_{c^X}(s'_i)$. We construct A'_i from s'_i, where we get a unique place q for X' also, therefore we get a local move (p, a, q) corresponding to a-duct (B', X') of s'_i. Renaming back c^X to c in A'_i to get A_i over Σ_i, for language of s_i, we get a local a-move (p, a, q) in A_i corresponding to a-duct (B, X). □

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 Σ. Then for the language of e, we can compute a product system with globals having same source property.*

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

Since expression e has equal source property, the set of $cables(a)$ 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 $loc(a) = \{1, \ldots, l\}$. Let $(B_1, \ldots, B_l)$ be the pre-tuple of *a-cables* of bucket Z of the partition of $cables(a)$. 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 Σ'_i.

So we get connected expression $e' = (s'_1, \ldots, s'_k)$ over distributed alphabet $\Sigma' = (\Sigma'_1, \ldots, \Sigma'_k)$. 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 $A' = \langle A'_1, \ldots, A'_k \rangle$ by applying Lemma 6 over distributed alphabet Σ' 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 Σ. 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 $Lang(e) = Lang(A)$. □

Example 6. Consider the connected expression $e = fsync(r_1, s_1)$ of Example 5. A $cables(a)$ relation for e is A set $cables(a) = \{d_1 = ((r_1, r_2), (s_1, s_2)), d_2 = ((r_1, r_3), (s_1, s_3)), d_3 = ((r_4, r_5), (s_4, s_5)), d_4 = ((r_4, r_6), (s_4, s_6))\}$. As shown in Example 5, expression e equal source, and it does not have unique sites property.

In the construction, a-cables d_1 and d_2 are put into one bucket z_1 and 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 after renaming a-duct (r_1, r_2) becomes an a^1-duct (r'_1, r'_2) and a-duct (s_4, s_5) becomes an a^2-duct (s'_4, s'_5), so on. So we have two sets of cables now for two global actions a^1 and a^2: $cables(a^1) = \{d'_1 = ((r'_1, r'_2), (s'_1, s'_2)), d'_2 = ((r'_1, r'_3), (s'_1, s'_3))\}$ 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

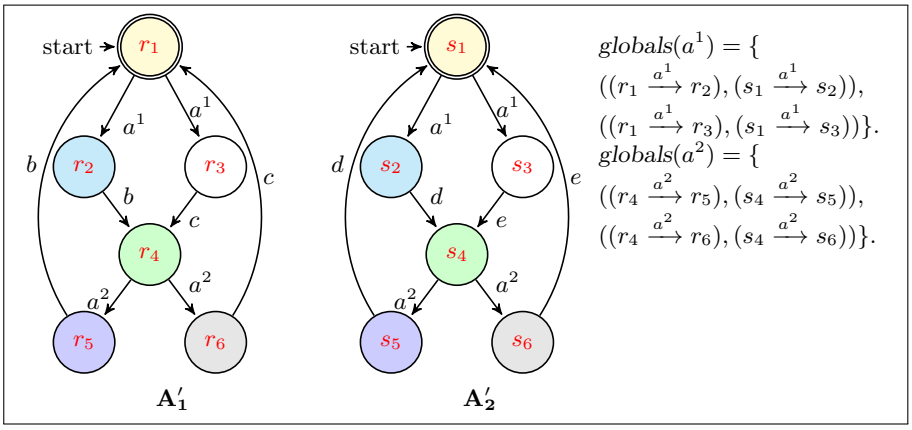

Fig. 7. Product system $A' = \langle A'_1, A'_2 \rangle$ used in Example 6

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'_is 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.

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 [9].

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. Gruber, H., Holzer, M.: Finite automata, digraph connectivity, and regular expression size. In: ICALP(2). pp. 39–50 (2008)
10. Hack, M.H.T.: Analysis of production schemata by Petri nets. Project Mac Report TR-94, MIT (1972)
11. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation - international edition (2. ed). Addison-Wesley (2003)
12. Kleene, S.C.: Representation of events in nerve nets and finite automata. In: Automata studies. pp. 3–41. Princeton (1956)
13. Kozen, D.: A completeness theorem for kleene algebras and the algebra of regular events. Inf. Comput. 110(2), 366–390 (1994)
14. Lodaya, K.: Product automata and process algebra. In: SEFM. pp. 128–136. IEEE (2006)
15. Lodaya, K., Mukund, M., Phawade, R.: Kleene theorems for product systems. In: DCFS, Proceedings. LNCS, vol. 6808. Springer (2011)
16. Mirkin, B.G.: An algorithm for constructing a base in a language of regular expressions. Engg. Cybern. 5, 110–116 (1966)
17. Mukund, M.: Automata on distributed alphabets. In: D'Souza, D., Shankar, P. (eds.) Modern Applications of Automata Theory. World Scientific (2011)
18. Phawade, R.: Direct product representation of labelled free choice nets. Int. J. Comp. Appl. 99(16), 1–8 (2014)
19. Phawade, R.: Labelled Free Choice Nets, finite Product Automata, and Expressions. Ph.D. thesis, Homi Bhabha National Institute (2015)
20. Phawade, R.: Kleene theorem for labelled free choice nets without distributed choice. Tech. rep., Indian Institute of Technology Bombay (2016), `https://www.cse.iitb.ac.in/internal/techreports/reports/TR-CSE-2016-79.pdf`
21. Phawade, R., Lodaya, K.: Kleene theorems for labelled free choice nets. In: 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)