Kleene Theorems for Labelled Free Choice Nets

Ramchandra Phawade and Kamal Lodaya

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

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

1 Introduction

Petri nets are an excellent visual representation of concurrency. But like any graphical notation they are less amenable to syntax. For finite automata, Kleene's regular expressions provide us with a formalism where we can switch between the graphical and the textual. For 1-bounded Petri nets, equivalent syntax has been provided by Grabowski [Gra81], Garg and Ragunath [GR92] and other authors. Here we place restrictions on this syntax in an effort to match the 1-bounded labelled free choice nets, a very well-studied subclass [Hac72] with more efficient analysis and algorithms [DE95]. It has been claimed that free choice nets can be useful in business process modelling [SH96], but our motivation is more conceptual than dictated by business concerns.

As is usual when dealing with subclasses, this turns out to be challenging. We also follow the example of finite automata and work directly with labelled nets, not relying on a renaming operator in the syntax. As in our earlier paper [LMP11], we rely on an intermediate formalism, "direct" products of automata, which are known to be weaker than 1-bounded nets [Zie87,Muk11]. There we identified a subclass called FC-products, and a graph-theoretic property called "structural cyclicity", for which we presented an equivalent syntax which was restricted to being without nested Kleene star operators.

The improvement in this paper is that on the system side we have an enlarged subclass called **FC-matching products**. On the syntax side we drop the structural cyclicity condition and do not place any restriction on the Kleene stars, thus (unlike in our earlier paper) including all regular expressions. We do have global restrictions. A "pairing" condition identifies synchronizations which will take place at run-time. Assuming a communication alphabet $\{a, b, c\}$, the expression $(a + a + b)(a + c + c)$ the a's in the two groups of parentheses will be paired into different synchronizations. Correspondingly we have a "matching" condition in the product systems. The matching condition produces free choice nets (and the converse also holds). Our proofs go through a subclass where communications are labelled with the place from which they are issued.

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:

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

Definition 1. *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 regular expression over alphabet Σ_i defining a nonempty language is given by:

$$s ::= a \in \Sigma_i | s_1 \cdot s_2 | s_1 + s_2 | s_1^*$$

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 associated with regular expressions which are known since the time of Brzozowski [Brz64], Mirkin [Mir66] and Antimirov [Ant96].

For each regular expression s over Σ_i, its initial actions form the set $Init(s) = \{a \mid av \in Lang(s) \text{ and } v \in \Sigma_i^*\}$ which can be defined syntactically. Similarly, we can syntactically check whether the empty word $\varepsilon \in Lang(s)$. Next we syntactically define derivatives [Ant96].

Definition 2. *Given regular expression s and symbol a, the partial derivatives of s wrt a, written $Der_a(s)$ are defined as follows.*

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

Inductively $Der_{aw}(s) = Der_w(Der_a(s))$.

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

We have the Antimirov derivatives $Der_a(ab + ac) = \{b, c\}$ and $Der_a(a(b + c)) = \{b + c\}$, whereas the Brzozowski a-derivative [Brz64] (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 $Init(D)$. We syntactically partition the a-sites of s, each set of the partition containing those coming from a common source derivative, as follows.

Definition 3. *For partitions X_1, X_2 with blocks D_1, D_2 containing elements d_1, d_2 respectively, we use the notation $(X_1 \cup X_2)[d/d_1, d_2]$ for the modified partition $((X_1 \cup X_2) \setminus \{D_1, D_2\}) \cup \{(D_1 \cup D_2 \cup \{d\}) \setminus \{d_1, d_2\}\}$.*

$$Part_a(b) = \emptyset \text{ if } a \neq b$$
$$Part_a(a) = \{\{a\}\}$$
$$Part_a(s_1{+}s_2) = \begin{cases} (Part_a(s_1) \cup Part_a(s_2))[s_1{+}s_2/s_1, s_2] & \text{if } a \in Init(s_1{+}s_2) \\ Part_a(s_1) \cup Part_a(s_2) & \text{otherwise} \end{cases}$$
$$Part_a(s_1^*) = \begin{cases} Part_a(s_1)[s_1^*/s_1] & \text{if } a \in Init(s_1) \\ Part_a(s_1) \cdot s_1^* & \text{otherwise} \end{cases}$$
$$Part_a(s_1 \cdot s_2) = \begin{cases} Part_a(s_1)[s_1 \cdot s_2/s_1] \cup Part_a(s_2) & \text{if } \varepsilon \in Lang(s_1) \\ Part_a(s_1) \cdot s_2 \cup Part_a(s_2) & \text{otherwise} \end{cases}$$

The next definition and the following proposition identify the key property of this partition of a-sites for this paper.

Definition 4. *Given a set of derivatives D and an action a, define the prefixes $Pref_a^D(L) = \{x \mid xay \in L, \exists d \in Der_x(L) \cap D, \varepsilon \in Der_{ay}(d)\}$, suffixes $Suf_a^D(L) = \{y \mid xay \in L, x \in Pref_a^D(L)\}$, and the relativized language $L^D = \{xay \mid xay \in L, \exists d \in Der_x(L) \cap D, \varepsilon \in Der_{ay}(d)\}$. We say that the derivatives in set D a-**bifurcate** L if $L^D \cap \Sigma^* a \Sigma^* = Pref_a^D(L)\ a\ Suf_a^D(L)$. If D is the set of all derivatives, we say L is a-**bifurcated**.*

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

Proof. By induction on the definition. □

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 properties of expressions will be important in this paper, where the derivatives are taken for regular expressions and also for the connected expressions defined in the next section.

Definition 5. *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**. It has **deterministic global actions** if for every global action a and every a-site $d \in Der(s)$, $|Der_a(d)| = 1$. It has **unique global actions** if it has both these properties.*

3 Connected Expressions over a Distribution

We have a simple syntax of connected expressions. The s_i can be any regular expressions (of any star-height), which is different from our earlier paper [LMP11].

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

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

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

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

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

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

The definitions of derivatives can be easily extended to connected expressions. 0 has no derivatives on any action. Given $e = fsync(s_1, s_2, \ldots, s_k)$, its derivatives are defined by induction using the derivatives of the s_i on action a:

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

We will use the word derivative for expressions such as $d = fsync(r_1, r_2, \ldots, r_k)$ above (essentially tuples of derivatives of regular expressions), and $d[i]$ for r_i. The number of derivatives can be exponential in k. Define $Init(d)$ to be those actions a such that $Der_a(d)$ is nonempty. If $a \in Init(d)$ we call d an a-**site**. The reachable derivatives are $Der(e) = \{d \mid d \in Der_x(e), x \in \Sigma^*\}$. For example, $fsync(ab, ba)$ has derivatives other than the expression itself, but none of them is reachable.

3.1 Properties of Connected Expressions

We now define some properties of connected expressions over a distribution. These will ultimately lead us to construct free choice nets. All but the last property are PTIME-checkable. The last property requires PSPACE since it runs over all reachable derivatives.

Definition 6. *Let $e = fsync(s_1, s_2, \ldots, s_k)$ be a connected expression over Σ. For a global action a, an a-**pairing** is a subset of tuples $\Pi_{i \in loc(a)} Part_a(s_i)$, the projections of these tuples covering the a-sites in s_i, such that if a block of $Part_a(s_j), j \in loc(a)$ appears in one tuple of the pairing, it does not appear in another tuple. (For convenience we also write $pairing(a)$ as a subset of $\Pi_{i \in loc(a)} Der(s_i)$ which respects the partition.) We call $pairing(a)$ equal choice if for every tuple in the pairing, the derivatives in the tuple have equal choice.*
*We extend the definition to connected expressions. A derivative $fsync(r_1, \ldots, r_k)$ is in pairing(a) if there is a tuple $D \in pairing(a)$ such that $r_i \in D[i]$ for all $i \in loc(a)$. For convenience we may write a derivative as an element of $pairing(a)$. Expression e is said to have **(equal choice) pairing of actions** if for all global actions a, there exists an (equal choice) $pairing(a)$. Expression e is said to be **consistent with a pairing of actions** if every reachable a-site $d \in Der(e)$ is in $pairing(a)$.*

Example 1. Let $(\Sigma_1 = \{a\}, \Sigma_2 = \{a\})$. Expression $fsync(aa, a)$ does not have a *pairing*. The two a's on the left are in different blocks of the partition and they have to pair with one block on the right, which is not allowed.

Example 2. Let $(\Sigma_1 = \{a\}, \Sigma_2 = \{a, b, c, d, f\})$. In expression $e = fsync(aa, bad + caf)$ we have two blocks on the left and two blocks on the right, so we can have a pairing. But e cannot be consistent with any pairing.

Example 3. Let $(\Sigma_1 = \{a, c\}, \Sigma_2 = \{b, c\}), \Sigma_3 = \{a, b, c\})$. Consider this expression $fsync((ac)^*, (bc)^*, (a(b + c))^*)$. Individual regular expressions are $r_1 = (ac)^*$, $r_2 = (bc)^*$ and $r_3 = (a(b + c))^*$. Now we have $r_1' = Der_a(r_1) = c(ac)^*$ and $Init(r_1') = \{c\}$. For r_3 we have, $r_3' = Der_a(r_3) = (b + c)(a(b + c))^*$ and $Init(r_3') = \{b, c\}$. r_1' and r_3' do not have equal choice.

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

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

On the other hand, to check consistency with a pairing of actions, we have to visit each reachable derivative, this can be done in PSPACE. □

4 Product Systems over a Distribution

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

Definition 7. *A sequential system over a set of actions Σ_i is a tuple $A_i = \langle P_i, \rightarrow_i, G_i, p_i^0 \rangle$ 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.

A run of the sequential system A_i on word w is a sequence $p_0 a_1 p_1 a_2, \ldots, a_n p_n$, from set $(P_i \times \Sigma_i)^* P_i$, such that $p_0 = p_i^0$ and for each $j \in \{1, \ldots, n\}$, $p_{j-1} \xrightarrow{a_j} p_j$. This run is said to be accepting if $p_n \in G_i$. The sequential system A_i accepts word w, if there is at least one accepting run of A_i on w. The language $L = Lang(A_i)$ of sequential system A_i is defined as $L = \{w \in {\Sigma_i}^* | w \text{ is accepted by } A_i\}$.

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

Definition 8. *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 relativizations L^R of a language $L \subseteq \Sigma_i^*$ consider projections to place $R[i]$ in A_i.

The initial product state of A is $R^0 = (p_1^0, \ldots, p_k^0)$, 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 global moves of A is $\Rightarrow = \bigcup_{a \in \Sigma} \Rightarrow_a$. Then for a global move

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

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

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

4.1 Properties of Product Systems

The first property for a product system is modelled on the free choice property of nets. It can be checked in PTIME by counting local moves with the same label. We also define another stronger property.

Definition 9. *For global $a \in \Sigma$, an a-matching is a subset of tuples $\Pi_{i \in loc(a)} P_i$, such that if a place $p \in P_j, j \in loc(a)$ appears in one tuple, it does not appear in another tuple. We say a product state R is in an a-matching 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 an a-matching such that for $i, j \in loc(a), \langle p, a, q \rangle \in \rightarrow_i$, the pre-place p is matched to a pre-place p' such that $\langle p', a, q' \rangle \in \rightarrow_j$ and such that all pre-places with a-transitions are covered by the tuples of the matching. A product system A is said to have* ***separation of labels*** *if for all $i \in Loc$, if $\langle p, a, p' \rangle, \langle q, a, q' \rangle \in \rightarrow_i$ then $p = q$.*

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

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

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

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

Definition 10. *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 moves from* p*. If the product system has a matching of labels and we require this whenever* p, p' *are related by the matching, we call the matching* ***conflict-equivalent****. A system having a conflict-equivalent matching is a weaker condition than the system being conflict-equivalent.*

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

Checking that a system is an FC-product or an FC-matching product is in PTIME because one makes a pass through all transitions with the same locations, computing for each pre-place which partition it falls into.

Proposition 4. *Let* A *be an FC-matching product system. For any* i*, if there exist local moves* $\langle p, a, p'\rangle, \langle p, b, p''\rangle$ *in* $\rightarrow_i$*, then* $loc(a) = loc(b)$*.*

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

4.2 Language of a Product System

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

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

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

Proposition 5. $L = Lang(A)$ *is the language of product system* $A = \langle A_1, \ldots, A_k\rangle$ *over distribution* Σ *iff*

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

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

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

Definition 11. *A run of* A *is said to be* ***consistent with a matching of labels*** *if for all global actions* a *and every prefix of the run* $R^0 \stackrel{v}{\Rightarrow} R \stackrel{a}{\Rightarrow} Q$*, the pre-places* $R{\downarrow}loc(a)$ *are in the matching.*

5 Connected Expressions and Product Systems

In this section we prove the main theorems of the paper. To place them in context of our earlier paper [LMP11], there we used a "structural cyclicity" condition which allowed a run to be split into finite parts from the initial product state to itself, since it was guaranteed to be repeated. The new idea in this paper is that runs are split up using matchings which correspond to synchronizations, what happens in between is not relevant for the connections across sequential systems. Hence extending our syntax to allow full regular expressions for the sequential systems does not affect the synchronization properties which are the main issue we are addressing. In Section 6 we outline the connections to labelled free choice nets which are detailed in another paper [PL14].

5.1 Synthesis of Systems from Expressions

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

Now we come to connected expressions, for which we will construct a product of automata.

Lemma 1. *Let e be a connected expression with unique global action sites. Then there exists a product system A with separation of labels accepting $Lang(e)$ as its language. If e had equal choice, then A is conflict-equivalent.*

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

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

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

We proceed from n downwards. For any place d_n in G there is a run from d_n on $\varepsilon \in Lang(d_n)$ in A_i. Inductively assume we have d_j such that there is a run $d_j \xrightarrow{a x_{j+1}} \ldots \xrightarrow{a x_n} d_n$ of A_i, so $x_{j+1} a x_{j+2} \ldots a x_n$ is in $Suf_a(Lang(s_i))$ since d_j is reachable from the initial place. Since there is a run $p \xrightarrow{a x_j} p$ in A_i' there are derivatives d_{j-1}, c_j of e, such that there is a run $d_{j-1} \xrightarrow{a x_j} c_j$ in A_i (when $j = 1$

we get $d_0 \xrightarrow{x_1} c_1$ by this argument). Since c_j quotients to p, it has an a-derivative c such that c is in $Der_{ax_ja}(d_{j-1})$ ($Der_{x_0a}(d_0)$ when $j = 1$). Because d_{j-1} is reachable from the initial place by some v and because some final state is reachable from c, $vx_j \in Pref_a(Lang(s_i))$ which is nonempty. By the unique global action sites condition and Proposition 1, since $x_{j+1} \dots ax_n$ is in $Suf_a(Lang(s_i))$, $vax_jax_{j+1} \dots ax_n$ is in $Lang(s_i)$ and so $x_jax_{j+1} \dots ax_n$ is in $Suf_a(Lang(s_i))$. This means that there is a run from some d_{j-1} on $ax_jax_{j+1} \dots ax_n$ ending in a final state d_n of A_i. So we have the induction hypothesis restored. If $j = 1$ we get d_0 which quotients to p_0 and has a run on w to d_n in G.

So we get a product system $A' = \langle A'_1, A'_2, \dots, A'_k \rangle$ defined over Σ. If the expression had equal choice, this system is conflict-equivalent. Because of the quotienting A' has separation of labels.

$$\begin{aligned} w \in Lang(e) &\text{ iff } \forall i, w{\downarrow_{\Sigma_i}} \in Lang(s_i), \text{ by definition} \\ &\text{iff } \forall i, w{\downarrow_{\Sigma_i}} \in Lang(A'_i) \\ &\text{iff } w \in Lang(A'), \text{ by Proposition 5.} \end{aligned}$$

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

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

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

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

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

for pre-places of a-moves in the product A we have a matching. If the pairing was equal choice, the matching is conflict-equivalent.

If the expression e is consistent with the pairing, all reachable a-sites are in the pairing, so we can partition $Lang(e) \cap \Sigma^* a \Sigma^*$ using the partitions in $Part_a(e)$. Letting D range over blocks of connected expressions, each block D contributes a global action a^D in the renaming, so we get an expression e' such that for every global action a^D, we have the unique a-sites property. Applying Lemma 1, we have the product system A' with separation of labels. By Proposition 3, every $Lang(A'_i)$ is a^D-bifurcated, and using the characterization of Proposition 5, $Lang(A') \cap (\Sigma')^* a^D (\Sigma')^* = Pref_{a^D}(Lang(A')) a^D Suf_{a^D}(Lang(A'))$. Since several actions a^D are renamed to a and the corresponding tuples of pre-places are recorded in the matching, by Proposition 3 and Proposition 5:

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

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

5.2 Analysis of Expressions from Systems

Lemma 2. *Let A be a FC-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 FC-product had deterministic global actions, then so do the regular expressions in the computed expression. If the FC-product was conflict-equivalent, the constructed expression has equal choice.*

Proof. Let $A = \langle A_1, \ldots, A_k \rangle$ be an FC-product with separation of labels, where A_i is a sequential system of A with places P, initial place p_0 and final places G. Kleene's theorem gives us an expression s_i for the language of A_i. We claim the required connected expression is $fsync(s_1, \ldots, s_k)$.

Consider global action a. By separation of labels there is a single state p in A_i enabling a. For simplicity let us assume there is only one global action a enabled at p. Let $Q = P \setminus \{p\}$. Let T be the set of transitions excluding the a-actions enabled at p. We wish to decompose the expression s_i that we started with into paths which go through p and paths which do not. Depending on whether we have a sequential transition $p \xrightarrow{a} p$, or transitions $p \xrightarrow{a} p_j,\ p_j \neq p$, or a combination of these two types, we obtain an expression with the same language as s_i:

$$e_p = \sum_{f \in G} e^T_{p_0,f} + e^T_{p_0,p} e^*_{p,p} e^Q_{p,f},$$

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

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

The superscripts T, Q indicates that these expressions are derived, as in the McNaughton-Yamada construction [MY60], for runs which only use the states Q or transitions T. Whichever be the case, we note that we have an expression with $D^a(e_p) = \{e^*_{p,p}e^Q_{p,f}\}$ as its singleton set of a-sites. If the system had deterministic global actions, the a-site would have only had one a-derivative. This idea can be easily extended to considering several global actions enabled at the same place, by considering a different refinement of s_i taking into account the combined possibilities. If the product system was conflict-equivalent, the a-sites are all equal choice.

But the expression s_i could have been obtained by considering the place p at an arbitrary point in the McNaughton-Yamada construction. Consider e_p as refining some intermediate expression s'_i for the place p. The expression e_p may make copies of parts of s'_i. This does not affect the deterministic global actions property. For $c \neq a$ the c-sites $D^c(e_p)$ are obtained as:

$$D^c(e_p) = \bigcup_{f \in G} D^c(e^T_{p_0,f}) \cup D^c(e^T_{p_0,p}) \cup D^c(e_{p,p}) \cdot e^*_{p,p} \cdot e^Q_{p,f} \cup D^c(e^Q_{p,f}).$$

That is, $Part_c(e_p)$ is preserved as a single block if it formed a single block in the earlier expressions. Thus the expression s_i has the unique sites property. □

Theorem 2. *Let A be a FC-matching product system. Then we can compute a connected expression for the language of A, where every regular expression has a pairing of actions. If the FC-product had deterministic global actions, then so do the regular expressions in the computed expression. If the matching was conflict-equivalent the pairing is equal choice. If all runs of A were consistent with the matching, the expression constructed will be consistent with the pairing.*

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

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

From the lemma we get a connected expression $e' = fsync(s_1, \ldots, s_k)$ for the language of A' over Σ' where every regular expression has unique global action sites. From the proof of the lemma we get for every sequential system A'_i in the product, for the global actions $a^1, \ldots, a^n$, tuples $D'(a^j) = \Pi_{i \in loc(a)} D'_i(a^j)$ which are sites for a^j in the expression s_i, for every j. Now substitute a for every letter $a^1, \ldots, a^n$ in the expression, each tuple D' is isomorphic to a tuple D of sites

for a in e and the sites are disjoint from one another. We let $pairing(a)$ be the partition formed by these tuples. Do the same for b obtaining $pairing(b)$. Repeat this process until all the global actions have been dealt with. The result is an expression e with pairing of actions. If the matching was conflict-equivalent, the pairing has equal choice.

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

$$L = Lang(A) \cap \Sigma^* a \Sigma^* = \bigcup_{R \in pre(a)} Pref_a^R(Lang(A))\ a\ Suf_a^R(Lang(A)).$$

The renaming of transitions depends on the source state, so L is isomorphic to

$$L' = Lang(A') \cap (\sum_j (\Sigma')^* a^j (\Sigma')^*) = \bigcup_{j=1,n} Pref_{a^j}(Lang(A'))a^j Suf_{a^j}(Lang(A')).$$

Keeping Proposition 5 in our hands, the lemma ensures that $Lang(A') = Lang(e')$ and the expression e' has unique a^j-sites forming a block $D'(j)$. Then L' can be written as $\bigcup_{j=1,n} Pref_{a^j}^{D'(j)}(Lang(e'))a^j Suf_{a^j}^{D'(j)}(Lang(e'))$. When we rename the a^j back to a we have a partition of $pairing(a)$ into sets D such that

$$L = \bigcup_{D \subseteq pairing(a)} Pref_a^D(Lang(e))\ a\ Suf_a^D(Lang(e)).$$

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

6 Nets

Definition 12. *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. It will be convenient to define $loc(t) = loc(\lambda(t))$.*

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

Definition 13. *Let $N' = (S \cap X, T \cap X, F \cap (X \times X))$ be a subnet of net $N = (S, T, F)$, generated by a nonempty set X of nodes of N. N' is called a **component** of N if,*

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

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

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

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

Definition 14. *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.

6.1 Properties of Nets

Definition 15 ([DE95]). *Let x be a node of a net N. The* cluster *of x, denoted by $[x]$, is the minimal set of nodes contaning x such that*

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

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

The next definitions will turn out to be the analogue to the separation of labels property of product systems. It is checkable in linear time.

Definition 16. *A labelled net $N = (S, T, F, \lambda)$ is said to have the* ***unique cluster property*** *(briefly,* ***ucp****) if $\forall a \in \Sigma$ having $|loc(a)| > 1$, there exists at most one cluster in which all transitions labelled a occur. It is* ***deterministic for synchronization*** *if for every a, every cluster contains at most one a-labelled transition.*

6.2 Net Systems and their Languages

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

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

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

Definition 17. *For a labelled net system $(N, M_0, \mathcal{G})$, its language is defined as $Lang(N, M_0, \mathcal{G}) = \{\lambda(\sigma) \in \Sigma^* \mid \sigma \in T^* \text{ and } M_0[\sigma\rangle M, \text{ for some } M \in \mathcal{G}\}$.*

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

Definition 18. *An **S-decomposable labelled net system** $(N, M_0, \mathcal{G})$ is an S-decomposable labelled net $N = (S, T, F, \lambda)$ along with an initial marking M_0 and a set of markings $\mathcal{G} \subseteq \wp(S)$, which is a **direct product**: if $\langle q_1, q_2, \ldots q_k\rangle \in \mathcal{G}$ and $\langle q'_1, q'_2, \ldots q'_k\rangle \in \mathcal{G}$ then $\{q_1, q'_1\} \times \{q_2, q'_2\} \times \ldots \times \{q_k, q'_k\} \subseteq \mathcal{G}$.*

6.3 Product Systems to Nets

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 using a standard construction. When we construct nets from product systems with a conflict-equivalent matching of labels with respect to which all runs are consistent, we can refine the construction above to choose $T' \subseteq T$ and get a free choice net.

Theorem 3 ([PL14]). *Let $(N, M_0, \mathcal{G})$ be the net system constructed from product system A above. Then N is an S-decomposable net with $Lang(N, M_0, \mathcal{G}) = Lang(A)$. Further, if A has deterministic global actions and all runs of A are consistent with a conflict-equivalent matching of labels, we can choose $T' \subseteq T$ such that the subnet N' generated by T' is a free choice net with deterministic synchronization and $(N', M_0, \mathcal{G})$ accepts the same language.*

6.4 Nets to Product Systems

Even if a net is 1-bounded and S-decomposable each component need not have only one token in it, but when we say that a 1-bounded net is S-decomposable we assume that each component has one token. For live and 1-bounded free choice nets, such S-covers can be guaranteed [DE95]. Now we can prove:

Theorem 4 ([PL14]). *Let $(N, M_0, \mathcal{G})$ be a live, 1-bounded, S-decomposable labelled free choice net system with deterministic synchronization. Then one can construct a product system A with deterministic global actions, which has a conflict-equivalent matching of labels that all its runs are consistent with. Further $Lang(N, M_0, \mathcal{G}) = Lang(A)$.*

7 Conclusion

In earlier work [LMP11], we showed that a graph-theoretic condition called "structural cyclicity" enables us to extract syntax from a conflict-equivalent product system. In the present work we have generalized this condition so that we can deal with a larger class of product systems with a conflict-equivalent matching. In our paper [PL14] we show a connection between free choice nets with deterministic synchronization and product systems which have these properties along with deterministic global actions. Thus we obtain a Kleene characterization for the class of labelled free choice nets with deterministic synchronization.

Acknowledgements. We would like to thank the referees of the PNSE workshop for urging us to improve the presentation of the proofs of the main theorems. This led us to invent Definition 3 and correct the site properties in Definition 5.

References

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

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

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

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

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

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

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

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

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

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

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

[PL14] Ramchandra Phawade and Kamal Lodaya. Direct product automaton representation of labelled free choice nets. Submitted, 2014.

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

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