Labelled free choice Petri nets, finite Product Automata, and Expressions

By

Phawade Ramchandra Babasaheb

MATH10200704004

The Institute of Mathematical Sciences, Chennai

A thesis submitted to the

Board of Studies in Mathematical Sciences

In partial fulfillment of requirements

For the Degree of

DOCTOR OF PHILOSOPHY

of

HOMI BHABHA NATIONAL INSTITUTE

July, 2014

Homi Bhabha National Institute

Recommendations of the Viva Voce Board

As members of the Viva Voce Board, we certify that we have read the dissertation prepared by Phawade Ramchandra Babasaheb entitled "Labelled free choice Petri nets, Product automata and Expressions" and recommend that it maybe accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

____ Date:

Chair - R. Ramanujam

____ Date:

Guide/Convener - Kamal Lodaya

____ Date:

Member 1 - Madhavan Mukund

____ Date:

Member 2 (External Examiner) - Paritosh Pandya

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to HBNI.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it may be accepted as fulfilling the dissertation requirement.

Date:

Place: Guide

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at Homi Bhabha National Institute (HBNI) and is deposited in the Library to be made available to borrowers under rules of the HBNI.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the Competent Authority of HBNI when in his or her judgement the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

(Phawade Ramchandra Babasaheb)

DECLARATION

I, hereby declare that the investigation presented in the thesis has been carried out by me. The work is original and has not been submitted earlier as a whole or in part for a degree / diploma at this or any other Institution / University.

(Phawade Ramchandra Babasaheb)

List of publications arising from the thesis:

- **Published in Journals**

  1. Ramchandra Phawade, *Direct product representation of labelled free choice nets*, in *International Journal of Computer Applications*, 99(16):1–8, 2014.
  2. Ramchandra Phawade and Kamal Lodaya, *Kleene theorems for synchronous products with matching*, in *Transactions on Petri nets and other models of concurrency X*, 2015(To appear).

- **Other refereed publications**

  1. Kamal Lodaya, Madhavan Mukund and Ramchandra Phawade, *Kleene theorems for product systems*, in *13th Descriptional complexity of formal systems* workshop (Markus Holzer, Martin Kutrib and Giovanni Pighizzini, eds.), Lecture Notes in Computer Science, vol. 6808, Springer, 2011, pp. 235–247.
  2. Ramchandra Phawade and Kamal Lodaya, *Kleene theorems for labelled free choice nets*, in *8th Petri nets and software enginering* workshop (Daniel Moldt and Heiko Rölke, eds.), CEUR-WS, vol. 1160, Aachen, 2014, pp. 75–89.

(Phawade Ramchandra Babasaheb)

ACKNOWLEDGEMENTS

I am sincerely grateful to Kamal Lodaya–my thesis advisor for providing his guidence and expertise. He walked me through the nitty gritties of research. He has been very kind and patient with me. I am indebted to R. Ramanujam(Jam) and Madhavan Mukund for supporting and providing me help whenever I was driven to despair and needed advice. Without them this thesis would not have been possible.

I am grateful to IMSc and CMI for providing excellent facilities to do research. I am thankful to computer science faculty from IMSc and CMI for creating excellent environment to work and exchange ideas. I am thankful to S. Akshay from IIT Bombay whose ready offer to visit IIT Bombay (which never realized before submitting the thesis to HBNI), has been a catalyst in winding up things faster. I am thankful to Sheeraz, Sreejith, Baskar, Yadu, Joydeep, Raja, Neha, Ramanathan and Anup for all the technical discussions we had.

Over all these years I have been part of various activities in my PhD life, and have gained lot of friends in Chennai, supporting me through all the ups and downs. I am grateful to Amal, Soumya, Sundar, Brenda(Bruno), Kamil Bhai for helping me out in my initial phase in IMSc. I am thankful to Brenda, Arghya, Dude, Navin, Issan and Raja for sharing movies. I would like to thank my friends Raja and Joydeep with whom I had a pleasant stay in my office, and had discussions which most of the times ended in walks in CPT campus! I am thankful to Sudhir, Neeraj, Sheeraz, Hassan and Dhriti –my companions in exciting long walks to Kovalam which were always filled to the brim with discussions on politics, poetry, philosophy. I am thankful to Baskar, Karthick(Dude), Sreejith, Yadu, Mubeena, Neeraj along with Sudhir for (often late night) games of badminton and Table Tennis. I am thankful to ever enthusiastic Baskar who never said no for accompanying me to swimming, jogging, cycling or playing badminton. I am thankful to Rusa and Rajesh, for accommodating me on the breakfast table during the last year of my stay in IMSc and,

also to Vinu Luckose for his timely advise. I thank my friends Rajarshi and Debakanta from MSc days who always encouraged me and I am also grateful to Rajeshwari, Shiva Shankar and KV Subramaniam from CMI for expressing concern whenever they met me.

I am indebted to Sheeraz for his advice, support and encouragement for last 16 years. Thank you Alok, Madhushree, Gopal, Neha, Rohan for being what you are. It would have been easy to lose my sanity without you people around. Thanks Butch(Motu) for all the unconditional love, you taught me to care and made sure that I would never run out of friends in life. I am thankful to the innoncent Kiddo, Gundaa, healthy Whitey, Mumma dog, playful Deer, daring Stripey, raider Youth, selfish Automaton, beautiful Helen, senti Music and survivor Backjump among all other inhabitants of 4th cross street. Their presence filled life with fun.

Outside IMSc I would like to thank my old friends Abhiram Singh, M. Parthasarathy, A. Senthil, Saurabh, Rashmi as well as friends from undergrad days Sachin Nair, Ajay Balachandran and Sangmeshwar Kadole . They stood close to me, despite the distance.

Finally I would like to thank my parents, my brother and sister for their patience and understanding; they have been victims of my indecisiveness all the time. They supported me in whatever path I chose. My deepest gratitude is to my grandparents and my first teacher Ali Sahib Sheikh. This thesis is dedicated to them.

Contents

Synopsis

Petri nets are a formal model of concurrent systems. They were first defined by Petri in his thesis [Pet62] and were presented at the IFIP 1962 congress in Munich [Pet63]. Nets are widely used in modelling various aspects of distributed systems.

There are several different notions of acceptance to define languages for labelled Petri nets [Pet76, Hac76, Gra81, GR92] with general markings, depending on restrictions on labelling and "final" markings. Some of these are studied by Peterson [Pet76] and in a survey article by Jantzen [Jan86]. Grabowski [Gra81] defined expressions matching the regular languages accepted by labelled 1-bounded nets. Mazurkiewicz [Maz77] considers P-type languages of 1-bounded nets [Pet76, Jan86], labelled with a concurrent alphabet. Ochmański [Och85] defines c-rational expressions and sets up a correspondence between them and regular trace languages.

An algebraic characterization in terms of recognition by finite partially commutative monoids is also discussed by Mazurkiewicz [Maz86] and described in this book. However this has not been used so far for characterizing subclasses of nets as has been done in automata theory.

In this thesis, our focus is on producing expressions which exactly describe various subclasses of trace-labelled 1-bounded free choice nets, equipped with an initial marking and a set of final markings, so that the L-type Mazurkiewicz trace languages of both formalisms are the same. To get these expressions for nets, we use product systems in between. For that we explore the question of direct product representation of labelled 1-bounded nets.

List of Figures

Chapter 1

Introduction

Petri nets are a formal model of concurrent systems. They were first defined by Petri in his thesis [Pet62] and were presented at the IFIP 1962 congress in Munich [Pet63]. Nets are widely used in modelling various aspects of distributed systems. See the books by Peterson [Pet81] and Reisig [Rei85], and the survey articles by Murata [Mur89] and Yen [Yen06]. A number of fundamental articles on Petri nets are found in the Advanced Courses on Petri nets held in Bad Honnef [BRR87], Dagstuhl [RR98] and Rostock [JvdAB$^+$13].

The class of Petri net languages when nets are unlabelled is very restricted, as they do not include even all regular languages. Consider a regular language $\{a, aa\}$, it can not be represented by any unlabelled Petri net. In this thesis we will only consider 1-bounded labelled Petri nets. In particular we consider subclasses of free choice nets. Free choice nets have a pleasant theory, see [BS83, TV84, BV84] and the book by Desel and Esparza [DE95]. From verification point of view, these nets have some advantages: for 1-bounded free choice nets reachability problem is PSPACE-complete [CEP95], checking liveness is in PTIME [ES92, Des92], and checking deadlock is NP-complete [CEP95]. For 1-bounded nets, all these problems are PSPACE-complete [CEP95].

Expressions are widely used by programmers to describe languages of software components. For finite state machines we have regular expressions given by Kleene [Kle56]. Regular expressions are at the heart of programs written in Perl, Python, Tcl. Regular expression libraries are written for many other languages [Fri02]. They are thought of as a user-friendly alternative to the finite state automata for describing software components [HMU03]. So correspondence between these two formalisms, machines and expressions, is desirable.

There are various Petri net based tools like CPNTools [cpn] and PEP [pep] for modelling and analysis. It may be useful to have expressions to describe Petri nets. On the other hand expressions can be used for axiomatizing language equivalence.

There are several different notions of acceptance to define languages for labelled Petri nets [Pet76, Hac76, Gra81, GR92] with general markings, depending on restrictions on labelling and "final" markings. Some of these are studied by Peterson [Pet76] and in a survey article by Jantzen [Jan86].

Jantzen's L-type languages [Pet76, Jan86] are defined for a net with an initial marking and a finite set of final markings. In this thesis we will mostly use such a definition $(N, M_0, \mathcal{G})$. Jantzen also defines P-type languages as those where any reachable marking is a final marking. These languages will be closed under taking prefixes of words.

Grabowski [Gra81] defined expressions for L-type languages accepted by 1-bounded nets. Mazurkiewicz [Maz77] considered P-type languages of 1-bounded nets [Pet76, Jan86]. Since concurrent transitions can be fired in any order, he defined a concurrent alphabet (Σ, I) where I is a binary independence relation between letters of Σ, and languages closed under this relation, that is, if $wabz$ is in the language and a and b are independent, then $wbaz$ is in the language as well. These are also called Mazurkiewicz trace languages.

The regular trace languages are those accepted by 1-bounded nets where the independence relation can be defined between transitions which have disjoint neighbourhoods.

For a labelled net this requires that the net be trace-labelled so that concurrency in its behaviour is always between independent actions. This condition was defined by Thiagarajan [Thi02]. Ochmański [Och85] defines c-rational expressions and sets up a correspondence between them and regular trace languages.

The book by Diekert and Rozenberg [DR95] describes more of this foundational research on trace languages. An algebraic characterization in terms of recognition by finite partially commutative monoids is also discussed by Mazurkiewicz [Maz86] and described in this book [DR95]. However this has not been used so far for characterizing subclasses of nets as has been done in automata theory.

In this thesis, our focus is on producing expressions which exactly describe various subclasses of trace-labelled 1-bounded free choice nets, equipped with an initial marking and a set of final markings, so that the L-type Mazurkiewicz trace languages of both formalisms are the same. We work with a more structured alphabet (distributed alphabet) rather than Mazurkiewicz's alphabet with independence relation (concurrent alphabet).

To obtain the two way translations between various subclasses of nets and expressions, we use product systems in between. This way we get correspondence between nets, product systems and expressions.

Lodaya, Ranganayukulu and Rangarajan [LRR03, Lod06] defined and studied other subclasses of S-nets [DE95] and SR-nets, for which corresponding series-rational expressions [LW00] are given. S-nets and T-nets are orthogonal to each other, although both are subclasses of free choice nets. SR-nets are orthogonal to free choice nets. The syntactic characterization of subclasses of free choice nets and T-nets reported in this thesis is new.

1.1 Preliminaries and Notations

We start with some preliminaries and fix notations which will be used in this thesis.

Let $\mathbb{N}$ denote the set of natural numbers. 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 $L \subseteq \Sigma^*$.

For a word w and $a \in \Sigma$, $|w|_a$ denotes the number of occurrences of the letter a that appear in w. The alphabet of a word w is $\alpha(w) = \{a \in A \mid |w|_a > 0\}$.

The projection of a word $w \in \Sigma^*$ to a set $\Delta \subseteq \Sigma$, denoted as $w{\downarrow_\Delta}$, is defined by: $\epsilon{\downarrow_\Delta} = \epsilon$ 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 a finite set* $\{1, 2, \ldots, k\}$. *A* ***distributed alphabet*** *over Loc is a tuple of nonempty sets* $\Sigma = (\Sigma_1, \Sigma_2, \ldots, \Sigma_k)$. *We also write* Σ *for* $\bigcup_{1 \le i \le k} \Sigma_i$. *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 distributed alphabet induces an independence relation on letters of Σ: letters a and b are in the independence relation I if and only if $loc(a)$ and $loc(b)$ are disjoint. This induces a trace equivalence $\sim$ on Σ^*: $w \sim w'$ iff w can be obtained from w' by a sequence of permutations of adjacent independent letters. For example, if aIb then $uabv \sim ubav$. A **Mazurkiewicz trace** [DR95] over (Σ, I) is an equivalence class of words with respect to $\sim$. Let $[w]$ denote a trace of word w. For a language L let $[L]$ denote its trace closure, defined as $[L] = \{[w] \mid w \in L\}$.

Definition 2. *The* ***shuffle*** *of two words, denoted by* $u\|v$ *is,*
$u\|v = \{w \mid w = w_1 w_2 \ldots w_{2n-1} w_{2n}$ *such that* $w_1 w_3 \ldots w_{2n-1} = u$ *and* $w_2 w_4 \ldots w_{2n} = v\}$.
Shuffle of two languages L_1 *and* L_2 *is defined as:*

$$L_1 \parallel L_2 = \bigcup \{w_1 \parallel w_2 \mid w_1 \in Lang(e_1),\ w_2 \in Lang(e_2)\}.$$

Definition 3. *The* ***synchronized shuffle*** *of k words* $w_1, w_2, \ldots, w_k$ *defined over* $\Sigma_1, \ldots, \Sigma_k$ *respectively, is* $sync(w_1, w_2, \ldots, w_k) = \{w \mid w{\downarrow}\Sigma_i = w_i\}$, *for all* $i \in \{1, 2, \ldots, k\}$. *The* *synchronized shuffle of k languages $L_1, L_2, \ldots, L_k$ defined over $\Sigma_1, \ldots, \Sigma_k$ respectively, is $sync(L_1, L_2, \ldots, L_k) = \{w \mid w{\downarrow}\Sigma_i \in L_i\}$,for all $i \in \{1, 2, \ldots, k\}$.*

1.2 Nets

Fix a distribution $(\Sigma_1, \Sigma_2, \ldots, \Sigma_k)$ of Σ. Labelled nets are defined over this alphabet.

Definition 4. *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 consider only those nets in which every transition has nonempty pre-set and post-set.

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. 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 $t \in T \cap X$, we have $|{}^\bullet t| = 1 = |t^\bullet|$ (N' is an S-net [DE95]),*
- *Under the flow relation, N' is connected.*

A set C of components of net N is called ***S-cover*** *for N, if every place of the net belongs to some component of 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.

1.2.1 Properties of Nets

Definition 6. *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 denoted by tuple (S_C, T_C), where S_C is the set of places and T_C is the set of transitions of C. 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 set $\{[x] \mid x \text{ is a node of } N\}$ is a partition of the nodes of N.

A further restriction gives the subclass of **T-nets**, or marked graphs [CHEP71, Hac72, DE95]. They are nets allowing communication but no choice.

Definition 7. *A net is called* T-net *if for each transition t in it $|^\bullet t| = |t^\bullet| = 1$.*

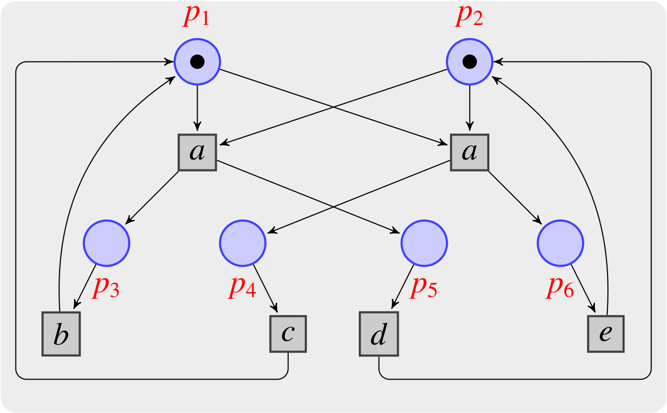


Figure 1.1: Live and 1-bounded, labelled free choice net

Example 8. *Consider the net shown in Figure 1.1. We can think of the token in place p_1 as a server for booking airline tickets and the token in place p_2 as a client. We have following transition labels in the net system:* a *is the action of booking airline ticket;*

b–breakfast provided by airline; c–breakfast not provided by airline; e–client carries breakfast; d–client does not carry breakfast. Firing sequence abd or adb means that the airline booking provides breakfast and and the client does not carry breakfast. On the other hand firing sequence ace or aec means that the airline does not provide breakfast and the client carries it.

Places p_1, p_2 along with two transitions labelled a form a cluster. Since all clusters in it are free choice net is a free choice net. On the other hand the net shown in Figure 1.3 is not free choice because the cluster of transition labelled c is not a free choice cluster.

Here free choice is a structural condition on the net. It distinguishes the internal and external choice operators used in process algebra [Mil80, Hoa85]. Conflict between synchronizations is an external choice, conflict between local actions is an internal choice, and there cannot be a conflict between synchronizations and local actions.

1.2.2 Net Systems and their Languages

A **net system** (N, M_0) is a labelled net with an initial marking. Sometimes we add a set of final markings to get the triple $(N, M_0, \mathcal{G})$, which we also call a net system. In this thesis we are only interested in 1-bounded (or condition/event) net systems, 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 at 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 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. A net system (N, M_0) is said to have deadlock, if at some reachable marking M no transition is enabled, if no such reachable marking exists then it is called deadlock-free.

An acyclic net system (N, M_0) is said to have **active deadlock** at some marking M, if no transition is enabled at M, and there exist at least one transition t such that ${}^{\bullet}t \cap M \neq \emptyset$.

Definition 9. *The language of a labelled net system $(N, M_0, \mathcal{G})$ is defined as $Lang(N, M_0, \mathcal{G}) = \{\lambda(\sigma) \in \Sigma^* \mid \sigma \in T^* \text{ and } M_0[\sigma\rangle M, \text{ for some } M \in \mathcal{G}\}$. For a net system (N, M_0) we assume every marking is final, hence its language $Lang(N, M_0)$ is prefix-closed.*

1.3 Product Systems over a Distribution

Product systems are used to obtain the two way translations between various subclasses of nets and expressions. We describe this model formally in this section.

Given a regular trace language over (Σ, I) presented as a monoid, Zielonka [Zie87] gave an alternate presentation called asynchronous automata. They are also called Zielonka automata over a distributed alphabet Σ. This is a distributed implementation of a regular trace language where component automata (which are usual sequential finite automata) run on the respective component alphabets. Other proofs of Zielonka's theorem are given in the trace book [DR95] and by Mukund and Sohoni [MS97].

In this thesis we work with simpler and less powerful distributed implementations of regular trace languages, called **product systems**. These are synchronized products of sequential systems as studied by Arnold [Arn98]. Different types of acceptance conditions turn them into **product automata**. Formally we use tuples of sequential component automata $A = (A_1, A_2, \ldots, A_k)$ defined over distributed alphabet $\Sigma = (\Sigma_1, \ldots, \Sigma_k)$, where A_i is over alphabet Σ_i. Product systems were introduced by Thiagarajan in the context of verification [Thi95, CMT99], product automata were studied by Mohalik and Ramanujam [MR97]. Mohalik's thesis [Moh98] differentiates between direct products and their boolean closure, which are called synchronized direct products. Some open questions are discussed in [Muk02]. Computational complexity issues are discussed in [GM06]. Mukund's survey [Muk11] describes the subtleties involved as the complexity rises from direct product automata to Zielonka automata.

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

Definition 10. *A sequential system over a set of actions Σ_i is a tuple $A_i = (P_i, \rightarrow_i, p_i^0, G_i)$ 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****. $\overset{a}{\rightarrow}_i$ is the set of local a-moves.*

A local move $\langle p, a, p' \rangle$ is said to be outgoing for place p and incoming for place p'.

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 of sequential system A_i is defined as $Lang(A_i) = \{w \in {\Sigma_i}^* | w \text{ is accepted by } A_i\}$.

Definition 11. *Let $A_i = (P_i, \rightarrow_i, p_i^0, G_i)$ 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 $(A_1, \ldots, A_k)$.*

Let $\Pi_{i \in Loc} P_i$ be the set of product states of A. We use $R[i]$ for the projection of a product state R in A_i, and $R \downarrow I$ for the projection to $I \subseteq Loc$.

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

Now we define a property which correspond to free choice property of nets.

Definition 12 (conflict-equivalent moves, states)**.** *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. In a product system, we say that a local state $p \in P_i$ is **conflict-equivalent** to a local state $p' \in P_j$, if for some action $a \in \Sigma$, p have an outgoing local move on a i.e., $\exists \langle p, a, q_1 \rangle \in \rightarrow_i$ implies $\exists \langle p', a, q'_1 \rangle \in \rightarrow_j$, and, conversely, moves from p' are matched by moves from p.*

Example 13. *In the product system shown in Figure 1.2 we have two b-moves which are not conflict-equivalent so the product system is not conflict-equivalent.*

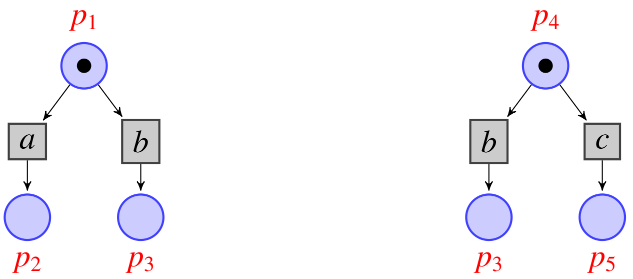


Figure 1.2: Non conflict equivalent Product system

1.3.1 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, $R = pre(a)$ for some a-labelled global move g,

Q is reached after va where, for all $j \in loc(a), g[j] = \langle R[j], a, Q[j] \rangle \in \rightarrow_j$ and for all $j \notin loc(a), R[j] = Q[j]$. We will call g a reachable global move.

A product system A is said to have a deadlock, if at some reachable global state R no global move is enabled. Otherwise A is said to be deadlock-free.

An acyclic product system A is said to have an active deadlock at some reachable state R if no transition is enabled at R, and there exist at least one global move g such that pre-places$(g) \cap R \neq \emptyset$.

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 14. *$L = Lang(A)$ is the language of product system $A = (A_1, \ldots, A_k)$ over distribution Σ iff $L = \{w \in \Sigma^* \mid$ for all $i \in \{1, \ldots, k\}$, there exists $u_i \in L$ such that $w\downarrow_{\Sigma_i} = u_i\downarrow_{\Sigma_i}\}$. Further $L = sync(Lang(A_1), \ldots,\ Lang(A_k))$.*

1.4 S-decomposability and Direct Product Representation

In order to write expressions for languages of net systems, we use language equivalent product systems as an intermediate formalism. Again in the reverse direction, starting with expressions we use product systems as an intermediate formalism while going to equivalent net systems. So product systems are central to our characterization of nets.

Definition 15 (Direct product representation)**.** *A labelled* 1*-bounded net system is said to be direct product representable when there exists a language equivalent product system for it.*

Direct product representation of languages of labelled 1-bounded net systems has been characterized in [CMT99]. Starting with a net system, their algorithm is doubly exponential in the size of the net, as one exponential is required to construct an asynchronous transition system, from which their algorithm starts. We find a direct product for a net if we are given one of its S-covers.

Definition 16. *A labelled net $N = (S, T, F, \lambda)$ is called* ***S-decomposable*** *if, there exists an S-cover 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 C.*

Proposition 17. *If labelled net N is S-decomposable then for all a-labelled transitions $t \in T$, $|loc(a)| \geq max(|{}^\bullet t|, |t^\bullet|)$.*

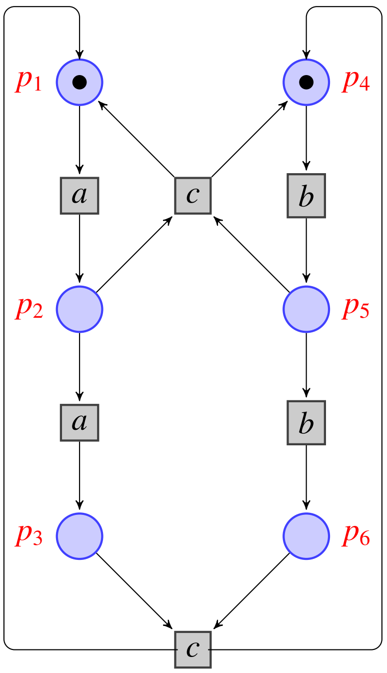
Proof. Consider a transition t labelled a with $|{}^\bullet t| > 1$. Let $\{p, q\} \subseteq {}^\bullet t$. Since N is S-decomposable, we have an S-cover for N. So there exist components $N_i = (S_i, T_i, F_i)$ and $N_j = (S_j, T_j, F_j)$ such that $p \in S_i$ and $q \in S_j$. If $i = j$ then p and q will belong to same component and by definition of S-cover transition t will also belong to it, which cannot be the case as components are S-nets, and a transition can not have multiple pre-places in an S-net. Therefore i and j are distinct and transition $t \in T_i \cap T_j$. Hence $|loc(a)| \geq |{}^\bullet t|$. Similarly we show $|loc(a)| \geq |t^\bullet|$. □

Now from S-decomposability we get 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, subnet (S_i, T_i, F_i) generated by S_i and T_i is an S-net, where F_i is an 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 $S_1 \times S_2 \times \ldots \times S_k$. However, this is not sufficient for direct product representability. Figure 1.3 gives a counterexample due to Zielonka [Zie87]. Zielonka's net is not free choice, in Figure 1.1 we give a free choice counterexample.

Example 18. *Consider the net in Figure 1.1. Now we look at the individual server and client processes as shown in Figure 1.4. Now we can see it as two processes communicating together, each one nondeterministically choosing the airline booking. Here all the*



Distributed alphabet is $\Sigma = (\Sigma_1 = \{a, c\}, \Sigma_2 = \{b, c\})$. and hence the independence relation is $I = \{(a, b)\}$. $Lang(N, M_0 = M_f = \{p_1, p_4\}) = [((ab + aabb)c)^*]_I$. This language is not definable by Direct Product Systems.

Figure 1.3: Zielonka net: S-decomposable but not Direct Product representable net

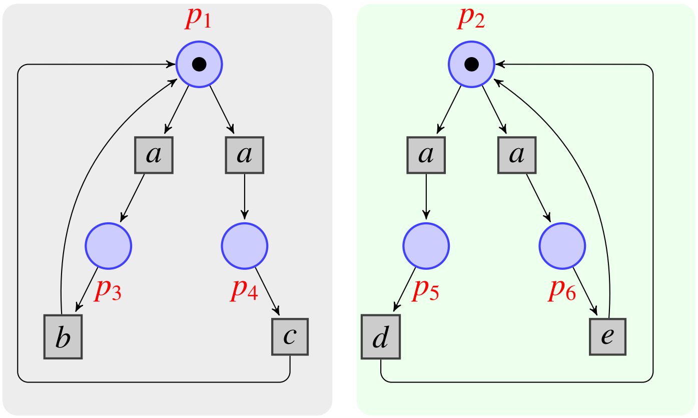


Figure 1.4: Direct Product for net in Figure 1.1

sequences of actions given by the net are also possible. Apart from that some unpleasant sequence can also happen. For example we can fire a sequence ace *which means that airline does not provide breakfast and the client also does not carry it.*

Theorem 19. *There is a live and 1-bounded labelled free choice net system which is not direct product representable.*

Proof. Figure 1.1 gives the net. For final marking $\{1,2\}$, the language accepted by live and 1-bounded labelled (extended) free choice net shown in Figure 1.1, is

$L = \{abd, adb, ace, aec\}^*$ over the distribution $\Sigma = (\Sigma_1 = \{a,b,c\}, \Sigma_2 = \{a,d,e\})$. It is a Mazurkiewicz trace language.

Let $w = abeacd$ for which $w{\downarrow_{\Sigma_1}} = abac$ and $w{\downarrow_{\Sigma_2}} = aead$. Now consider $u_1 = abdace$ for which $u_1{\downarrow_{\Sigma_1}} = abac$ and $u_2 = aceabd$ for which $u_2{\downarrow_{\Sigma_2}} = aead$. Since both $u_1, u_2 \in L$ using characterization given in Proposition 14 we get $w \in L$, which is a contradiction. □

In Section 4.3, we will identify a couple of sufficient conditions for decomposition into product automata.

1.5 Constructing Nets from Product Systems

Definition 20 (Net construction)**.** *Given a product system $A = (A_1, A_2, \ldots, A_k)$ over distribution Σ, there is a generic construction of a net system $(N = (S, T, F, \lambda), M_0, \mathcal{G})$ as follows:*

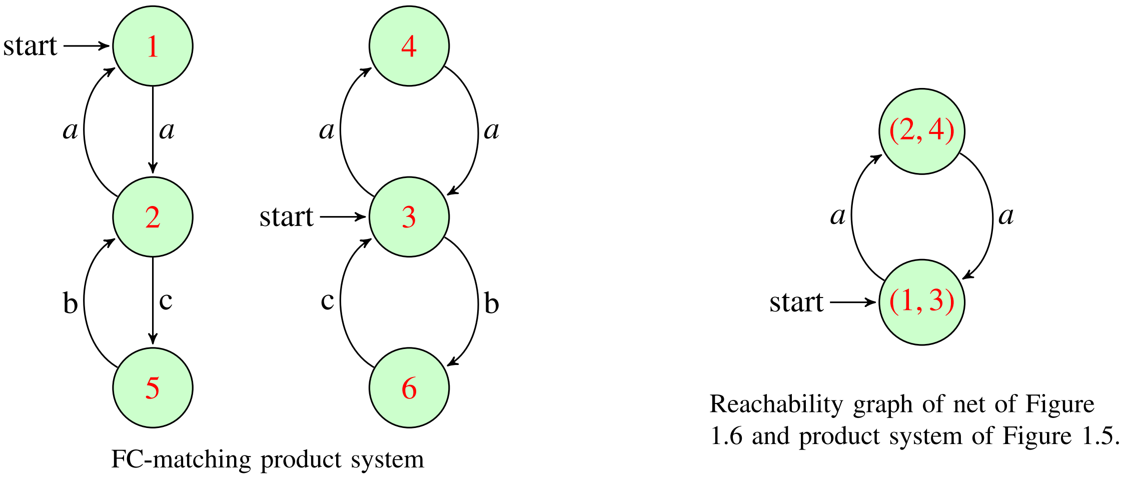
- *$S = \cup_i P_i$, the set of places.*
- *$T = \cup_a T_a$, where T_a is $\Rightarrow_a$, the set of a-labelled global moves.*
- *The labelling function λ labels by a by the transitions in T_a.*
- *The flow relation $F = \{(p,g),(g,q) \mid g \in T_a, g[i] = \langle p,a,q \rangle, i \in loc(a)\}$.*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

Since a global action a can be in every component A_i of the product system and there can be an arbitrary number n_i of a-labelled choices in each component, the resulting a-cluster in the net has $n_1 \times \cdots \times n_k$ transitions which can be exponential in the size of the product system. If the product system was deterministic for global actions, then the constructed net is polynomial in size. If the final product states were a direct product $G_1 \times \cdots \times G_k$, the final markings will also be direct product.

Unfortunately, this construction fails to produce a free choice net because of its profligacy.

Consider the product system shown in Figure 1.5. The distributed alphabet is $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, b, c\})$. The "reachability graph" of the product system shown in the same figure shows that only two global moves are reachable.

The net shown in Figure 1.6 is the result of the product construction. It is language equivalent to the given product system, 1-bounded but not free choice. Although the net system is deadlock-free, some transitions are never fired in it.



FC-matching product system

Reachability graph of net of Figure 1.6 and product system of Figure 1.5.

Figure 1.5: Deadlock-free Product System and its Reachability Graph

In Section 4.4 we will identify sufficient conditions for constructing labelled free choice nets from product automata.

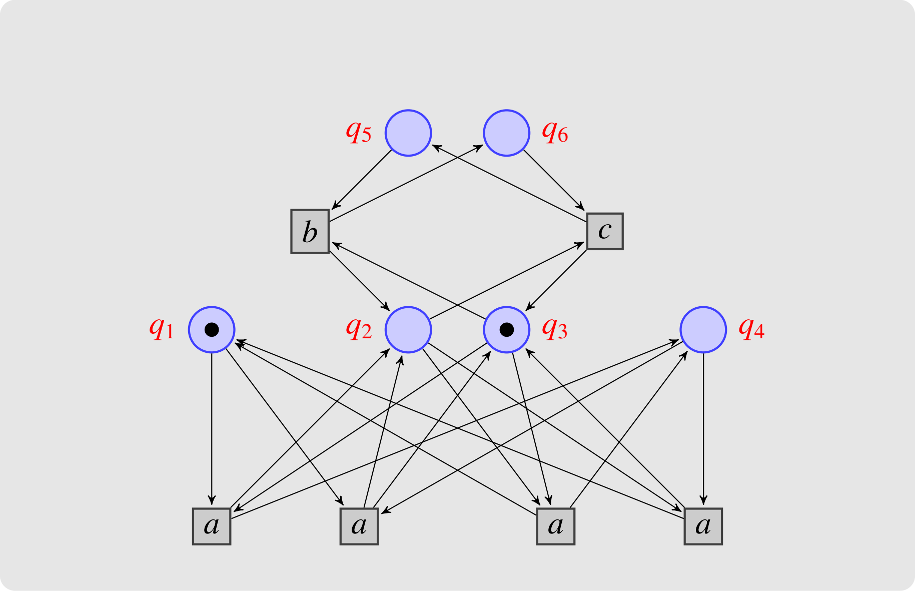


Figure 1.6: Deadlock-free Net obtained from Product System of Figure 1.5

1.6 Thesis organization

In this introductory chapter we have considered classes of labelled, 1-bounded and S-decomposable Petri nets, which satisfy distributed choice property (DCP). With given initial marking and set of final markings, we consider its languages as set of labelled sequences which starting in initial marking lead to one of the final marking. For various subclasses of such nets we have given corresponding product systems and expressions, along with two way conversions from nets to product systems and product systems to nets. One contribution of this chapter is a live and 1-bounded labelled free choice net which is not direct product representable. This last part is published in [Pha14].

In the Second chapter, we give definitions and explanations mainly required for the syntax which we shall be considering in this thesis. We also recall some earlier work. One contribution of this chapter is a syntactically-defined partitioning of the derivatives and a result which relates the partitioning to an abstract conception of state. This last part has been mentioned without detailed proofs in the article [PL14].

In the Third chapter, we consider 1-bounded labelled free choice nets. We define T-dags, FC-dags, T-product systems and structurally cyclic FC-product systems. We give constructions relating them to acyclic T-net systems, acyclic free choice net systems and live T-net systems. We also define corresponding classes of expressions. This last work which establishes correspondence between expressions and product systems has been published in the article [LMP11].

In the Fourth chapter, we consider live and 1-bounded labelled free choice nets, both with a unique cluster property and without it. We give corresponding class of product systems from which these nets are constructed. This is published in [Pha14].

In the Fifth chapter, we establish the correspondence between product systems with separation of labels property and product expressions with unique global actions. Also we establish the correspondence between product systems with conflict-equivalent matching, satisfying consistency of matching and the product expressions with equal-choice pairing, and satisfying consistency with pairing. A large part of these results has been published in [PL14] and [PL15].

In the Sixth chapter, we consider a class of labelled 1-bounded nets which strictly include labelled 1-bounded free choice nets. This is a subclass of 1-bounded nets, obtained as a result of net construction from direct products. It seems intuitive that these nets should be decomposable into sequential machines. In this chapter, we give some counterexamples to our attempts to characterize these nets. We also examine the role of S-decomposability.

In the Seventh chapter, we conclude our thesis.

Chapter 2

Expressions and Partitions of Derivatives

In Chapter 1 we saw Petri net systems and product systems defined over distributed alphabets. In this chapter we give syntax and semantics of expressions which correspond to them.

For describing various properties of expressions we use derivatives of expressions. We review concepts of derivatives of regular expressions and introduce some new concepts like partitions of derivatives and derivatives of connected expressions.

In this section, we describe properties of regular expressions.

2.1 Regular Expressions

First we define some subclasses of regular expressions over the alphabet Σ_i:

Word over Σ_i	$w ::= a \in \Sigma_i \mid w_1 w_2$
Sum over Σ_i	$s ::= a \in \Sigma_i \mid s_1 s_2 \mid s_1 + s_2$
Regular expression over Σ_i	$r ::= a \in \Sigma_i \mid r_1 r_2 \mid r_1 + r_2 \mid r^*$

The language of constant 0 is $\emptyset$ and that of 1 is $\{\epsilon\}$. 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 of the component languages respectively. As a measure of the size of an expression we will use its length $|r|$ and also for its alphabetic width $wd(r)$—the total number of occurrences of letters of Σ in r.

Definition 21 (Initial actions of a regular expression)**.** *The set of initial actions of a regular expression r is* $Init(r) = \{a \mid aw \in Lang(r) \text{ and } w \in \Sigma_i^*\}$.

$Init(r)$ can be defined syntactically by induction, as given below.

$$\begin{aligned}
Init(a) &= \{a\} \\
Init(s_1^*) &= Init(s_1) \\
Init(s_1 + s_2) &= Init(s_1) \cup Init(s_2) \\
Init(s_1 \cdot s_2) &= \begin{cases} Init(s_1) \cdot s_2 \cup Init(s_2) & \text{if } \epsilon \in Lang(s_1) \\ Init(s_1) & \text{otherwise} \end{cases}
\end{aligned}$$

We can syntactically check whether the empty word $\epsilon \in Lang(s)$ as shown below.

$$\begin{aligned}
EmptyWord(\epsilon) &= \text{TRUE} \\
EmptyWord(a) &= \text{FALSE} \\
EmptyWord(s_1^*) &= \text{TRUE} \\
EmptyWord(s_1 + s_2) &= EmptyWord(s_1) \text{ OR } EmptyWord(s_2) \\
EmptyWord(s_1 \cdot s_2) &= EmptyWord(s_1) \text{ AND } EmptyWord(s_2)
\end{aligned}$$

Now we give overview of derivatives of regular expressions, using which various properties are defined later.

2.1.1 Derivatives of Regular Expressions

Derivative of a regular expression r with respect to an action a is one or more regular expressions describing a set of words w such that $aw \in Lang(r)$. Derivatives were first introduced by Brzozowski [Brz64]. Using derivatives he gave a construction to get deterministic finite state automaton from a given regular expression, which could be exponential in the size of regular expression. Later Mirkin [Mir66] and Antimirov [Ant96] modified this notion to define partial derivatives, which could be used to construct non-deterministic finite state machines with number of states linear in the the size of regular expression. Sakarovitch's book has a modern exposition [Sak09]. Below we inductively define Antimirov derivatives [Ant96].

Definition 22. *Given regular expression s and symbol a, the set of partial* derivatives *of s wrt a, written* $\mathsf{Der}_a(s)$ *are defined as follows.*

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

Inductively $\mathsf{Der}_{aw}(s) = Der_w(Der_a(s))$.

The set of all partial derivatives $\mathsf{Der}(s) = \bigcup_{w \in \Sigma_i^*} \mathsf{Der}_w(s)$, *where* $\mathsf{Der}_\epsilon(s) = \{s\}$. *For a given set R of regular expressions its derivative with respect to some letter a is the union of derivatives of individual regular expressions with respect to letter a i.e.,* $\mathsf{Der}_a(R) = \bigcup_{rinR} \mathsf{Der}_a(r)$. *A derivative d of s with global* $a \in \mathsf{Init}(d)$ *is called an* ***a-site*** *of s. Expression s is said to have* ***equal choice*** *if for all a, all its a-sites have the same set of initial actions.*

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

For words, $Der_{aw}(E)$ is defined to be $Der_a(Der_w(E))$, with $Der_\epsilon(E) = E$.

Example 23. *For example,* $\mathsf{Der}_a(ab + ac) = \{b, c\}$, *while* $\mathsf{Der}_a(a(b + c)) = \{b + c\}$.

Example 24. *For regular expression $E = x^*(xx + y)^*$, we compute its partial derivatives as:*

- $Der_\epsilon(E) = \{x^*(xx + y)^*\}$
- $Der_x(E) = \{x^*(xx + y)^*,\ x(xx + y)^*\}$
- $Der_y(E) = \{(xx + y)^*\}$
- $Der_x(x(xx + y)^*) = \{(xx + y)^*\}$
- $Der_y(x(xx + y)^*) = \emptyset$
- $Der_x((xx + y)^*) = \{x(xx + y)^*\}$
- $Der_y((xx + y)^*) = \{(xx + y)^*\}$

From partial derivatives, using derivatives as states, an ϵ-free NFA can be constructed.

Theorem 25 ([Ant96]). *Let $\mathrm{Der}(E) = \{d \mid d \in \mathrm{Der}_w(E)\ \text{ and }\ w \in \Sigma^*\}$, denote the set of all partial derivatives of the regular expression E. The cardinality of the set $\mathrm{Der}(E)$ of a regular expression E is less than or equal to $wd(E) + 1$.*

The equation automaton of regular expression E, $\mathcal{E}_E = (Q, \Sigma, i, T, \delta)$, is defined by:

- $Q = \mathrm{Der}(E)$,
- $i = E$,
- $T = \{p \mid \epsilon \in Lang(p)\}$,
- $\delta(p, a) = \mathrm{Der}_a(p)$, *for all* $p \in Q$ *and for all* $a \in \Sigma$.

For a regular expression E, its equation automaton is constructed in time $O(wd(E)^3 \cdot |E|^2)$ in worst case, and it has $wd(E) + 1$ number of states, and $O(wd(E) + 1)^2 \cdot \Sigma)$ number of transitions, and no ϵ-transitions.

For a sum s, the automaton is acyclic.

For a word w, the automaton consists of a single path.

Equation automaton for expression $E = x^*(xx + y)^*$ is given in Figure 2.1.

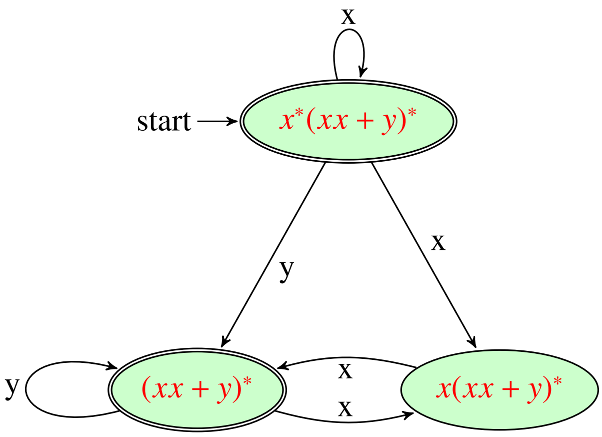


Figure 2.1: Equation automaton of $E = x^*(xx + y)^*$

From Antimirov's construction following remark is immediate.

Remark. If A_i is an automaton constructed from regular expression r_i then $\langle p, a, q\rangle \in\rightarrow_i$ iff $r_q \in \mathrm{Der}_a(r_p)$ where p, q are reachable states in P_i corresponding to regular expressions r_p, r_q in $\mathrm{Der}(r_i)$.

Now we define some properties using Antimirov derivatives.

A derivative d of s with global $a \in Init(d)$ is called an a-**site** of s.

Regular expressions s and s' are said to have **equal choice** (or be in equal choice) if they have same set of initial actions i.e., $Init(s) = Init(s')$. For a set D of derivatives, we collect all initial actions to form $Init(D)$.

Example 26. *Consider a regular expression $r = a(b+c)d(b+c)^*$. The set of its derivatives is $\mathrm{Der}(r) = \{r, (b + c)d(b + c)^*, d(b + c)^*, (b + c)^*\}$. For derivative $(b + c)d(b + c)^*$ of r, its set of initial actions is $\mathrm{Init}((b + c)d(b + c)^*) = \{b, c\}$. Therefore, derivative $(b + c)d(b + c)^*$ is a b-site and a c-site but it is not an a-site. For b-site $(b + c)^*$ of r, its set of initial actions* *is Init$((b+c)^*) = \{b, c\}$. Sets of initial actions for all b-sites of r are equal, and this is true for all c-sites and a-sites. Therefore, expression r has equal choice property.*

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

In next subsection, we show a way of partitioning derivatives of a regular expression.

2.1.2 Partitions of Derivatives and States of Automata

Derivatives are analogous to states of a finite automaton. But finite automata are more succinct than regular expressions [GH08], so a state in a finite automaton may be related to more than one derivative of the corresponding regular expression. In this thesis we syntactically define a **partitioning** $Part_a(r)$ of the derivatives of r which retain a correspondence with states of an automaton where a is an initial action.

We syntactically partition the a-sites of s, each set of the partition containing those coming from a common source derivative, as follows.

Definition 27. *Let X_1 be a partition of a-sites of s_1 and X_2 be a partition of a-sites of s_2, where regular expression $s = s_1 \cdot s_2$ or $s = s_1 + s_2$. For partitions X_1, X_2 with blocks D_1, D_2 containing elements d_1, d_2 respectively, we use the notation $(X_1 \cup X_2)[d/d_1, d_2]$ for the modified partition $((X_1 \setminus \{D_1\}) \cup (X_2 \setminus \{D_2\}) \cup \{(D_1 \cup D_2 \cup \{d\}) \setminus \{d_1, d_2\}\}$. And, for partition X with block D_1 in it, having d_1 in it, $X[d/d_1]$ is the modified partition $X \setminus \{D_1\} \cup \{(D_1 \setminus \{d_1\}) \cup \{d\}\}$.*

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

Example 28. *For expression aa the partition of a-sites is:* $\mathrm{Part}_a(aa) = \{\{aa\}, \{a\}\}$. *For expression b it is* $\mathrm{Part}_a(b) = \emptyset$. *The a-sites of expression* $aa + b$ *can be partitioned by this representation:* $\mathrm{Part}_a(aa + b) = \{\{aa + b\}, \{a\}\}$. *The a-sites of expression* $(aa + b)^*aa$ *are:* $\mathrm{Part}_a((aa+b)^*aa) = \{\{(aa+b)^*aa\}, \{a(aa+b)^*aa\}, \{a\}\}$. *Finally, the a-sites of* $a^*(aa+b)^*aa$ *are described by the partition:* $\mathrm{Part}_a(a^*(aa+b)^*aa) = \{\{a^*(aa+b)^*aa\}, \{a(aa+b)^*aa\}, \{(aa+b)^*aa\}, \{a\}\}$.

Definition 29. *Given a set D of a-sites of regular expression s, an action a and a language L, we define the relativized language* $L^D = \{xay \mid xay \in L, \exists d \in \mathrm{Der}_x(s) \cap D, \exists d' \in \mathrm{Der}_{ay}(d)$ *with* $\epsilon \in \mathrm{Lang}(d')\}$, *and the prefixes* $\mathit{Pref}_a^D(L) = \{x \mid xay \in L^D\}$, *and the suffixes* $\mathit{Suf}_a^D(L) = \{y \mid xay \in L^D\}$. *We say that the derivatives in set D* ***a-bifurcate*** *L if* $L^D = \mathit{Pref}_a^D(L)\; a\; \mathit{Suf}_a^D(L)$. *(The left to right direction always holds.)*

Example 30. *Let* $L = \mathrm{Lang}((aa)^*) = \{(aa)^k \mid k \geq 0\}$. *Then* $L^{(aa)^*} = L^{a(aa)^*} = \{(aa)^k \mid k \geq 1\}$. *Hence we have,* $\mathit{Pref}_a^{a(aa)^*}(L) = \{a^{2k} \mid k \geq 0\} = \mathit{Suf}_a^{(aa)^*}(L)$ *and* $\mathit{Suf}_a^{a(aa)^*}(L) = \{a^{2k+1} \mid k \geq 0\} = \mathit{Pref}_a^{(aa)^*}(L)$. *The derivatives* $(aa)^*$ *and* $a(aa)^*$ *both a-bifurcate L, but the set* $D = \{(aa)^*, a(aa)^*\}$ *does not, as* $a^2 \in \mathit{Pref}_a^{a(aa)^*}(L)$, *and* $a^2 \in \mathit{Suf}_a^{(aa)^*}(L)$, *but* $a^2aa^2 \notin L^D$.

Proposition 31. *Every block D of the partition* $\mathrm{Part}_a(s)$ *a-bifurcates* $\mathrm{Lang}(s)$.

Proof. By induction on the definition. Base case $s = a$ for any a in Σ is easy as there is only one derivative.

(Case $s = s_1 + s_2$**)**: In the case that any block D of $Part_a(s)$ was part of $Part_a(s_1)$ or $Part_a(s_2)$ as it is, then it is clear that D a-bifurcates $Lang(s_1 + s_2)$. Now consider the case in which D was newly introduced. So, $D = D_1 \setminus \{s_1\} \cup D_2 \setminus \{s_2\} \cup \{s\}$, where $s_1 \in D_1$ and $s_2 \in D_2$, and $D_1 \in Part_a(s_1)$, and $D_2 \in Part_a(s_2)$. Let $w = xay \in L^D$, then it is clear that w is in $Pref_a^D(L)\ a\ Suf_a^D(L)$. Now consider a word w in $Pref_a^D(L)\ a\ Suf_a^D(L)$. So it is of the form $w = xay$, where $x \in Pref_a^D(L)$ and $y \in Suf_a^D(L)$. First, assume that derivative $d' \in D_1 \setminus \{s_1\}$ such that $d' \in Der_x(s_1)$ such that there exists a derivative d in $Der_{ay}(d')$ with $\epsilon \in Lang(d)$. Hence $xay \in Lang(s_1)$, implying $xay \in Lang(s_1 + s_2)$. Now, assume that derivative $d' \in D_2 \setminus \{s_2\}$ such that $d' \in Der_x(s_2)$ such that there exists a derivative d in $Der_{ay}(d')$ with $\epsilon \in Lang(d)$. Hence $xay \in Lang(s_2)$, implying $xay \in Lang(s_1 + s_2)$. Finally assume that if derivative $d' = s_1 + s_2$ then $x = \epsilon$ and there exists a derivative d in $Der_{ay}(d')$ with $\epsilon \in Lang(d)$. Therefore one of the following three cases must occur: there exists a derivative d_3 in $Der_{ay}(s_1)$ with $\epsilon \in Lang(d_3)$ or there exists a derivative d_4 in $Der_{ay}(s_2)$ with $\epsilon \in Lang(d_4)$ or both. So either $xay \in Lang(s_1)$ or $xay \in Lang(s_2)$ or both, implying $xay \in Lang(s_1 + s_2)$, in each of these cases as required.

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

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

Induction hypothesis, applied to the block $D[s_2/s_1 s_2]$ of $Part_a(s_2)$ (this is the reverse of the replacement in the definition of $Part_a(s_1 s_2)$), gives ay in $Lang(s_2)$. Since $\epsilon \in Lang(s_1)$, ay is in $Lang(s_1 s_2)$. So $xay \in L^{s_1 s_2} \subseteq L^D$.

(Case $s = s_1^*$**):** Since $x \in Pref_a^D(Lang(s_1^*))$, there exists a derivative d' in D such that $x \in$

$Pref_a^{d'}(Lang(s_1^*))$. We know that derivative d is of the form $d = d_1 s_1^*$, where d_1 is an a-site of s_1. Let $x = x_1 x_2 \cdots x_{l-1} x_l$ where $x_1, x_2, x_{l-1} \in Lang(s_1)$ and $x_l \in Pref_a^{d_1}(Lang(s_1))$. Since $y \in Suf_a^D(Lang(s_1^*))$, there exists a derivative d'' in D such that $y \in Suf_a^{d''}(Lang(s_1 \cdot s_2))$. We know that derivative d'' is of the form $d'' = d_2 s_1^*$, where d_2 is an a-site of s_1. Let $y = y_1 y_2 \cdots y_m$, where $y_2, y_3, \cdots, y_m \in Lang(s_1)$ and $y_1 \in Suf_a^{d_2}(Lang(s_1))$.

We know that, if $D_1 \in Part_a(s_1)$ then $D_1 \cdot s_1^*$ is a block in $Part_a(s_1^*)$. So d_1 and d_2, the a-sites of s_1 belong to some block D_1 of $Part_a(s_1)$. As $x_l \in Pref_a^{d_1}(Lang(s_1))$ and $y_1 \in Suf_a^{d_2}(Lang(s_1))$, using induction hypothesis, $x_l \cdot y_1 \in Lang(s_1)$. Therefore, $x_1 x_2 \cdots x_{l-1} \cdot a \cdot y_1 y_2 \cdots y_m \in Lang(s_1^*)$. Hence the proof. □

We give an example below to illustrate partitioning of derivatives.

Example 32. *Consider a regular expression $r = (aaa)^* aaa$ with $L = Lang(r)$. Its set of derivatives $Der(r) = \{d_1 = r = (aaa)^* aaa,\ d_2 = aa(aaa)^* aaa,\ d_3 = a(aaa)^* aaa,\ d_4 = aa,\ d_5 = a,\ d_6 = \epsilon\}$, where $Der_a(r) = \{d_2, d_4\}$, $Der_a(d_2) = \{d_3\}$, $Der_a(d_3) = \{d_1\}$, $Der_a(d_4) = \{d_5\}$, $Der_a(d_5) = \{d_6\}$.*

As an example of a set of a-sites of r, which do not a-bifurcate $Lang(r)$, consider $D = \{d_1, d_2\}$. Since, $aaa \in Lang(r)$, $a \in Suff_a^{d_2}(L)$, hence $a \in Suff_a^D(L)$, and, as $aaaaaa \in Lang(r)$, $aaa \in Pref_a^{d_2}(L)$, therefore $a \in Pref_a^D(L)$. But, $a \cdot a \cdot aaa \notin Lang(r)$, it is clear that D do not a-bifurcate $Lang(r)$.

For action a, the partition of a-sites of expression r are: $Part_a(r) = \{D_1 = \{d_1\},\ D_2 = \{d_2, d_4\},\ D_3 = \{d_3, d_5\}\}$.

In Figure 2.2 we give an automaton constructed using derivatives of $r = (aaa)^* aaa$.

Each block D of $Part_a(r)$, a-bifurcates $Lang(r)$. Which means that if there are two derivatives d and d' in D, and if a word $w = xay$ visits d_1 after x and word $w' = x'ay'$ visits d_2 after x', then words xay' and $x'ay$ also belong in the $Lang(r)$. This means that all such words passing through D, their prefixes (or past) before a are irrelevant. After reaching a

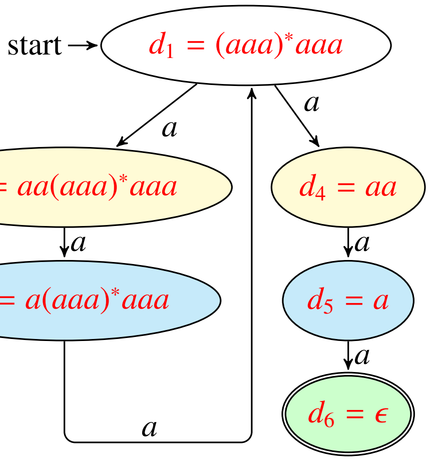


Figure 2.2: Automaton for expression $(aaa)^*aaa$

at D it can take some other words suffix (or future) and still get accepted in the language.

In the equation automaton of r, states of A relating to a block D can be collapsed, without changing the language accepted. Formally we prove this in Chapter 5. As of now we produce an automaton for using automaton of Figure 2.2, by collapsing states corresponding to a block into one state. This automaton is shown in Figure 2.3.

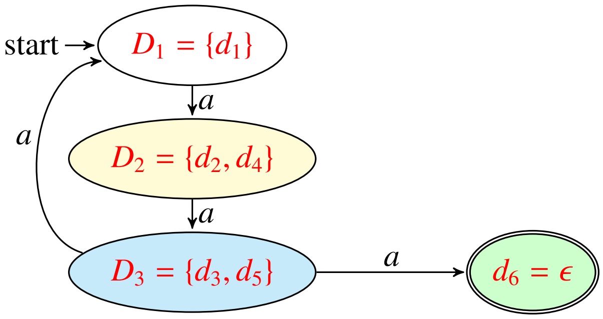


Figure 2.3: Automaton for expression $(aaa)^*aaa$

In next section we introduce various subclasses of product expressions and its derivatives.

2.2 Product Expressions

Now we introduce different kinds of product expressions over the distributed alphabet Σ corresponding to various net subclasses.

Connected-T-expression over Σ	$t ::= 0 \mid fsync(w_1, \dots, w_k)$, word w_i defined over Σ_i^*
Connected-FC-expression over Σ	$c ::= 0 \mid fsync(s_1, \dots, s_k)$, sum s_i defined over Σ_i
ω-T-expression over Σ	$f ::= t^\omega \mid par(t_1, t_2)$
ω-FC-expression over Σ	$o ::= c^\omega \mid par(c_1, c_2)$
Product expression over Σ	$e ::= fsync(r_1, \dots, r_k)$, regular expressions r_i defined over Σ_i^*

Sometimes we use the term connected expressions for both Connected-FC-expressions and Connected-T-expressions, similarly we use ω-expressions for both ω-T-expressions and ω-FC-expressions. These are all used in Chapter 3. The full class of product expressions is used in Chapter 5.

Definition 33 (equal choice Property for connected Expressions)**.** *A connected-FC-expression $e = fsync(s_1, s_2, \dots, s_k)$ is said to satisfy **equal choice** property if for all $i, j \in loc(a)$, s'_i is an a-site of s_i and s'_j is an a-site of s_j then s'_i and s'_j have equal choice.*

Example 34. *Let $\Sigma = (\Sigma_1 = \{a, c\}, \Sigma_2 = \{b, c\}), \Sigma_3 = \{a, b, c\})$. Consider the 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 $r'_1 = \mathrm{Der}_a(r_1) = c(ac)^*$, and $Init(r'_1) = \{c\}$. For r_3 we have, $r'_3 = \mathrm{Der}_a(r_3) = (b + c)(a(b + c))^*$, and $Init(r'_3) = \{b, c\}$. This violates equal choice property.*

Remark. All connected T-expressions satisfy equal choice property, hence their languages are always defined (but they could be empty).

Lemma 35. *The equal choice property for a connected-FC-expression e can be checked in $O(wd(e)^2|\Sigma|k)$.*

Proof. For given derivatives r'_i and r'_j, to check whether $Init(r'_i) = Init(r'_j)$ can be done in time linear in the size of $wd(r'_i) + wd(r'_j)$, which is $O(wd(e))$. For a given r_j and an action $a \in \Sigma$, to find $r'_j \in Der(r_j)$ such that $Der_a(r'_j) \neq \emptyset$ can be done in $O(wd(r_j) + 1)$ which is $O(wd(e))$. So checking equal-choice property at each local derivative takes $O(|\Sigma| wd(e)k)$ time. And, total number of such derivatives is $wd(e) + k$ which is $O(wd(e))$. Hence, total time needed to check equal choice property is $O(wd(e)^2|\Sigma|k)$. □

Now in the next subsection we give semantics of expressions whose syntax we saw at the begining of this subsection.

2.2.1 Semantics of Expressions

The semantics of each of the product expressions is a language over Σ. For connected-FC-expressions c it is a language of nonempty finite words. For ω-expressions e it is a language of infinite words. Because the distributed alphabet generates an independence relation (Section 1.1), we have languages of Mazurkiewicz traces under this independence relation.

For the connected expression 0, we have $Lang(0) = \emptyset$.

For the connected expression $e = fsync(s_1, s_2, \ldots, s_k)$, if e satisfies equal choice, its language is given as $Lang(e) = sync(Lang(s_1), Lang(s_2), \ldots, Lang(s_k))$ otherwise it is undefined.

Consider now the expression c^ω. For it $Lang(c^\omega) = [(Lang(c))^\omega]$, the trace closure under the independence relation, where $K^\omega = \{w_1 w_2 \cdots \mid \text{ for all } i, w_i \in K\}$. Each equivalence class is a set of infinite words.

Finally the semantics of the *par* operator is defined to be shuffle of languages.

2.2.2 Derivatives for Product Expressions

The definitions of derivatives can be easily extended to product expressions.

0 has no derivatives on any action.

Definition 36. *Let $e = fsync(s_1, s_2, \ldots, s_k)$ be a expression defined over distributed alphabet Σ. Then* ***global derivative*** *of e wrt an action a is the set of product expressions: $Der_a(e) = \{fsync(s'_1, s'_2, \ldots, s'_k) \mid$ for all $i \in loc(a), s'_i \in \mathrm{Der}_a(s_i)$ and for all $j \notin loc(a), s'_j = s_j\}$. If for every a, $Der_a(e)$ is empty, e is called a* ***deadlock****.*

For words, $Der_{aw}(e)$ is defined to be $Der_a(Der_w(e))$ by induction, with $Der_\epsilon(e) = e$. Let $\mathrm{Der}(e) = \{d \mid d \in Der_w(e)$ and $w \in \Sigma^*\}$ denote the set of all global derivatives of the product expression e. If no global derivative of e is a deadlock, we say that e is ***deadlock-free****.*

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.

A derivative d of e with global $a \in Init(d)$ is called an a-**site** of e.

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 are of $O(wd(r_1) \times wd(r_2) \times \cdots \times wd(r_k)$ which can be exponential in k.

Lemma 37. *Deadlock in a connected-T-expression c can be checked in time $O(wd(c)^2)$ and for a connected-FC-expression it can be checked in* NP.

Proof. The complexity bound for a connected-T-expression holds because we track at most $wd(c)$ tokens (represented by pointers in the expression) through a word of length at most $wd(c)$ to determine whether we reach the end of each T-sequence. This does not work for connected expressions: for example, $fsync(ab + ac, ad + ae + af)$ has six runs beginning with a in the resultant product. Now we use nondeterminism to guess the word letter-by-letter and move tokens. On any letter, if there is a derivative in one component of an $fsync$ but none in another, we have a deadlock. □

Theorem 38. *Equivalence checking of connected-T-expressions is polynomial time and for connected-FC-expressions is in* coNP.

Proof. Let $t_1 = fsync(w_1, w_2, \ldots, w_k)$ and $t_2 = fsync(u_1, u_2, \ldots, u_k)$ be two connected-T-expressions defined over distributed alphabet Σ. Let $L_1 = Lang(t_1)$ and $L_2 = Lang(t_2)$. First using Lemma 37 we check if there are deadlocks in t_1 and t_2 in polynomial time. If there are deadlocks in both, then languages of both expressions are empty and we are done. If there is deadlock in one and not in other then also we are done. So we have L_1 and L_2 both non-empty. We claim that to check if $L_1 = L_2$ it is sufficient to examine if for all $i \in \{1, \ldots, k\}, w_i = u_i$. We prove this claim below.

If for all $i \in \{1, \ldots, k\}, w_i = u_i$ then it clear that $L_1 = L_2$. Now assume that $L_1 = L_2$. We assume the contrary, i.e., for some i, $w_i \neq u_i$. Since there are no deadlocks in t_1 and t_2 both there exists a word $w \in L_1$ such that $w{\downarrow_{\Sigma_i}} = w_i$ and there exists a word $u \in L_2$ such that $u{\downarrow_{\Sigma_i}} = u_i$. Since $L_1 = L_2$, then $u \in L_1$ and $w \in L_2$, by definition of sync over words, it must be the case that $u_i = w_i$, which is a contradiction to our assumption that $u_i \neq w_i$. Hence, language equivalence checking of two connected-T-expressions is done in polynomial time.

For connected-FC-expressions, for each sum, we first get equivalent sequential system which is acyclic using Antimirov's construction. Hence we get a product system where each component is acyclic. For such systems language equivalence checking is in coNP [SJ09]. □

Corollary 39. *Emptiness checking of connected T-expressions is polynomial time and for connected expressions is in* coNP.

2.3 Conclusion

In this chapter we defined syntax and semantics of various classes of expressions which are used in thesis. We described a way of partioning derivatives of regular expressions. We also defined derivatives of product expressions. For describing various properties we used derivatives of expressions.

Chapter 3

Structurally cyclic Systems and Expressions

In this chapter we give correspondence between structurally cyclic product systems and ω-connected expressions. In this chapter we also give construction for obtaining acyclic FC-nets from connected expressions satisfying equal choice property.

3.1 Structurally cyclic Product Systems

Definition 40. *A product system over distributed alphabet Σ is called **T-product system**, if in each sequential system, every local state has at most one incoming local move and at most one outgoing local move.*

Definition 41. *A product system is **free choice**, more briefly an **FC-product system**, if for every a such that $|loc(a)| > 1$, every pair of a-labelled local moves is conflict-equivalent. We will also use **FC-dag** for FC-products in which each sequential system is acyclic , and we will also use **T-dag** for T-products in which each sequential system is acyclic.*

Remark. Each T-product system is also a FC-product system.

Definition 42. *A global state is **live** if for any run from it and any reachable global move $t = \Pi_{i \in Loc(a)} p_i \xrightarrow{a}_i q_i$, the run can be extended so that move t occurs. A product system is **live** if its initial global state is live.*

Definition 43. *A product system A with initial global state $\Pi_{i \in Loc} p_0^i$ is **structurally cyclic** if for all i, removal of each local state p_0^i from sequential system A_i makes resulting system acyclic.*

The Figure 3.1 shows a product system with only one sequential component in it. Removing p_1 does not eliminate all cycles in the sequential system, so it is not structurally cyclic.

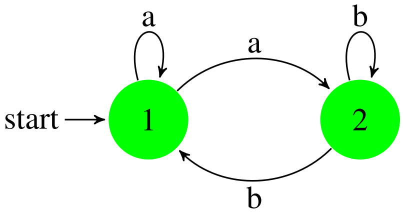


Figure 3.1: Product system: not structurally cyclic

3.2 Structurally cyclic Nets

Definition 44. *We say that a net system N is **structurally cyclic** if the initial marking M_0 is a* feedback vertex set *(that is, removing that set of places from N makes the resulting system acyclic).*

The Figure 3.2 shows a net system which is live and 1-bounded. Removing p_1 eliminates all cycles in the net, so it is structurally cyclic.

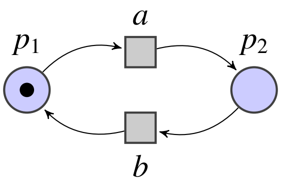


Figure 3.2: Structurally cyclic net system

3.3 From Connected-FC-Expressions to FC-dags and Acyclic Free Choice Nets

3.3.1 From Equal Choice Connected-FC-Expressions to deadlock-free FC-dags

Lemma 45. *Let c be a connected expression, satisfying equal choice property, defined over distributed alphabet Σ. Then there exists a connected FC-dag A free of active deadlocks which accepts $Lang(c)$. The size of the constructed system is $O(wd(c))$. From connected T-expressions, construction of T-product takes $O(wd(c)^2)$ time.*

Proof. For connected expression 0 we produce an empty product system.

Using Lemma 37, we can check deadlocks in NP for a connected-FC-expression, and in $O(wd(c)^2)$ time for a connected-T-expression.

If active deadlocks are present then, we return the empty product system, covered by the empty set! If there is no active deadlock, we proceed as follows:

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 construction in Theorem 25, such that $Lang(s_i) = Lang(A_i)$, for all $i \in \{1, \ldots, k\}$. This can be done in polynomial time of $wd(s)$.

So we get a product system $A = (A_1, A_2, \ldots, A_k)$ defined over Σ, in time $O(k \cdot wd(c))$.

$$
\begin{aligned}
w \in Lang(e) \text{ iff } & w{\downarrow_{\Sigma_i}} \in Lang(s_i) \\
\text{iff } & w{\downarrow_{\Sigma_i}} \in Lang(A_i) \\
\text{iff } & w \in Lang(A), \text{ by Lemma 14.}
\end{aligned}
$$

Therefore, $Lang(e) = Lang(A)$.

If c is equal choice expression then using Remark 2.1.1, we get that constructed product system is conflict-equivalent. □

3.3.2 From deadlock-free FC-dags to Acyclic Free Choice Nets

In this section, starting with a FC-dag we give a construction (using ideas from [CMT99]), to obtain an acyclic free choice net system. Hence, we get a construction from equal choice connected-FC-expressions to acyclic free choice nets.

Theorem 46. *Let A be a FC-dag without active deadlocks defined over distributed alphabet Σ. Then there exists an acyclic free choice net system accepting $Lang(A)$.*

Proof. From FC-dag A we construct an acyclic FC-net, utilizing the fact that $Lang(A)$, and hence the set of all prefixes of $Lang(A)$, is a finite set.

Consider any word w in this set and let $q = (p_1, \ldots, p_k)$ be one, out of possibly many, reachable global states of FC-dag on word w, where for each $i \in \{1, \ldots, k\}$ state p_i is reached in A_i on word $w{\downarrow}\Sigma_i$.

If there is a move from q to q' on a letter a in Σ_j, $j \neq i$, say $q' = (p'_1, \ldots, p'_k)$ with $p'_i = p_i$ for i outside $loc(a)$. We club p_j and p'_j into an i-interval of futures of the j'th component which are indistinguishable at i. However, if there is a move on a letter not in Σ_i from q to some other state q'' with j'th component p''_j, then p_j and p''_j are again in an i-interval, but the future and hence the interval may not be the same.

A longest i-interval $[p_j, r_j]$ of the j'th component from a beginning state p_j to an ending state r_j, both in the j'th component, has intervening paths in the j'th component solely through transitions outside Σ_i. Thus, from the i'th component, the j'th component could be in any one of these states. The net we will construct is covered by acyclic systems $1, \ldots, k$. The places of the i'th system of the net are tuples of longest i-intervals. That is, for every word w which is a prefix of $Lang(A)$, in the first acyclic system we have a place for the tuple $(p_1, [p_2, p_2'], [p_3, p_3'], \ldots, [p_k, p_k'])$ with longest 1-intervals for components 2 to k where the first component would have reached p_1 on $w{\downarrow}\Sigma_1$ (p_1 is an abbreviation for the 1-interval $[p_1, p_1]$) and the j'th component, $j \neq i$, would have reached a state in the 1-interval $[p_j, p_j']$. In the second acyclic system, on w we have a place for the tuple $([p_1, p_1''], p_2, [p_3, p_3''], \ldots, [p_k, p_k''])$ with longest 2-intervals in components 1 and 3 to k, where the second component would have reached p_2 on $w{\downarrow}\Sigma_2$ and the other components would be within their respective 2-intervals. Thus each place carries its system's view of the global states it could be in, differentiated by the end state which could have been reached, which is relevant for further interaction.

Now if $wa \in Pref(L(A))$, then it means that in the FC-dag we have a global move $q \xrightarrow{a} q'$. So there exist local moves $p_j \xrightarrow{a} p_j'$ for $j \in loc(a) = \{i_1, \ldots, i_l\}$. Let $Loc \setminus loc(a) = \{j_1, \ldots, j_l\}$. After processing w, let $r_j = ([..p_{i_1}], \ldots, [..p_{i_l}], [..p_{j_1}], \ldots, [..p_{j_l}])$ be the place produced in j-th system of net corresponding to state q such that, $\{p_{i_1}, \ldots, p_{i_l}, p_{j_1}, \ldots, p_{j_l}\} = \{p_1, p_2, \ldots, p_k\}$. For each q there is only one such r_j in the j-th system of FC-net. Now for each j, we produce a place $r_j' = ([..p_{i_1}'], \ldots, [..p_{i_l}'], [..p_{j_1}'], \ldots, [..p_{j_l}'])$ in the net, corresponding to the global state q' such that $\{p_{i_1}', \ldots, p_{i_l}', p_{j_1}', \ldots, p_{j_l}'\} = \{p_1', p_2', \ldots, p_k'\}$. Again for each q' there is only one such r_j' in the j-th system of FC-net.

Now we take new transition t labelled a, which has these places r_j as its set of pre-places, and places r_j' as its set of post-places. The transition constructed for a has exactly $|loc(a)|$ pre- and post-places. Firing this transition takes the net from the marking reached after w, corresponding to a global state reached after w in the product system, to a marking reached after wa in each j'th system which corresponds to a global state reached after wa in the product system. We have that $wa\downarrow\Sigma_1, \ldots, wa\downarrow\Sigma_k$ is reached in the tuple $q' = (p'_1, \ldots, p'_k)$ and the j-intervals beginning with p'_j reflect this for j in $loc(a)$, for other i the i-intervals do not change.

All transitions leading from w to its immediate extensions (that is, $wa, wb, \ldots$ which are prefixes of an accepted word, and there may be many transitions for each extension) form a cluster, which we prove below that it is free choice, in the case of a T-expression only one such extension is possible, so the result is a T-net.

We repeat this, for each word w which is prefix of some word in $Lang(A)$.

To prove that constructed net is free choice, consider places r_1 and r_2 of some cluster and transitions t_1 labelled a and transition t_2 labelled b, such that $t_1 \in {r_1}^\bullet \cap {r_2}^\bullet$ and $t_1 \in {r_1}^\bullet$. We have to prove that $t_1 \in {r_2}^\bullet$. Since an a-labelled transition is in the post of both the places r_1 and r_2 it must be the case that there exists a global reachable state q of the product system which corresponds to these two places(i.e., set of local states of q is same as the set of end states of all the intervals in r_1 and, same is true for r_2 also), and q enables an a-labelled global move in the FC-dag. Let q be one of the states reached in FC-dag after reading some word w. Since we have $t_2 \in {r_1}^\bullet$ as well, it implies that q also enabled an b-labelled global move in the FC-dag. Since r_2 is a place which corresponds to q, by construction, $r_2 \in {t_2}^\bullet$ as required.

From above construction it is clear that for each global move labelled a, which is firable in the FC-dag at some state q, we have a transition with the same label, and with the preset of places which correspond to q. Also, in FC-dag each global move is fired only once and so is the corresponding transition in the net. When we reach state q in FC-dag, we have a marking in the net which marks its corresponding places and vice versa. Hence, FC-dag and constructed net are language equivalent. □

As an example of above construction consider equal choice, connected-FC-expression:

$e = fsync(b(cad + cae), fag)$. Figure 3.3.2 shows a FC-dag with no active deadlocks, constructed from expression e using Lemma 45 for its language. Figure 3.4 shows acyclic FC-net system constructed from it using Theorem 46. It is clear that FC-product dag and acyclic FC-net are language equivalent, but FC-net system has active deadlocks in it.

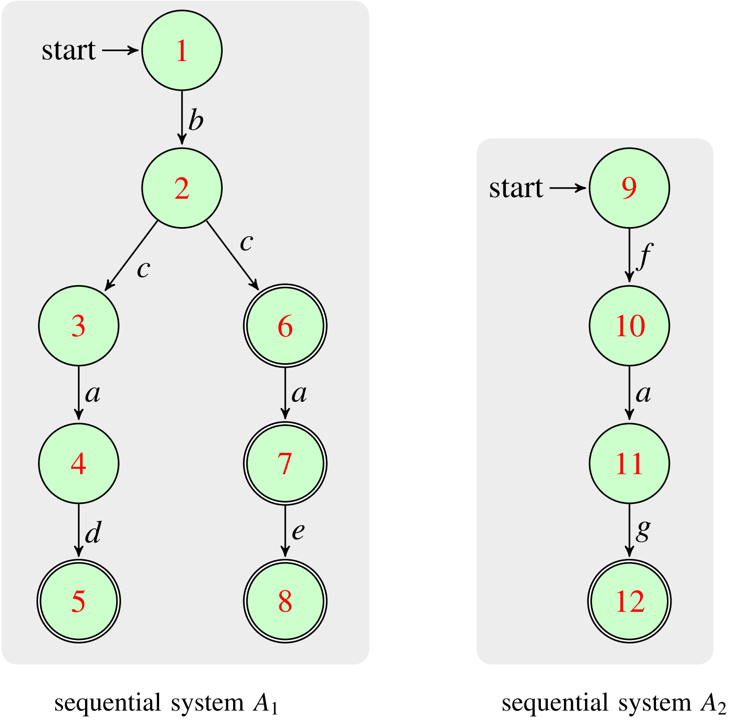


Figure 3.3: FC-dag for $fsync(b(cad + cae), fag)$

Using construction in Theorem 46 we get acyclic FC-net system(possibly with active deadlocks) from an FC-dag with no active deadlocks. But starting from a T-dag with no active deadlocks, we get an acyclic T-net system with no active deadlocks.

Corollary 47. *Let A be T-dag with no active deadlocks then we can construct acylic T-net system with no active deadlocks for its language Lang(A).*

Proof. From Theorem 46 we get language equivalence acyclic net system for T-dag A. Each sequential system in A is a path, and so each sequential component in acyclic T-net system is also a path. Again by construction, each such sequential component has only one initial place marked, and has only one final place. □

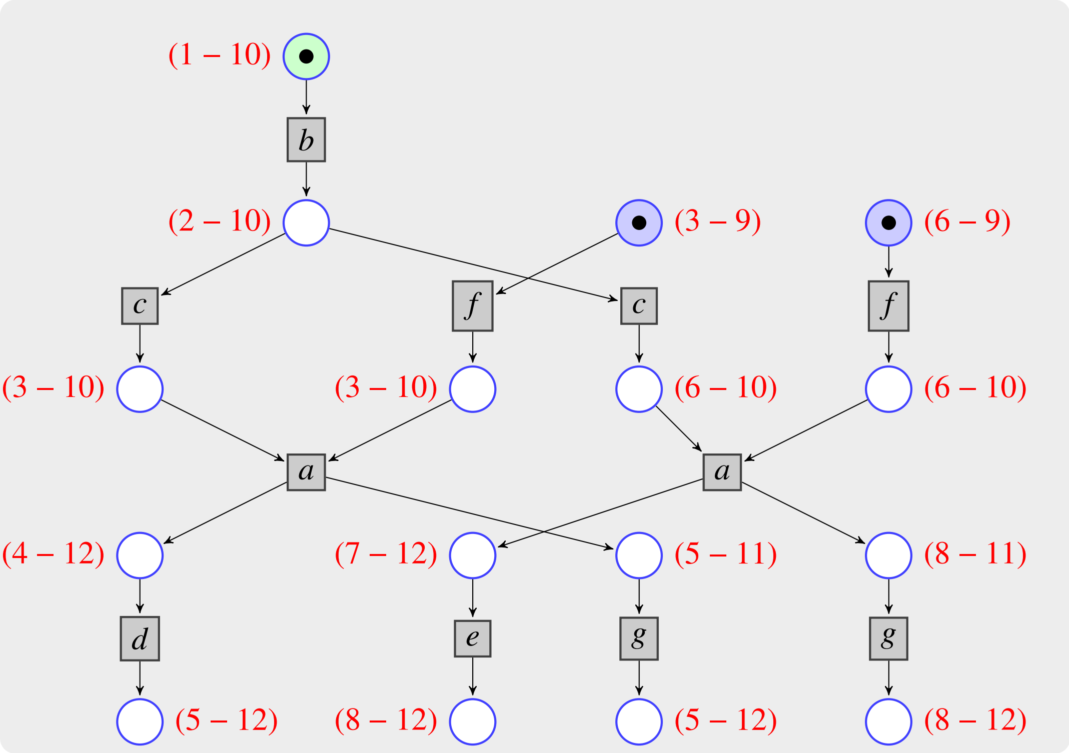


Figure 3.4: Acyclic FC-net for $f sync(b(cad + cae), fag)$

3.4 From equal choice ω-FC-Expressions to live, Structurally cyclic FC-Product Systems

For the expression c^ω we map in polynomial time the ω-power operation to the construction of an FC-product. If it is an ω-T-expression, we can also get a live T-net system.

Lemma 48. *Let $e = c^\omega$ be an ω-FC-expression over distributed alphabet Σ with $Lang(c)$ a nonempty language of nonempty words. Then there exists a live and structurally cyclic FC-product accepting $Lang(e)$. The size of the constructed system is $O(wd(c))$. From ω-T-expressions, construction of T-products takes $O(wd(c)^2)$ time.*

Proof. For the expression c^ω over distributed alphabet Σ, consider the deadlock-free, connected FC-dag $A = (A_1, A_2, \dots, A_k)$ for c, accepting the language K, obtained from the previous Lemma 45. Recall that the trace equivalence generated from the independence relation of Σ saturates K, that is, $K = [K]$.

For each acyclic sequential system A_i, we fuse the initially marked places of A_i with its sink places (which are different since K does not have the empty word). Call this transition system A'_i. Hence we get new product system $A' = (A'_1, A'_2, \ldots, A'_k)$. The product A' satisfies the following properties:

(1) Since A is a FC-product, A' is an FC-product.

(2) It is structurally cyclic since by construction the initial global state is a feedback vertex set.

(3) Fusing the sink and source places makes each acyclic system of A strongly connected in A'.

Now we prove that (A', M_0) is live.

Assume a reachable global state M and a reachable global move t. Since there are no active deadlocks, we have a firing sequence which will take us from M to some final state of A. The same firing sequence will take us from M to M_0 in A'. Now t is reachable from M_0.

We now show that the language of A' is $Lang(e) = [Lang(c)^{\omega}]$.

By construction $K^{\omega} \subseteq Lang(A')$. Since A' has the same locations as A, it generates the same trace equivalence and hence we have that $[K^{\omega}] \subseteq [Lang(A')] = Lang(A')$.

To prove the converse inclusion, $Lang(A') \subseteq [K^{\omega}]$, suppose not and we have w accepted by A' but not in $[K^{\omega}]$. We can remove prefixes of w which are in $[K]$, so let us assume $w = uav$, u is a proper prefix of K and ua is not a prefix of a word in $[K^{\omega}]$. Since A was deadlock-free, there is some extension ub that is a prefix of K such that b is enabled after executing u. If a and b are dependent and they are both enabled, in a FC-dag they have the same locations, and ua would be a prefix of K as well. Hence a and b are independent and we can commute them. We apply this argument repeatedly to increase the length of the prefix; but since K is a finite language, after some point we will find that $w = uav$ for some $u \in [K]$ after which a is enabled, hence a is enabled at the initial global state of A. We can remove this prefix and again continue the argument. This shows that w is in $[K^\omega]$, a contradiction.

Size of the product system constructed and required time follows from Lemma 45. □

For the expression $par(e_1, e_2)$, all occurrences of letters in e_1 are independent of those in e_2, so that the net corresponding to them is obtained by taking the disjoint union of the two component product systems, and its language is the shuffle of the two sublanguages. Clearly the size of the constructed system is $O(wd(e_1)) + O(wd(e_2))$. So we conclude:

Theorem 49. *For every ω-expression e, there is a live and structurally cyclic FC-product of size $O(wd(e))$ accepting $Lang(e)$.*

Corollary 50. *Let f be an ω-T-expression over distributed alphabet Σ with $Lang(c)$ a nonempty language of nonempty words. Then there exists a live T-net system accepting $Lang(f)$.*

Proof. For the expression c^ω, consider the connected T-dag $B = (B_1, B_2, \ldots, B_k)$ for c with no active deadlocks accepting the language K, obtained from Lemma 45. Recall that the trace equivalence generated from the independence relation of Σ saturates K, that is, $K = [K]$. Now using Theorem 46, we get an acyclic T-net system (A, M_0) for its language, $Lang(B)$. By Corollary 47, and by construction, each such sequential component has only one initial place marked, and has only one final place.

For each sequential component A_i, we fuse the initially marked places of A_i with its sink places (which are different since K does not have the empty word). Call this net system A'_i. Hence we get new net system (A', M_0). By following the proof of Lemma 48 this is a live T-net system accepting $Lang(f)$.

This can be extended to the *par* operation by taking disjoint union of the T-systems. □

3.5 Structurally cyclic FC-products to equal choice ω-FC-Expressions

In this section we discuss how to build language equivalent expressions for a given FC-product. We follow the same strategy as in the previous section, working through dags and FC-dags before tackling the general case.

Lemma 51. *Let A be a sequential system which is a dag. Then there exists an equivalent sum s for its language. The alphabetic width of this expression is quadratic in A and it can be computed in time quadratic in A.*

Proof. First, we delete all nodes unreachable from initial state. Then for each move, we consider any path starting from initial state and reaching some final state, which includes this move. Let $\langle p, a, p' \rangle$ be a move. Let p_0 be the initial state. Since A is a dag, Then finding a path which leads from p_0 to p is linear time [CLRS01] and if q is some final state of A then again finding existence of a path from p' to q is in linear time. We know that for at least one final state for which there is a path from p' to it. Joining these paths we get a path which includes label of this particular local move. This is done in a linear time in the size of A. We write down sequence of labels of this path starting from p_0 to q.

Then we write down a sum expression which has sequences for all these paths. This can be done in quadratic time in the size of A. Clearly, it is language equivalent to sequential dag system we started with.

Each move appears in a path and the length of each path is linear in A which gives a quadratic upper bound for the size of expression also.

□

Next, we construct expressions for FC-dags. We do not check whether the expression has deadlocks.

Lemma 52. *Let A be a connected FC-dag. There is a equal choice connected expression c for $Lang(A)$ of alphabetic width $O(|A|^2)$ which can be computed in $O(|A|^3)$ time.*

Proof. Using Lemma 51, we obtain in quadratic time equivalent sum expressions s_i of size quadratic in the alphabetic width, for each sequential component of the product. Its alphabetic width is quadratic in the size of A. Language equivalence proof is similar that of Lemma 45 given in previous section.

For all i, sum s_i is of the form $s_i = w_{i_1} + w_{i_2} + \ldots + w_{i_l}$, where each word w_{i_h} describes a path from initial state to some final state of A_i.

Let $i,\ j$ belong to $loc(a)$. Consider the case where s_i and s_j are a-sites themselves. In sequential systems A_i and A_j, respective initial states p_0^i and p_0^j have outgoing local moves on action a. If there some local move on action b at p_0^i then as A is conflict-equivalent, p_0^j also have an outgoing local move on action b. This implies that, if s_i is an b-site then s_j must be a b-site. Therefore, s_i and s_j have equal choice.

Now consider the case where $s_i' \in \mathcal{PD}(s_i)$ and $s_i' \neq s_i$. Then, s_i' is just a word over Σ_i. Hence in Σ, there is only one action, with which it has a non-empty derivative. This is true for any $s_j' \in \mathcal{PD}(s_j)$ and $s_j' \neq s_j$. Therefore if s_i' and s_j' are a-sites, then it is trivially true that they have equal choice. □

Finally we have a cubic time algorithm from live structurally cyclic FC-products to ω-expressions.

Theorem 53. *Let A be a live, structurally cyclic FC-product. Then we can compute in cubic time an ω-FC-expression of alphabetic width $O(|A|^2)$ for the accepted language.*

Proof. Consider A a given live, structurally cyclic FC-product. Each A_i is structurally cyclic, and its initial state $\{p_0^i\}$ is feedback vertex set. Now we adopt a small trick. Make a copy $p_0'^i$ of the place p_0^i in A_i change the system so that the edges coming into p_0^i are replaced by edges into the corresponding places of $p_0'^i$. Since $\{p_0^i\}$ is a feedback vertex set, the resulting sequential system A'_i is a dag. This can be done in time $O(|A_i|)$. Since, product system A was connected and live, resultant system A' is a connected and deadlock-free FC-dag of size $O(|A|)$.

By Lemma 52 we can compute in $O(|A|^3)$ time a connected expression c of alphabetic width $O(|A|^2)$ for this FC-dag. We claim the expression c^ω describes the language of the original product system (A, M_0). The proof follows the same arguments as in Lemma 48.

□

For each connected FC-product, use the argument above, and then use the *par* operator to obtain the shuffle of the languages. This preserves both the time complexity and the expression's alphabetic width. We can extend the result above to deal with product systems which are not necessarily live, but structural cyclicity is crucially used. The constructed expression is not checked for deadlocks.

Corollary 54. *Let A be a structurally cyclic product FC-system. Then we can compute in polynomial time a shuffle of connected and ω-expressions, of alphabetic width polynomial in* $|A|$*, for the accepted language.*

Finally, the algorithms of this section can be seen to produce T-sequences, connected T-expressions and ω-T-expressions in case we are given a product T-system which is path-like, a T-dag and live, respectively, since T-systems are structurally cyclic. Thus we have efficient Kleene characterizations for product T-systems as well.

3.6 Conclusion

In this chapter we worked with expressions for structurally cyclic product and net systems.

- For connected FC-expressions, we constructed equivalent acyclic FC-product systems and acyclic free choice net systems. Conversely from acyclic FC-product systems we obtained equivalent connected FC-expressions.

- For connected T-expressions, we constructed equivalent acyclic T-product systems and acyclic T-net systems. Conversely from acyclic T-product systems we obtained equivalent connected T-expressions.

- For ω-FC-expressions, we constructed equivalent live and structurally cyclic FC-product systems. We also proved the converse.

- For ω-T-expressions, we constructed equivalent live and structurally cyclic T-product systems and T-net systems. We also proved the converse for T-product systems.

To go from net systems to product systems, we need to use S-decomposability. This is done in the next chapter in a more general setting: see Corollaries 72 and 73. We conjecture that for an ω-FC-expression there is an equivalent live and structurally cyclic free choice net system.

Chapter 4

Free Choice Nets and Product Systems with Matching

In this chapter we consider product systems with separation of labels, and with matchings. We give constructions from these product systems to nets with unique cluster property and without it. We give constructions in the reverse direction also.

4.1 Properties of Product Systems

Definition 55. *A product system A is* ***deterministic for global actions*** *if for every global action a, every place has only one outgoing a-move.*

Definition 56. *A product system A is said to have* ***separation of labels*** *if for all $i \in Loc$, whenever $\langle p, a, p' \rangle, \langle q, a, q' \rangle \in \rightarrow_i$ then $p = q$.*

A system having separation of labels property may have many a-labelled moves in each of its sequential component, but all of them are outgoing moves from an unique place in it.

Example 57. *Let $\Sigma = \{a, b\}$ be a distributed alphabet with distribution $(\Sigma_1 = \Sigma_2 = \Sigma)$.*

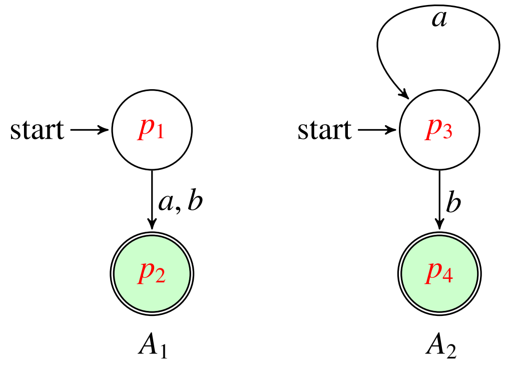


Figure 4.1: Product system $A = (A_1, A_2)$ with separation of labels

Consider the product system $A = (A_1, A_2)$ shown in Figure 4.1. For global action a, place p_1 is the only place in A_1 having outgoing a-moves and, place p_3 is the only place in A_2 having outgoing a-moves.

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

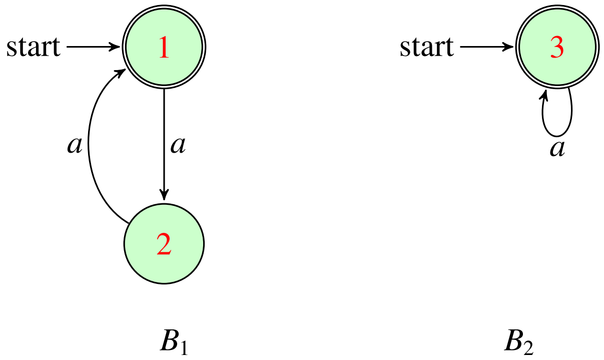


Figure 4.2: Product system $B = (B_1, B_2)$ without separation of labels

On the other hand, consider product system $B = (B_1, B_2)$ shown in Figure 4.2, and defined over the distributed alphabet $\Sigma' = \{a\}$ having distribution $\Sigma'_1 = \Sigma'_2 = \Sigma'$. Since sequential system B_1 has more than one place having outgoing a-moves, product system B does not have the separation of labels property.

Proposition 58. *Let A be a product system defined over distributed alphabet Σ. Then we can check if it satisfies separation of labels property in* PTIME.

Proof. In each sequential machine A_i of A, for each global action a of Σ, check if there is only one place having outgoing local a-moves. This requires visiting each place only once. □

The first property for a product system identifies synchronizations which will come together into a cluster of a free choice net. We also define another stronger property.

Definition 59 (matching, conflict-equivalent matching)**.** *For global $a \in \Sigma$, an a-matching is a subset of tuples $\Pi_{i \in loc(a)} P_i$ such that, each place $p \in P_i, i \in loc(a)$ having an outgoing local a-move(i.e., if $\exists \langle p, a, q \rangle \in \rightarrow_i$), appears in exactly one tuple of a-matching. When two places p and p' appear in a tuple of a-matching, then we say that they are matched on action a. We say a product state R is in an a-matching if its projection $R \downarrow loc(a)$ is in the a-matching. An a-matching of a product system is said to be* ***conflict-equivalent*** *if any two places which are matched on action a are conflict-equivalent.*

Definition 60 (product system with matching, FC-matching product)**.** *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 all $i, j \in loc(a), \langle p, a, q \rangle \in \rightarrow_i$, the pre-place p is matched to a pre-place p' such that there exists a local a-move $\langle p', a, q' \rangle$ in $\rightarrow_j$.*

We call a product system an ***FC-matching product*** *if it has a conflict-equivalent matching for each global action a.*

In earlier Chapter 3 we used the definition of an FC-product. The definition of FC-matching product is a generalization since conflict-equivalence is not required for all a-moves uniformly but refined into smaller equivalence classes depending on the matching.

Example 61. *Let $\Sigma = \{a, b, c\}$ be a distributed alphabet with distribution $(\Sigma_1 = \Sigma_2 = \Sigma)$. Consider the product system $A = (A_1, A_2)$ shown in Figure 4.3. The matching relations are:* matching$(b) = \{(1, 4)\}$, matching$(a) = \{(2, 5), (1, 4)\}$ *and* matching$(c) = \{(2, 5)\}$.

The local move $\langle p_1, a, p_2 \rangle \in \rightarrow_1$ in A_1 is conflict-equivalent with local move $\langle p_4, a, p_5 \rangle \in \rightarrow_2$, but it is not conflict-equivalent with local move $\langle p_5, a, p_7 \rangle \in \rightarrow_2$.

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

Since local move $\langle p_1, a, p_2\rangle \in\rightarrow_1$ is not conflict equivalent with local move $\langle p_5, a, p_7\rangle \in\rightarrow_2$, for global action a, not all local a-moves are conflict-equivalent to each other. Therefore, product system A is not an FC-product.

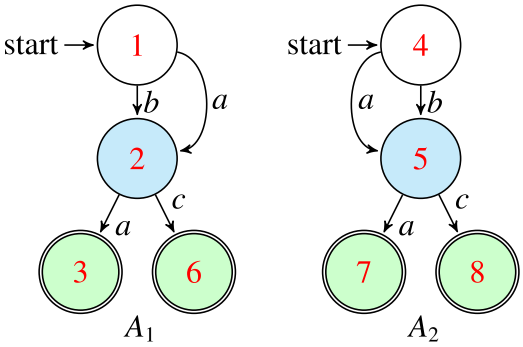


Figure 4.3: Product system with matching of labels

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

Proposition 62. *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 $j \in loc(b)$. □

The next definition is semantic. If a system has separation of labels, the property obviously holds.

Definition 63 (consistency with matching)**.** *Let A be a product system with matching of labels. A run of A is said to be* ***consistent with a matching of labels*** *iff for all global actions a and every prefix of the run* $R^0 \overset{v}{\Rightarrow} R \overset{a}{\Rightarrow} Q$*, the pre-places* $R\downarrow loc(a)$ *are in the a-matching.*

Proposition 64. *For product system A with matching of labels, checking if A is FC-matching product can be done in* PTIME*, and checking if all runs of A are consistent with given matching of labels can be done in* PSPACE*.*

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

To check if all runs of A are consistent with given matching of labels we need to visit each reachable global state of A at most once, which can be done in PSPACE. Let n be the maximum number of states any A_i have, and v_i be $\log(n)$ bit vector which can store state of A_i. Let $(v_1, \ldots, v_k)$ be a $k\log(n)$ bit tuple which can store a product state of A. Maximum number of product states of A is n^k. Hence, to reach any such state, from initial state, maximum length of word is n^k. An $k\log(n)$ bit vector is required to store the length of word.

Now we design a NDTM M, which accepts A iff it is not consistent with matching of labels . To check this it has to guess a reachable state R, which is not in matching. Each computation of M guesses a word of length upto n^k. Then verifies if it A has a run on it, at the same time for each reachable R enabling some global action a, checks if R is in matching(a). If there exists such R, then M outputs YES, and accepts input A and halts. By Savitch's theorem, we have a DTM which decides if A is consistent with matching of labels in PSPACE. □

4.2 Properties of Nets

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

Definition 65. *A labelled net $N = (S, T, F, \lambda)$ is said to have the* ***unique cluster property*** *(briefly,* ***ucp****) if for all globals $a \in \Sigma$, there exists at most one cluster in which all transitions labelled a occur. N is* ***deterministic for synchronization*** *if for every global a, every cluster contains at most one a-labelled transition.*

4.2.1 Distributed Choice and Direct Product Representation of Nets

In Figure 1.1 we saw that nets should be restricted in some way so that one obtains direct product representability. In this section we identify this condition. It is called **distributed choice**.

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

Clearly $post(C_a) \subseteq postdecomp(C_a)$. The following definition appears to be new and is key to direct product representability. It says that every post-decomposition is represented in the cluster.

Definition 66. *An S-decomposable net $N = (S, T, F, \lambda)$ is said to be* ***distributed choice*** *if, for all a in Σ and for all clusters C of N, $postdecomp(C_a) \subseteq post(C_a)$.*

We will use distributed free choice for nets which are distributed choice as well as free choice. Note that an unlabelled S-decomposable net can be thought of as being labelled by its set of transitions T, in which case the definition is satisfied. Our example in Figure 1.1 is not distributed choice.

Proposition 67. *For an S-decomposable net $N = (S, T, F, \lambda)$ checking distributed choice is in* PTIME.

Proof. For checking this condition we have to visit each cluster only once. Then for each symbol we have to check the condition if $postdecomp(C_a) \subseteq post(C_a)$. We know that $post(C_a) \subseteq \Pi_{i \in loc(a)} S_i$, as well as $postdecomp(C_a) \subseteq \Pi_{i \in loc(a)} S_i$. And, $post(C_a) \subseteq postdecomp(C_a)$. Hence, to check $postdecomp(C_a) \subseteq post(C_a)$, we have to just check if $|postdecomp(C_a)| \subseteq |post(C_a)|$, Clearly $|post(C_a)| = |C_a|$.

$|C_a[i]| = |C_a{}^{\bullet} \cap S_i|$ is bounded by C_a because each transition contributes at most one place to this set by S-decomposability and there are at most $|C_a|$ transitions.

Hence, $|\Pi_{i \in loc(a)} C_a[i]| \leq k|C_a|$. Therefore, $|postdecomp(C_a)| \leq k|C_a|$. Both sets can be counted in $k|C_a|+|C_a|$ time. Hence one cluster take at most $|\Sigma|(|k|C_a|+|C_a|)$ time. Therefore to check this condition for all clusters in the net is PTIME in the size of net. □

As we will see, the definition of distributed choice nets is required in the proof which goes from nets to product systems, in Chapter 3 and Chapter 4. This is independent of the definition of free choice. Thus we see distributed choice as a condition which has remained hidden since most of the work on nets did not consider labellings.

A free choice net allows us to choose between letters of alphabet, where all of them should either be global or all of them should be local. Using the external choice and internal choice operators of process algebra [Mil80, Hoa85], $a + b$ of free choice is either $a \sqcap b$ or $a \Box b$. The distributed choice property, applied to conflicts between various events labelled with a single global action, says it does not matter which one is taken, all possibilities should be available. So $ab \Box ac = (a \Box a)(b + c)$, where a is a global action: we can also say $ab \Box ac = (a \Box a)(b + c) = a(b + c)$.

One might ask whether the complicated condition of Definition 66 is really necessary for product decomposition. We first considered just the cardinality between the two sets, $post(C_a)$ and C_a i.e., $|post(C_a)| = |C_a|$. But our initial example net shown in Figure 1.1, satisfies this condition and in Theorem 19 we have shown that its language is not direct product representable.

Next we considered the cardinality between the two sets, $postdecomp(C_a)$ and C_a i.e., $|postdecomp(C_a)| = |C_a|$. Unfortunately a variant of our initial example which satisfies this condition is not direct product representable. The net is S-decomposable, free choice and satisfies the unique cluster property. For $\{p_1, p_2\}$ the only final marking, the labelled net shown in the Figure 4.4, accepts the Mazurkiewicz trace language $L = \{abd, adb, ace, aec\}^*$ over the distribution $\Sigma = (\Sigma_1 = \{a, b, c\}, \Sigma_2 = \{a, d, e\})$.

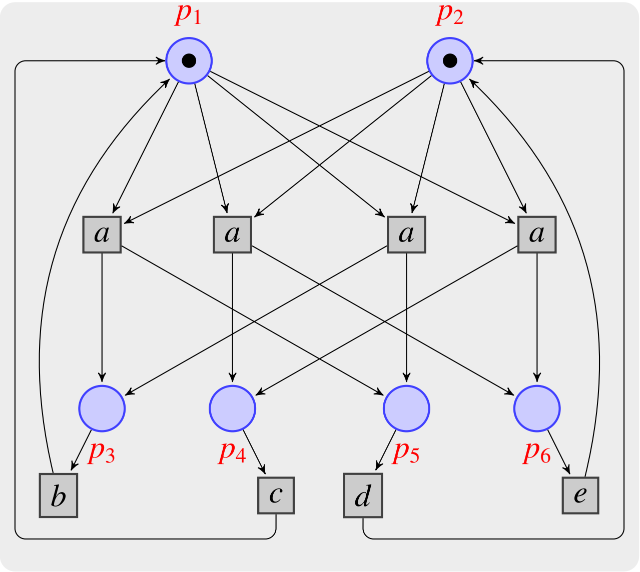


Figure 4.4: Labelled free choice net, which is not direct product representable

Proposition 68. *There is no direct product automaton over Σ representing the language of net shown in Figure 4.4.*

Proof. Let $w = abeacd$. Then $w_1 = w{\downarrow_{\Sigma_1}} = abac$ and $w_2 = w{\downarrow_{\Sigma_2}} = aead$. Since both $w_1, w_2 \in L$ But we have $u_1 = abdaec \in L$ with $u_1{\downarrow_{\Sigma_1}} = abac$ and $u_2 = aecadb \in L$ with $u_2{\downarrow_{\Sigma_2}} = aead$. So using the characterization given in Proposition 14, we get word $abeacd \in L$ which is a contradiction. □

4.3 Distributed Free Choice Nets to FC-matching 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 describe a simple generic construction of a product system from a net which is S-decomposable and distributed free choice. By assuming more properties, we get more properties of the constructed product system. In the next section, we do another simple generic construction of a net from a product system, the properties of S-decomposability and distributed choice are obtained automatically, and again we can get more properties if desired.

Let $(N, M_0, \mathcal{G})$ be a 1-bounded and S-decomposable labelled net system, where $N = (S, T, F, \lambda)$ is the underlying net. Let $N_i = (S_i, T_i, F_i)$ denote components in the S-cover, for all i in $\{1, 2, \ldots, k\}$. We define $P_i = S_i, G = \{(M \cap P_1, \ldots, M \cap P_k) \mid M \in \mathcal{G}\}$. If $\mathcal{G}$ was a direct product set of final markings, we can define $G_i = \{M \cap P_i \mid M \in \mathcal{G}\}$ and set G to be their product $G_1 \times \cdots \times G_k$. Let p_i^0 be the unique state in $M_0 \cap P_i$. For each $t \in T_i$, we know that, there exist places $p, p' \in S_i$ such that (p, t) and (t, p') belong to F_i. Formally we define set of local moves, $\rightarrow_i = \{\langle p, \lambda(t), p' \rangle \mid t \in T_i \text{ and } (p,t),(t,p') \in F_i, \text{ for } p, p' \in P_i\}$. So we get sequential system $A_i = \langle P_i, \rightarrow_i, p_i^0 \rangle$ corresponding to the component (S_i, T_i, F_i). Hence we get the product system $A = (A_1, A_2, \ldots, A_k)$ over distributed alphabet Σ. The size of A is linear in the size of the net. If N was deterministic for synchronization then the constructed system A is deterministic for global actions.

Theorem 69. *Let $(N, M_0, \mathcal{G})$ be a 1-bounded and S-decomposable labelled distributed choice net system, and A is the product system constructed in the beginning of Section 4.3. Then*

1. *If net is free choice then A is a FC-matching product and*
2. *in addition if the net system is live, then*
   (a) *all runs of A are consistent with the matching.*
   (b) *Lang$(N, M_0, \mathcal{G})$ = Lang(A).*

Proof. Let $N = (S, T, F, \lambda)$ be the underlying net. Let M be a reachable marking. Since A_i is a component, number of tokens in P_i remain constant, and we know that P_i had one token at M_0, so we always have one unique $r_i \in P_i$ such that $r_i = M \cap P_i$, for all i in Loc. So we get $R(M) = (r_1, r_2, \ldots, r_k)$, which is a product state. In the reverse direction, for any product state R, taking union of all places in R gives us a unique set of places $M(R)$ which serves as a marking of net.

1. Since net is free choice, we get a conflict-equivalent matching of labels by, taking tuples of pre-places of a cluster, making A an FC-matching product.
2. Assume that the net system is live.
   (a) Suppose we have two places, say p_1 in location 1 and p_2 in location 2 which are not matched on any action a, which means that they are not in the same

cluster of the free choice net N. Again by construction, let the local move $p_1 \xrightarrow{a} p'_1$ be obtained from the net transition t_1 with pre-place p_1 coming from cluster $C_1 = (S_1, T_1)$. By S-decomposability, C_1 has a matching pre-place q_2 in location 2. Similarly let $p_2 \xrightarrow{a} p'_2$ be obtained from t_2 with pre-place p_2 coming from cluster $C_2 = (S_2, T_2)$, with matching pre-place r_1 in location 1 using S-decomposability.

Since we started with a 1-bounded S-cover, initially each component had only one token in it, and in a component number of tokens remains constant at any reachable marking. So the places q_2 and r_1 are distinct from p_1 and q_2 respectively. Again using the 1-bounded S-cover, q_2 and r_1 have to be unmarked at the net marking $M(R)$. By liveness, some transition of C_1 will get fired and by free choice, for this we need to have tokens in all places of S_1. To bring the token in to q_2 we have to fire some transition in C_2, for which by free choice we need to put a token in place r_1, which has to come from p_1. In short we have two dead places from where a transition can never be fired, contradicting liveness.

This means that all the places $p_1, \ldots, p_l$ are in the same cluster. As a consequence the run of the product system at this marking (and inductively at all reachable markings) will be consistent with the matching of labels defined in the construction.

(b) We will show language equivalence by showing the stronger property that the maps $R(.)$ and $M(.)$ constitute an isomorphism of reachable markings of N with reachable product states of A. Clearly this is the case for the initial marking and the initial product state.

To prove $Lang(N, M_0, \mathcal{G}) \subseteq Lang(A)$, we show that for a transition $t \in T$ of the net system, labelled a, and $M[t\rangle M'$ in the net, then we have a global move g in the product system with label a yielding $R(M) \overset{a}{\Rightarrow} R(M')$ in the product system. We know that ${}^\bullet t \subseteq M$ and $t^\bullet \subseteq M'$. Hence for each $i \in loc(t)$ we have a place $r_i \in P_i$ such that $(r_i, t) \in F_i$. And since N_i corresponding to A_i is a component there exists another place $r'_i \in P_i$ such that $(t, r'_i) \in F_i$. But by construction we get a transition $\langle r_i, a, r'_i \rangle \in \rightarrow_i$, $\forall i \in loc(t)$. So by definition of a product system we get a global move $g = \Pi_{i \in loc(t)} \langle r_i, a, r'_i \rangle$ in constructed product system A. So we get ${}^\bullet t = \textsf{pre-places}(g)$ and $t^\bullet = \textsf{post-places}(g)$. So each $r_i \in \textsf{pre-places}(g)$ also belong to tuple $R(M)$ at i-th place. Repeating the same argument for M' we get that $r'_i \in \textsf{post-places}(g)$ also belong to tuple $R(M')$. So in the product system we have $R(M) \overset{a}{\Rightarrow} R(M')$.

In the reverse direction, to prove $Lang(A) \subseteq Lang(N, M_0, \mathcal{G})$, we show that, if $R \overset{a}{\Rightarrow} R'$ in the product system using a global move g then we have $M(R)[a\rangle M(R')$ in the net system, when R is a reachable product state. This is the direction which uses consistency of matching which we get by using liveness of net.

Inductively we know from the isomorphism that $M(R)$ is a reachable marking and this extends the isomorphism to $M(R')$. Let $loc(a) = \{1, \ldots, l\}$ and $g = \langle \langle p_1, a, p'_1 \rangle, \ldots \langle p_l, a, p'_l \rangle \rangle$.

We proved above that A is consistent with constructed matching. Therefore, being a reachable state of A, R is in a-matching. Hence $\textsf{pre-places}(g)$ belong to a-matching. So by construction, these places belong to same cluster of net. By S-decomposability it also means that $\textsf{post-places}(g)$ are post-places of the same cluster. Since cluster is free choice, all transitions have same pre-places. By distributed choice, the post-decomposition $\langle p'_1, \ldots, p'_l \rangle$ is represented by one of the a-labelled transitions in the cluster. Firing this transition we have $M(R)\ [a\rangle\ M(R')$ in the net system and the isomorphism is inductively extended. Since the final markings of the net get related to the final product states, we get language equivalence of net and product system.

□

4.3.1 Free Choice Nets with the Unique Cluster Property to Product Systems with Separation of labels

If the net satisfies the unique cluster property, we get separation of labels and we do not need to use liveness in the proof.

Corollary 70. *Let $(N, M_0, \mathcal{G})$ be a 1-bounded, S-decomposable labelled distributed free choice net having the unique cluster property, and A the product system constructed at the beginning of Section 4.3. Then A is an FC-product with separation of labels, and $Lang(N, M_0, \mathcal{G}) = Lang(A)$.*

Proof. Let $N = (S, T, F, \lambda)$ be the underlying net. Let M be a reachable marking. Since A_i is a component, number of tokens in P_i remain constant, and we know that P_i had one token at M_0, so we always have one unique $r_i \in P_i$ such that $r_i = M \cap P_i$, for all i in Loc. So we get $R(M) = (r_1, r_2, \dots, r_k)$, which is a product state. In the reverse direction, for any product state R, taking union of all places in R gives us a unique set of places $M(R)$ which serves as a marking of net.

Since net is free choice, and using S-decomposability we get that A is FC-product. As N have unique cluster property and S-decomposable, constructed system A have separation of labels.

As in the proof of Theorem 69 we will show language equivalence by showing the stronger property that the maps $R(.)$ and $M(.)$ constitute an isomorphism of reachable markings of N with reachable product states of A. Clearly this is the case for the initial marking and the initial product state.

To prove $Lang(N, M_0, \mathcal{G}) \subseteq Lang(A)$, we show that for a transition $t \in T$ of the net system, labelled a, and $M[t\rangle M'$ in the net, then we have a global move g in the product system with label a yielding $R(M) \overset{a}{\Rightarrow} R(M')$ in the product system. We know that ${}^\bullet t \subseteq M$ and $t^\bullet \subseteq M'$. Hence for each $i \in loc(t)$ we have a place $r_i \in P_i$ such that $(r_i, t) \in F_i$. And since N_i corresponding to A_i is a component there exists another place $r'_i \in P_i$ such that $(t, r'_i) \in F_i$. But by construction we get a transition $\langle r_i, a, r'_i \rangle \in \rightarrow_i$, $\forall i \in loc(t)$. So by definition of a product system we get a global move $g = \Pi_{i \in loc(t)} \langle r_i, a, r'_i \rangle$ in constructed product system A. So we get ${}^\bullet t = \textsf{pre-places}(g)$ and $t^\bullet = \textsf{post-places}(g)$. So each $r_i \in \textsf{pre-places}(g)$ also belong to tuple $R(M)$ at i-th place. Repeating the same argument for M' we get that $r'_i \in \textsf{post-places}(g)$ also belong to tuple $R(M')$. So in the product system we have $R(M) \overset{a}{\Rightarrow} R(M')$.

In the reverse direction, to prove $Lang(A) \subseteq Lang(N, M_0, \mathcal{G})$, we show that, if $R \overset{a}{\Rightarrow} R'$ in the product system using a global move g then we have $M(R)[a\rangle M(R')$ in the net system, when R is a reachable product state.

Inductively we know from the isomorphism that $M(R)$ is a reachable marking and this extends the isomorphism to $M(R')$. Let $loc(a) = \{1, \ldots, l\}$ and $g = \langle \langle p_1, a, p'_1 \rangle, \ldots \langle p_l, a, p'_l \rangle \rangle$.

We proved above that A have separation of labels. By S-decomposability of net and by construction, these places belong to same cluster of net. By S-decomposability it also means that $\textsf{post-places}(g)$ are post-places of the same cluster. Since cluster is free choice, all transitions have same pre-places. By distributed choice, the post-decomposition $\langle p'_1, \ldots, p'_l \rangle$ is represented by one of the a-labelled transitions in the cluster. Firing this transition we have $M(R)$ $[a\rangle$ $M(R')$ in the net system and the isomorphism is inductively extended. Since the final markings of the net get related to the final product states, we get language equivalence of net and product system.

In the proof of Theorem 69, we used liveness to prove consistency of matching, which in turn was used to prove that pre-places of g belonged to the same cluster. But here we use unique cluster property to prove that. □

4.3.2 Acyclic Free Choice Nets to Connected-FC-expressions

Now we show that for a acyclic free choice net system with no active deadlocks, we get a product system having matching with which it is consistent.

Theorem 71. *Let $(N, M_0, \mathcal{G})$ be a 1-bounded, S-decomposable labelled distributed choice net system which is acyclic, and A is the product system constructed as in the beginning of Section 4.3. Then*

1. *If net is free choice then A is a FC-matching product and*
2. *in addition if the net system do not have active deadlocks, then*
   - *(a) all runs of A are consistent with the matching.*
   - *(b) $Lang(N, M_0, \mathcal{G}) = Lang(A)$.*

Proof. This proof is given on the similar lines of proof of Theorem 69. Let $N = (S, T, F, \lambda)$ be the underlying net. Let M be a reachable marking. Since A_i is a component, number of tokens in P_i remain constant, and we know that P_i had one token at M_0, so we always have one unique $r_i \in P_i$ such that $r_i = M \cap P_i$, for all i in Loc. So we get $R(M) = (r_1, r_2, \ldots, r_k)$, which is a product state. In the reverse direction, for any product state R, taking union of all places in R gives us a unique set of places $M(R)$ which serves as a marking of net.

1. Since net is free choice, we get a conflict-equivalent matching of labels by, taking tuples of pre-places of a cluster, making A an FC-matching product.
2. Assume that the net system do not have active deadlocks.
   - (a) Suppose we have two places, say p_1 in location 1 and p_2 in location 2 which are not matched on any action a, which means that they are not in the same cluster of the free choice net N. Again by construction, let the local move $p_1 \xrightarrow{a} p_1'$ be obtained from the net transition t_1 with pre-place p_1 coming from

cluster $C_1 = (S_1, T_1)$. By S-decomposability, C_1 has a matching pre-place q_2 in location 2. Similarly let $p_2 \overset{a}{\rightarrow} p'_2$ be obtained from t_2 with pre-place p_2 coming from cluster $C_2 = (S_2, T_2)$, with matching pre-place r_1 in location 1 using S-decomposability.

Since we started with a 1-bounded S-cover, initially each component had only one token in it, and in a component number of tokens remains constant at any reachable marking. So the places q_2 and r_1 are distinct from p_1 and q_2 respectively. Again using the 1-bounded S-cover, q_2 and r_1 have to be unmarked at the net marking $M(R)$. At this marking we fire as many transitions as possible. Since net is acyclic we can fire only finitely many transitions and we reach a marking M' where no transition can be fired. Since net N has no active deadlocks, some transition of C_1 will get fired and by free choice, for this we need to have tokens in all places of S_1. To bring the token in to q_2 we have to fire some transition in C_2, for which by free choice we need to put a token in place r_1, which has to come from p_1. In short, we have two places from where a transition can never be fired, but they do have a transition in their post also, this contradicts the fact that N do not have active deadlocks.

This means that all the places $p_1, \ldots, p_l$ are in the same cluster. As a consequence, the run of the product system at this marking (and inductively at all reachable markings) will be consistent with the matching of labels defined in the construction.

(b) Once we have consistency of matching, which we obtained by using the fact that acyclic net N do not have active deadlocks, proof of language equivalence of net and product system is same as the proof of language equivalence of Theorem 69.

□

Now we can get expressions for acyclic free choice net systems and live T-net systems of

Corollary 72. *Let* $(N, M_0, \mathcal{G})$ *be a* 1*-bounded, S-decomposable, acyclic, labelled distributed choice net system which do not have active deadlocks. Then there exists a connected-FC-expression for its language.*

Proof. Use Theorem 71, to get language equivalent product system, in which each sequential system is acyclic. Then apply Lemma 52 to get a language equivalent connected-FC-expression. □

Corollary 73. *Let* $(N, M_0, \mathcal{G})$ *be a* 1*-bounded, S-decomposable, live, labelled T-net system. Then there exists a language equivalent* ω*-T-expression for it.*

Proof. We know that a live T-net system is structurally cyclic. When a T-net system is S-decomposable, each sequential system is a cycle. Since a T-net has only one transition in each cluster, it satisfies distributed choice property trivially. Now applying Theorem 69, we get a language equivalent FC-product system, in which each sequential system is a cycle, hence a T-product system which is structurally cyclic. We use Theorem 53 to get language equivalent ω-T-expression for it. □

4.4 FC-matching Product Systems to Distributed Free Choice Nets

In this section we see that conflict-equivalent matchings are sufficient to obtain free choice nets from product systems.

In Definition 20 we saw a generic construction of a net system $(N = (S, T, F, \lambda), M_0, \mathcal{G})$ from a given product system $A = (A_1, A_2, \ldots, A_k)$, which we repeat here for convenience.

- $S = \cup_i P_i$, the set of places.

- $T = \cup_a T_a$, where T_a is $\Rightarrow_a$, the set of a-labelled global moves.

- The labelling function λ labels by a by the transitions in T_a.

- The flow relation $F = \{(p, g), (g, q) \mid g \in T_a, g[i] = \langle p, a, q\rangle, i \in loc(a)\}$.

- $M_0 = \{p_1^0, \ldots, p_k^0\}$, the initial product state.

- $\mathcal{G} = G$, the set of final product states.

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 distributed free choice net.

If we remove nonreachable transitions from the constructed net, its language remains the same, moreover, the net becomes free choice when the product system has conflict-equivalent matchings and all its runs are consistent with it. Free choice net shown in Figure 4.5 is obtained from net of Figure 1.6 by pruning out transitions which are never firable.

Theorem 74. *Let $(N, M_0, \mathcal{G})$ be the net system constructed from product system A as in the construction given in beginning of Section 4.4. Then*

1. *N is a S-decomposable net.*

2. *N satisfies distributed choice property.*

3. *$Lang(N, M_0, \mathcal{G}) = Lang(A)$.*

4. *Further, if A is FC-matching product and all runs of A are consistent with the given matching of labels, then we can choose $T' \subseteq T$ such that the subnet N' generated by T' is a free choice net and $(N', M_0, \mathcal{G})$ accepts the same language.*

5. *Further, if the product system was deterministic for global actions, the net is deterministic for synchronization.*

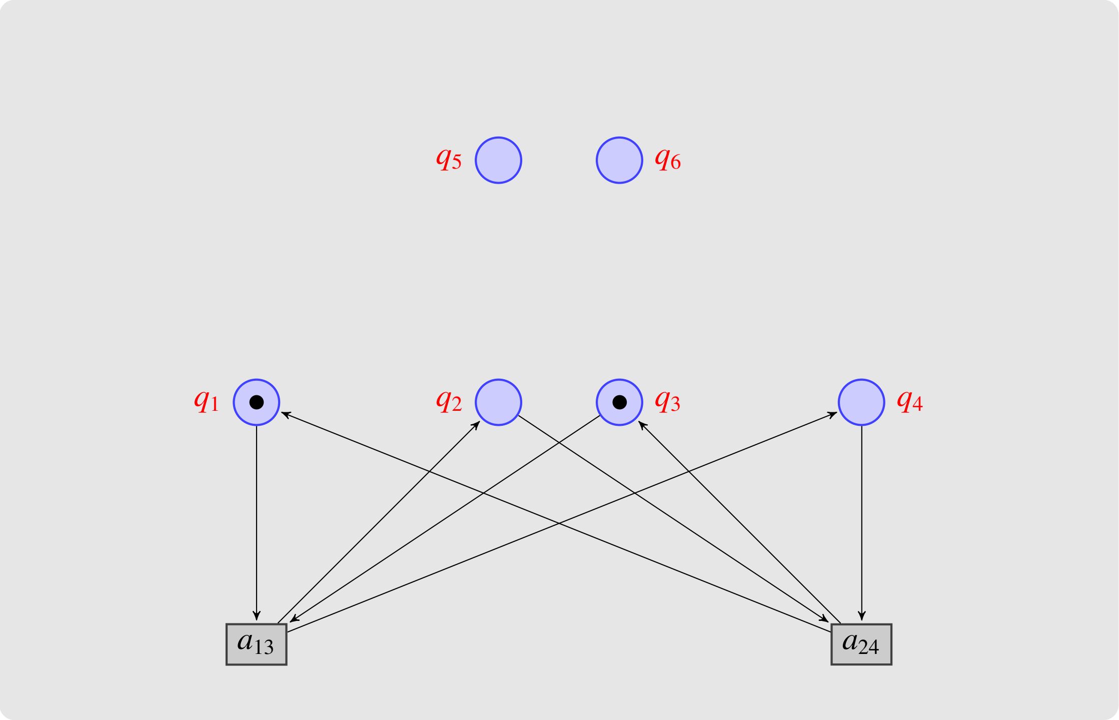


Figure 4.5: Free Choice Net obtained from Net of Figure 1.6

Proof. 1. That N is S-decomposable follows from the fact that we can build components $N_i = (S_i, T_i, F_i)$ from product system A. Take $S_i = P_i$ and $T_i = \{\lambda^{-1}(a) \mid a \in \Sigma_i\}$ by definition. The flow relation F can be written as the union of, $F_i = \{(p, g), (g, q) \mid g[i] = \langle p, a, q \rangle\}$.

2. Now we want to prove that N satisfies distributed choice. Consider an a-labelled transition t in a cluster C of N'. Because of S-decomposability it has exactly one pre-place and one post-place from each location of a. Given a pre-place p and post-place q in A_i there can be only one local move $\langle p, a, q \rangle \in \rightarrow_i^a$. So for a fixed pre-place p, any local move on action a is uniquely identified by its post-place. Therefore the post-places of t uniquely identify a global move in A or transition of net.

3. Now we prove that $Lang(N, M_0, \mathcal{G}) = Lang(A)$. In the construction, the initial marking is $M_0 = \{p_1^0, \ldots, p_k^0\}$ where p_i^0 is the initial place of the sequential system A_i in the product. Inductively, for any product state R of the product system we can associate a unique marking $M(R)$. The set of transitions T of the constructed net is

the set of global moves of the product system, and since the places of the net are the set obtained by taking union of places of all sequential systems of the product, we have that $^\bullet g = \textsf{pre-places}(g)$ and $g^\bullet = \textsf{post-places}(g)$. So if there is a product state R' obtained by taking global move g having $\lambda(g) = a$ at product state R, then we get $M(R)[a\rangle M(R')$ in the net system produced.

On the other hand, from initial marking M_0 of net system, we can construct the initial product state R^0 of the product system, by taking its intersection with the places of a sequential system. So, $\{p_i^0\} = M_0 \cap P_i$, for all $i \in Loc$. Since the result of intersection is a singleton set, we write this with abuse of notation as $p_i^0 = M_0 \cap P_i$. Inductively, we can associate a reachable product state with any given reachable marking of the net, as in the component corresponding to location i, there can be exactly one place which is marked.

Consider a transition t in the net system. if $p \in P_i \cap {}^\bullet t$ then after firing this transition t, token from place p is circulated back in the P_i for some place $q \in P_i \cap t^\bullet$. So if we have $M[a\rangle M'$ in the net system then, we get $R(M) \overset{a}{\Rightarrow} R(M')$ in the product system, where $R(M)$ and $R(M')$ are the product states corresponding to the markings M and M' respectively, of net system.

This establishes an isomorphism between the set of reachable product states of the product system and the set of reachable markings of net. Because the initial marking and the final markings correspond to the initial product state and the final product states we get language equivalence of net and product system.

4. Now we assume that all runs of the product system are consistent with a conflict-equivalent matching of labels. Our choice of the subset of transitions $T' \subseteq T$ is to keep those whose pre-places are part of reachable product states. This does not violate S-decomposability or language equivalence.

   We want to prove that N' is a free choice net. Let $C = (S_C, T_C)$ be a cluster of constructed net N'. We have to prove that it is a free choice cluster. If $|S_C| = 1$ or

$|T_C| = 1$ then C is trivially a FC-cluster. So we consider the case where $|S_C| > 1$ and $|T_C| > 1$. Let $p, q \in S_C$ and $t_1, t_2 \in T_C$, such that $p \in {}^\bullet t_1 \cap {}^\bullet t_2$ and $q \in {}^\bullet t_1$. We have to prove that $q \in {}^\bullet t_2$.

Consider the case where, $\lambda(t_1) = \lambda(t_2) = a$. We know that $t_1 \in p^\bullet \cap q^\bullet$ and we have above proved that N' is S-decomposable net system, so places p and q do not belong to same component of net. Without loss of generality, assume that p is in component N_i and q is in component N_j with $i \neq j$. So transition $t_1 \in T_i \cap T_j$ implying $|loc(a)| > 1$. Since p, q were part of a reachable product state and runs of A are consistent with matching of labels, pre-place p must have the matching pre-place q. Since t_2 is in $p^\bullet$, because of consistency with matching of labels, all global moves on action a with p must use q. Thus $q \in {}^\bullet t_2$ as required.

Now consider the case where, $\lambda(t_1) = a$ and $\lambda(t_2) = b$. Since $|{}^\bullet t_1| > 1$, by Proposition 17 we have $|loc(a)| > 1$. Net N' is S-decomposable, so places p and q do not belong to same component of net. Without loss of generality, assume that p is in component N_i and q is in component N_j with $i \neq j$. By construction there exist local moves $\langle p, a, p' \rangle, \langle p, b, p'' \rangle \in \rightarrow_i$ and $\langle q, a, q' \rangle \in \rightarrow_j$. As the matching is conflict-equivalent, local moves $\langle p, a, p' \rangle$ and $\langle q, a, q' \rangle$ are conflict-equivalent, implying existence of a local move $\langle q, b, q' \rangle \in \rightarrow_j$. By Proposition 62 we get $loc(a) = loc(b)$.

Since runs of product system A are consistent with the matching of labels, when p has outgoing moves on action a and b, they will match with outgoing moves from q in A_j. Since the transitions of N' come from the global moves of the product system, transition t_2, which is labelled b, has same set of pre-places as t_1 labelled a, implying $q \in {}^\bullet t_2$ as required.

5. Since set of global moves of product system is used to construct set of transitions of the net, and reachable state spaces of both net and product system are isomorphic as proved above, then the determinism of global actions in the product system gives rise to a net which is deterministic for synchronization.

□

Corollary 75. *Let* $(N = (S, T, F, \lambda), M_0, \mathcal{G})$ *be the net system constructed from product system A, as at the beginning of Section 4.4. If A is an FC-product with the separation of labels property then N is a distributed free choice net with the unique cluster property.*

Proof. Since A satisfies separation of labels property by construction, we get the unique cluster property straightaway, and as A have conflict-equivalence by construction, we get that the net is free choice.

In the proof of Theorem 74, we pruned transitions whose pre-places belonged to different clusters, meaning these transitions corresponded to global moves, whose pre-states did not belong to matching. This is not required here, as all global moves labelled by some global action a, have same set of pre-places, as A satisfies separation of labels property. □

4.5 Conclusion

In Chapter 3, it has been shown that a graph-theoretic condition called "structural cyclicity" enables us to extract syntax from a conflict-equivalent product system. In the present work we explore the connection between free choice nets and product systems which have conflict-equivalence and other properties. In particular we have a broader class of product systems, where the conflict-equivalence is not statically fixed.

While the existence of a conflict-equivalent matching is sufficient to construct free choice nets, it is not necessary. Consider the example product system in Figure 4.6 below.

For the product system given in Figure 4.6 we get a net shown in the Figure 4.7 with initial marking $M_0 = \{(1, 3)\}$, and set of final markings $\mathcal{G} = \{(1, 3)\}$. This net is not free choice although it is language equivalent to the product system from which it was constructed.

But a free choice net for this example product system is obtained by unfolding the second

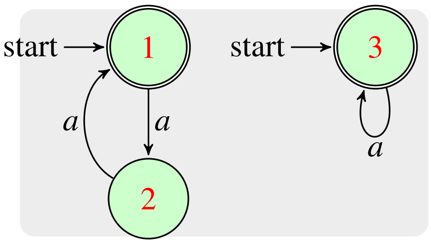


Figure 4.6: Direct product system with language $L = \{aa\}^*$

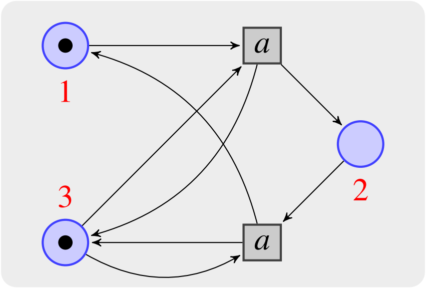


Figure 4.7: Non free choice net

sequential system to obtain a matching of labels.

Chapter 5

Product Systems with Matchings and Product Expressions with Pairings

In this chapter we give product expressions for product systems with matchings. So using the results from earlier Chapter 4, we get expressions for free choice nets with unique cluster property and without it.

5.1 Properties of Product Expressions

First we define properties required to be satisfied expressions, corresponds to unique cluster property of nets defined in Chapter 4.

Definition 76 (unique sites)**.** *If for all global actions a occurring in s, the partition* $\mathrm{Der}_a(s)$ *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.*

A product expression $e = fsync(r_1, r_2, \ldots, r_k)$ *is said to have unique sites if each* r_i *have unique sites property.*

Definition 77 (equal choice for set of derivatives)**.** *Let $e = fsync(s_1, s_2, \dots, s_k)$ be a product expression over Σ. Let D be a set of derivatives of s_i i.e., $D \subseteq \mathcal{PD}(s_i)$ and let D' be a set of derivatives of s_j i.e., $D' \subseteq \mathcal{PD}(s_j)$. Then D is said to be in equal choice with D' if $Init(D) = Init(D')$.*

We now define some properties of product expressions over a distribution. These will ultimately lead us to construct free choice nets.

Definition 78 (pairing, equal choice pairing)**.** *Let $e = fsync(s_1, s_2, \dots, s_k)$ be a product expression over Σ. For a global action a, an a-**pairing** is a subset of tuples $\Pi_{i \in loc(a)}\mathrm{Der}_a(s_i)$, the projections of these tuples covering the a-sites in s_i, such that if a block of $\mathrm{Der}_a(s_j)$, $j \in loc(a)$ 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 all blocks of derivatives in the tuple are in equal choice.*

We extend the definition to product expressions. A derivative $fsync(r_1, \dots, r_k)$ 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 $pairing(a)$.

Definition 79 (expression with pairing, expression with equal choice pairing)**.** *Expression e is said to have **pairing of actions** if for all global actions a, there exists an $pairing(a)$. Expression e is said to have **equal choice pairing of actions** if for all global actions a, there exists an equal choice $pairing(a)$.*

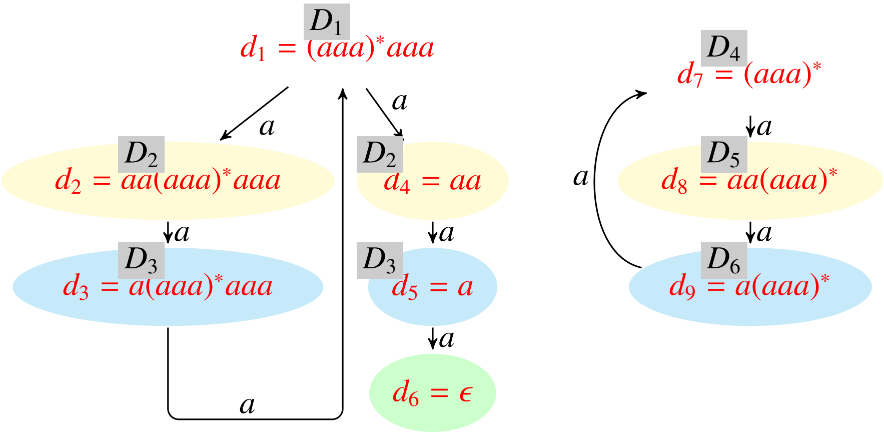
Definition 80 (consistency with pairing)**.** *Expression e with pairing of actions, is said to be **consistent with a pairing of actions** if every reachable a-site $d \in Der(e)$ is in $pairing(a)$.*

Example 81. *Consider a distribution $\Sigma_1 = \Sigma_2 = \{a\}$ and a product expression $\mathrm{fsync}(aa, a)$ defined over it. The partition for aa over Σ_1 is $\mathrm{Part}_a(aa) = \{\{aa\}, \{a\}\}$ and for the expression a over Σ_2 is $\mathrm{Part}_a(a) = \{\{a\}\}$. Since two blocks of $\mathrm{Part}_a(aa)$ cannot be paired with* *one block of* $\mathrm{Part}_a(a)$*, expression* $\mathrm{fsync}(aa, a)$ *does not have a pairing. Since there are two blocks in the partition* $\mathrm{Part}_a(aa)$*, expression aa does not have unique sites property, neither does* $\mathrm{fsync}(aa, a)$*.*

Example 82. *Consider a product expression* $e = \mathrm{fsync}(aa, bad+caf)$ *over the distribution* $(\Sigma_1 = \{a\}, \Sigma_2 = \{a, b, c, d, f\})$*. For action a, The partition over* Σ_1 *is* $\mathrm{Part}_a(aa) = \{\{aa\}, \{a\}\}$ *and the partition over* Σ_2 *is* $\mathrm{Part}_a(bad + caf) = \{\{ad\}, \{af\}\}$*. The a-sites of expression e are* $\{\mathrm{fsync}(aa, ad), \mathrm{fsync}(aa, af)\}$*. There are two possible pairings for action a: one is* $\{(\{aa\}, \{af\}), (\{a\}, \{ad\})\}$ *and another is* $\{(\{aa\}, \{ad\}), (\{a\}, \{af\})\}$*. The derivative aa on the left appears in the pairing with two different reachable a-sites of the right hand side, which belong to two different blocks of* $\mathrm{Part}_a(bad + caf)$*. Hence e is not consistent with respect to any of the above pairings.*

Example 83. *Consider the product expression* $\mathrm{fsync}(r_1, r_2, r_3)$ *with* $r_1 = (ac)^*$*,* $r_2 = (bc)^*$ *and* $r_3 = (a(b + c))^*$ *over the distribution* $(\Sigma_1 = \{a, c\}, \Sigma_2 = \{b, c\}, \Sigma_3 = \{a, b, c\})$*. Now we have* $r'_1 = \mathrm{Der}_a(r_1) = c(ac)^*$ *and* $\mathrm{Init}(r'_1) = \{c\}$*. For* r_3 *we have,* $r'_3 = \mathrm{Der}_a(r_3) = (b + c)(a(b + c))^*$ *and* $\mathrm{Init}(r'_3) = \{b, c\}$*. Expressions* r'_1 *and* r'_3 *are c-sites of expressions* r_1 *and* r_3 *respectively. With sets of derivatives* $D_1 = \{r'_1\}$ *and* $D_3 = \{r'_3\}$ *as the only blocks in the respective partitions of c-sites i.e.,* $\mathrm{Part}_c(r_1) = \{D_1\}$ *and* $\mathrm{Part}_c(r_3) = \{D_3\}$*. As* $\mathrm{Init}(D_1) = \mathrm{Init}(r'_1) = \{c\}$*,* $\mathrm{Init}(D_3) = \mathrm{Init}(r'_3) = \{b, c\}$ *and* $\mathrm{pairing}(c) = \{(D_1, D_3)\}$*,* $\mathrm{pairing}(c)$ *is not equal choice. Therefore, product expression e does not have equal choice. However one can see that e has unique sites property.*

Example 84. *Consider a product expression* $e = \mathrm{fsync}((aaa)^*aaa, (aaa)^*)$*. The a-derivatives are* $\mathrm{Der}_a(e) = \{\mathrm{fsync}(aa(aaa)^*aaa, aa(aaa)^*), \mathrm{fsync}(aa, aa(aaa)^*)\}$*. With respect to word aa,* $\mathrm{Der}_{aa}(e) = \{\mathrm{fsync}(a(aaa)^*aaa, a(aaa)^*), \mathrm{fsync}(a, a(aaa)^*)\}$*. With respect to word aaa,* $\mathrm{Der}_{aaa}(e) = \{\mathrm{fsync}((aaa)^*aaa, (aaa)^*), \mathrm{fsync}(\epsilon, (aaa)^*)\}$*. The language of product expression e is* $\mathrm{Lang}(e) = \{(aaa)^k \mid k \geq 1\}$*. See Figure 5.1 where derivatives of* $d_1 = (aaa)^*aaa$ *and* $d_7 = (aaa)^*$ *are shown. the set of derivatives of* $e = \mathrm{fsync}(d_1, d_7)$*, with respect to all words* $w \in \Sigma^*$*:* $\mathrm{Der}(e) = \{(d_1, d_7), (d_2, d_8), (d_4, d_8), (d_3, d_9), (d_5, d_9), (d_6, d_7)\}$ *and, its set of*



$\mathsf{Der}((aaa)^*aaa)$ $\mathsf{Der}((aaa)^*)$

Figure 5.1: Derivatives of d_1 and d_7 of expression $e = \mathit{fsync}(d_1, d_7)$ with $\mathit{pairing}(a) = \{(D_1, D_4), (D_2, D_5), (D_3, D_6)\}$.

a-sites is $\{(d_1, d_7), (d_2, d_8), (d_4, d_8), (d_3, d_9), (d_5, d_9)\}$.

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

Proposition 85. *For a product 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. □

5.2 Properties of Product Systems

We extend the definitions of relativized languages and bifurcations from Chapter 2 to places and product states of a product system.

Definition 86. *Given a place p of A_i, we define relativized languages $Pref_a^p(L) = \{x \mid xay \in L, p_0 \xrightarrow{x} p \xrightarrow{ay} G_i\}$, similarly $Suf_a^p(L)$. Let $L^p = \{xay \mid xay \in L, p_0 \xrightarrow{x} p \xrightarrow{ay} G_i\}$.*

Given a set of places D of A_i, we define relativized languages $Pref_a^D(L) = \{x \mid xay \in L$, and $\exists p \in D$ such that $p_0 \xrightarrow{x} p$, similarly $Suf_a^D(L)$. Let $L^D = \{xay \mid xay \in L$ and $\exists p \in D$ such that $x \in Pref_a^p(L)$ and $y \in Suf_a^p(L)\}$.

*For sequential sysetm A_i, a place p a-**bifurcates** L_i if $L_i^p = Pref_a^p(L_i)\ a\ Suf_a^p(L_i)$.*

*For sequential sysetm A_i, a set of places D a-**bifurcates** L_i if $L_i^D = Pref_a^D(L_i)\ a\ Suf_a^D(L_i)$.*

While a place with outgoing a-moves, always a-bifurcates language of automaton, a set of places may not.

Definition 87. *Given a global state r of A, we define relativized languages $Pref_a^r(L) = \{x \mid xay \in L, p^0 \xrightarrow{x} r \xrightarrow{ay} G_i\}$, similarly $Suf_a^r(L)$. Let $L^r = \{xay \mid xay \in L, p_0 \xrightarrow{x} r \xrightarrow{ay} G_i\}$.*

Given a set of global states R of A, we define relativized languages $Pref_a^R(L) = \{x \mid xay \in L$, and $\exists p \in R$ such that $p_0 \xrightarrow{x} p$, similarly $Suf_a^R(L)$. Let $L^R = \{xay \mid xay \in L$ and $\exists p \in R$ such that $x \in Pref_a^p(L)$ and $y \in Suf_a^R(L)\}$.

*For product system A, a global state r a-**bifurcates** L if $L^r = Pref_a^r(L)\ a\ Suf_a^r(L)$.*

*For product sysetm A, a set of places R a-**bifurcates** L if $L^R = Pref_a^R(L)\ a\ Suf_a^R(L)$.*

As in the case of sequential systems, a global state which enables global a-move, always a-bifurcates language of product system, a set of global product states may not.

Proposition 88. *Let $A = (A_1, \ldots, A_k)$ 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. □

5.3 Synthesis of Product Systems with Matchings from Expressions with Pairings

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

Lemma 89. *Let e be a product expression with partitions which give unique sites (for every global action). 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 FC-product.*

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 place 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-moves. 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 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 $d_j \xrightarrow{ax_{j+1}} \ldots \xrightarrow{ax_n} d_n$ of A_i when $j > 0$, and $d_0 \xrightarrow{x_1}\xrightarrow{ax_2} \ldots \xrightarrow{ax_n} d_n$ when $j = 0$, which will show the desired inclusion.

We proceed from n downwards. For any place d_n in G there is a run from d_n on $\epsilon \in Lang(d_n)$ in A_i. Inductively assume we have d_j such that there is a run $d_j \xrightarrow{ax_{j+1}} \ldots \xrightarrow{ax_n} d_n$ of A_i, so $x_{j+1} a x_{j+2} \ldots a x_n$ is in $Suf_a(Lang(s_i))$ since d_j is reachable from the initial place. Since there is a run $p \xrightarrow{ax_j} p$ in A_i' there are derivatives d_{j-1}, c_j of s_j, such that there is a run $d_{j-1} \xrightarrow{ax_j} c_j$ in A_i (when $j = 1$ we get $d_0 \xrightarrow{x_1} c_1$ by this argument). Since c_j quotients to p, it has an a-derivative c such that c is in $Der_{ax_j a}(d_{j-1})$ ($Der_{x_0 a}(d_0)$ when $j = 1$). Because d_{j-1} is reachable from the initial place by some v and because some final place is reachable from c, $vx_j \in Pref_a(Lang(s_i))$ which is nonempty. By the unique sites condition and Proposition 31, since $x_{j+1} \ldots a x_n$ is in $Suf_a(Lang(s_i))$, $vax_j a x_{j+1} \ldots a x_n$ is in $Lang(s_i)$ and so $x_j a x_{j+1} \ldots a x_n$ is in $Suf_a(Lang(s_i))$. This means that there is a run from some d_{j-1} on $ax_j a x_{j+1} \ldots a x_n$ ending in a final place d_n of A_i. So we have the induction hypothesis restored. If $j = 1$ we get d_0 which quotients to p_0 and has a run on w to d_n in G.

So we get a product system $A' = \langle A_1', A_2', \ldots, A_k' \rangle$ defined over Σ. Because of the quotienting A' has separation of labels. That means for a global action a, for i, j in $loc(a)$, sequential machines A_i', A_j' has only one place which has outgoing local a-moves. Let p_i^a be that place in A_i' and let p_j^a be that place in A_j'. On the other hand, since e had unique sites, for a global action a and for i, j in $loc(a)$, expression s_i has only one block D_i in the partition of a-sites of s_i and expression s_j has only one block D_j in the partition of a-sites of s_j. Therefore, all a-sites of s_i are in this block D_i, and all a-sites of s_j are in block D_j. Therefore $\mathit{pairing}(a)$ has only one tuple which have D_i and D_j appearing in it. Since e has equal choice property, we have $\mathit{Init}(D_i) = \mathit{Init}(D_j)$. Because of quotienting construction, block D_i corresponds to the place p_i^a in A_i' and block D_j corresponds to the place p_j^a in A_j'. So each outgoing local a-move of p_i^a is conflict-equivalent to each outgoing local a-move of place p_j^a.

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

$$\begin{aligned} w \in \mathit{Lang}(e) &\text{ iff } \forall i, w{\downarrow_{\Sigma_i}} \in \mathit{Lang}(s_i), \text{ by definition of synchronized shuffle} \\ &\text{ iff } \forall i, w{\downarrow_{\Sigma_i}} \in \mathit{Lang}(A_i') \\ &\text{ iff } w \in \mathit{Lang}(A'), \text{ by Proposition 14.} \end{aligned}$$

□

Theorem 90. *Let $e = \mathit{fsync}(s_1, \ldots, s_k)$ be a product expression over a distribution Σ with a pairing of actions. Then there exists an product system with a matching of labels A over Σ, accepting $\mathit{Lang}(e)$. If the pairing was equal choice, the matching is conflict-equivalent. If the expression is consistent with the pairing, all runs of A will be consistent with the matching.*

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

Consider global action a and tuple of blocks $D = \Pi_{i \in loc(a)} D_i in pairing(a)$. By Proposition 31 D_i a-bifurcates $\mathit{Lang}(s_i)$. We rename for all i in $loc(a)$, the occurrences of a in s_i which correspond to an a in $\mathit{Init}(D_i)$, by the new letter a^{D_i}. This is done for all global actions to obtain from e a new expression $e' = \mathit{fsync}(s_1', \ldots, s_k')$ over a distribution Σ', where every

s'_i now has the unique sites property. For any word $w \in \mathit{Lang}(e)$, there is a well-defined word $w' \in \mathit{Lang}(e')$.

By Lemma 89 we obtain a product system A' with separation of labels for $\mathit{Lang}(e')$. Say $p(a^D)$ is the pre-place for action a^D in A'_i. We change all the $\langle p(a^D), a^D, q\rangle$ moves to $\langle p(a^D), a, q\rangle$ in all the A'_i to obtain a product system A over the alphabet Σ. As $w' \in \mathit{Lang}(e') = \mathit{Lang}(A')$ is well-defined from w and, as the renaming of labels of moves does not remove any paths, w is in $\mathit{Lang}(A)$. Conversely, for every run on w accepted by A, because of the separation of labels property, there is a well-defined run on w' with the label of a move appropriately renamed depending on the source state, which is accepted by A', hence w' is in $\mathit{Lang}(e')$. So renaming w' to w gives a word in $\mathit{Lang}(e)$.

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

If the expression e is consistent with the pairing, all reachable a-sites are in the pairing, so we can partition $\mathit{Lang}(e) \cap \Sigma^* a \Sigma^*$ using the partitions in $\mathit{Part}_a(e)$. Letting D range over blocks of product 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 89, we have the product system A' with separation of labels. By Proposition 88, every $\mathit{Lang}(A'_i)$ is a^D-bifurcated, and using the characterization of Proposition 14, $\mathit{Lang}(A') \cap (\Sigma')^* a^D (\Sigma')^* = \mathit{Pref}_{a^D}(\mathit{Lang}(A')) a^D \mathit{Suf}_{a^D}(\mathit{Lang}(A'))$. Since several actions a^D are renamed to a and the corresponding tuples of pre-places are recorded in the matching, by Proposition 88 and Proposition 14:

$$\bigcup_{R \in \mathit{matching}(a)} \mathit{Pref}^R_a(\mathit{Lang}(A))\ a\ \mathit{Suf}^R_a(\mathit{Lang}(A)) \subseteq \mathit{Lang}(A) \cap \Sigma^* a \Sigma^*.$$

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

As an illustration of constructing product system with matching from expression with pairing, using Theorem 90 which employs Lemma 89 in its proof, consider the expression in Example 84, for which we produce a product system as was shown in Example 91.

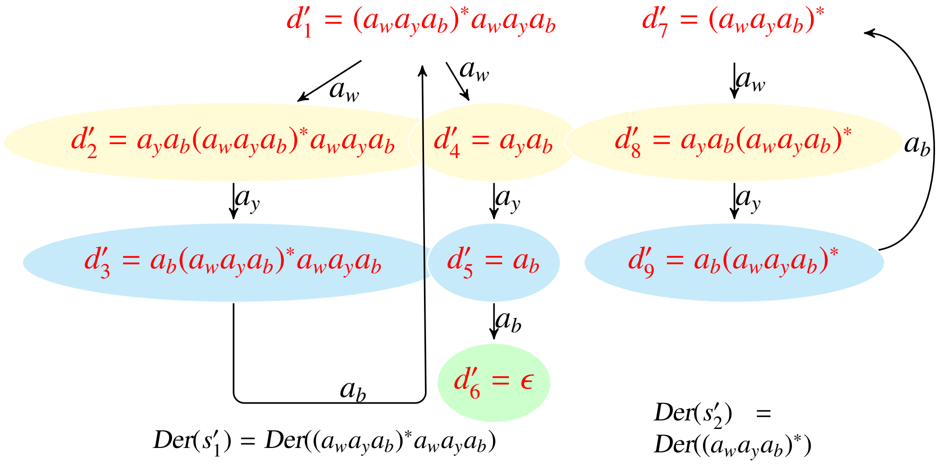

$d'_1 = (a_w a_y a_b)^* a_w a_y a_b$
$d'_7 = (a_w a_y a_b)^*$
a_w
a_w
a_w
$d'_2 = a_y a_b (a_w a_y a_b)^* a_w a_y a_b$
$d'_4 = a_y a_b$
$d'_8 = a_y a_b (a_w a_y a_b)^*$
a_b
a_y
a_y
a_y
$d'_3 = a_b (a_w a_y a_b)^* a_w a_y a_b$
$d'_5 = a_b$
$d'_9 = a_b (a_w a_y a_b)^*$
a_b
$d'_6 = \epsilon$
a_b
$\mathrm{Der}(s'_1) = \mathrm{Der}((a_w a_y a_b)^* a_w a_y a_b)$
$\mathrm{Der}(s'_2) = \mathrm{Der}((a_w a_y a_b)^*)$

Figure 5.2: Derivatives of s'_1 and s'_2 of $e' = fsync(s'_1, s'_2)$ with unique sites property

Example 91. *As we have seen in Example 84, the pairing relation for expression* $e = fsync((aaa)^*aaa, (aaa)^*))$*,* $\mathrm{pairing}(a) = \{(D_1, D_4), (D_2, D_5), (D_3, D_6)\}$*. Let* $w = (D_1, D_4), y = (D_2, D_5)$ *and* $b = (D_3, D_6)$*.*

Then using these tuples, we get a new alphabet $\Sigma' = \{a_w, a_y, a_b\}$ *with distribution* $\Sigma'_1 = \{a_w, a_y, a_b\}$ *and* $\Sigma'_2 = \{a_w, a_y, a_b\}$*. Each a in* s_i *belong to only one block in* $\mathrm{Part}_a(s_i)$ *and that block belong to only one tuple in the* $\mathrm{pairing}(a)$*. Therefore, by renaming each a in* s_i *by its corresponding tuple in* $\mathrm{pairing}(a)$*, we get* $s'_1 = (a_w a_y a_b)^* a_w a_y a_b$ *and* $s'_2 = (a_w a_y a_b)^* a_w a_r a_b$ *over alphabet* Σ'_1 *and* Σ'_2 *respectively. Hence, we have a product expression* e' *over* Σ' *as,* $e' = fsync((a_w a_y a_b)^* a_w a_y a_b, (a_w a_y a_b)^*)$*.*

Expressions s'_1 *and* s'_2 *have unique sites property. In Figure 5.2, derivatives of* s'_1 *and* s'_2 *are shown. The blocks in the partitions of their respective* a_x*-sites, where* $x \in \{w, y, b\}$ *are:* $D'_1 = \{d'_1\}, D'_2 = \{d'_2, d'_4\}, D'_3 = \{d'_3, d'_5\}, D'_4 = \{d'_7\}, D'_5 = \{d'_5\}, D'_6 = \{d'_6\}$*. Now by Lemma 89 we can fuse derivatives in the respective blocks to get product system*

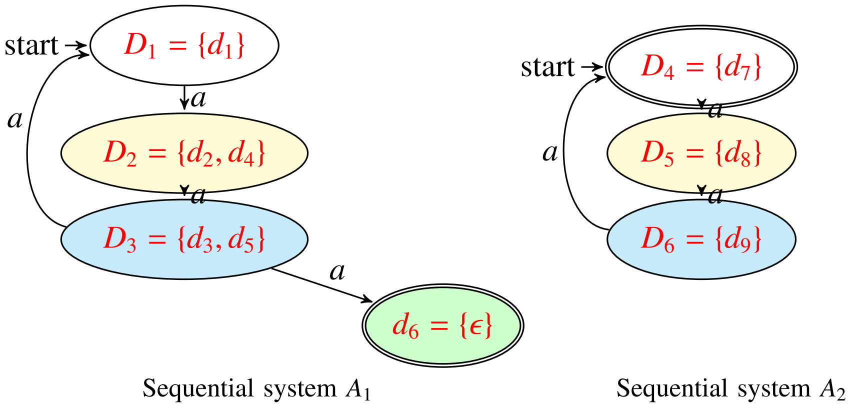


Figure 5.3: Product system $A = (A_1, A_2)$ with separation of labels

$A' = (A'_1, A'_2)$, having separation of labels property, and which is language equivalent to expression e'. The set of places of sequential system A'_1, is $\{D'_1, D'_2, D'_3, d'_6\}$, and of sequential system A'_2, is $\{D'_4, D'_5, D'_6\}$. In each A'_i we have only one place which has outgoing a_x-moves. So each a_x contributes only one tuple of places in matching(a). *Therefore,* matching$(a) = \{(D_1, D_4), (D_2, D_5), (D_3, D_6)\}$. *The final product system over Σ is shown in Figure 5.3.*

5.4 Analysis of Expressions with Pairings from Product Systems with Matchings

Lemma 92. *Let A be a conflict-equivalent product system with separation of labels. Then we can compute a product expression for the language of A with partitions of the regular expressions which have unique sites and specified pairings which have equal choice.*

Proof. Let $A = \langle A_1, \ldots, A_k \rangle$ be a product system with separation of labels, where A_i is a sequential system of A with places P, initial place p_0 and final places G. Kleene's theorem gives us expressions for the words which have runs from a given state to another using a specified set of states [MY60] and these are put together. Let us suppose that all the states which do not have any global actions enabled are dealt with first. After that we add the states with global actions, we do an induction on the number of these states.

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

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

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

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

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

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

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

By induction the right hand expressions had the unique c-sites property, the c-partition collapses all the derivatives above into a single block. We claim the derivatives in this four-way union c-bifurcate the language $Lang(e^R_{p_0,f})$. If the state q was visited in only one of the four cases there is nothing to prove. The interesting case is when there is a path from p to q as well as from q to p, and separate paths from p_0 to p and from p_0 to q. In this case the second and the third components of the union will both be nonempty. Suppose $w_1 = x_1cy_1$ with $x_1 \in \mathit{Lang}(e^Q_{p_0,q})$ and $cy_1 \in \mathit{Lang}(e^T_{q,p}(e^Q_{p,p})^*e^Q_{p,f})$, and $w_2 = x_2cy_2$ with $x_2 \in \mathit{Lang}(e^Q_{p_0,p}e^T_{p,q})$ and $cy_2 \in \mathit{Lang}(e^T_{q,p}(e^Q_{p,p})^*e^Q_{p,f})$. But then x_1cy_2 is in $\mathit{Lang}(e^Q_{p_0,q}e^T_{q,p}(e^Q_{p,p})^*e^Q_{p,f})$ and hence in $\mathit{Lang}(e^R_{p_0,f})$. Similarly word x_2cy_1 is in the language $\mathit{Lang}(e^Q_{p_0,p}e^T_{p,q}e^T_{q,p}(e^Q_{p,p})^*e^Q_{p,f})$ and also in $\mathit{Lang}(e^R_{p_0,f})$. In both cases the same derivatives, giving the language for the expression $e^T_{q,p}(e^Q_{p,p})^*e^Q_{p,f}$, appear in the set D^c. By equal choice, this argument extends if other global actions are also enabled along with c. □

Theorem 93. *Let A be a product system with a conflict-equivalent matching. Then we can compute a product expression for the language of A, having an equal choice pairing of actions.*

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

We construct a new product system A' where, for the places in the j'th tuple of the $\mathit{matching}(a)$, we change the label of the outgoing a-moves to a^j; similarly for the places in tuples of the $\mathit{matching}(b)$; and so on. We now have a new product system where the letter a of the alphabet has been replaced by the set $\{a^1, \ldots, a^n\}$; the letter b has been replaced by another set; and so on, obtaining a new distribution 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 92.

From the Lemma 92 we get a product expression $e' = \mathit{fsync}(s_1, \ldots, s_k)$ for the language of A' over Σ' where every regular expression has unique sites. From the proof of the Lemma 92 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 places 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 moves depends on the source place, 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 14 in our hands, the Lemma 92 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 $matching(a)$, the product states in $pre(a)$ would all be in the $matching(a)$, and we obtain that the expression e is consistent with the $pairing(a)$.

□

Example 94. *Let Σ be a distributed alphabet and $(\Sigma_1 = \{a\}, \Sigma_2 = \{a\})$ be a distribution of Σ. Consider a product system $A = (A_1, A_2)$ with matching, defined over Σ, as shown in Figure 5.3. A matching relation for global action a is: $matching(a) = \{(D_1, D_4), (D_2, D_5), (D_3, D_6)\}$.*

Let $w = (D_1, D_4), y = (D_2, D_5)$ and $b = (D_3, D_6)$. Hence, we have new alphabet $\Sigma' =$

$\{a_w, a_y, a_b\}$ with distribution $\Sigma'_1 = \{a_w, a_y, a_b\}$ and $\Sigma'_2 = \{a_w, a_y, a_b\}$. We now have a new product system $A' = (A'_1, A'_2)$ in which each action labelled a of has been replaced by an action from $\{a_w, a_y, a_b\}$; Again by the definition of matching, the new system A' has separation of labels. Hence we can apply Lemma 92, to get a product expression $e' = \mathsf{fsync}((a_wa_ya_b)^*a_wa_ya_b, (a_wa_ya_b)^*)$ defined over Σ', language equivalent to A' and have unique sites. Derivatives for $s'_1 = (a_wa_ya_b)^*a_wa_ya_b$ and $s'_2 = (a_wa_ya_b)^*a_wa_ra_b$ are shown in the Figure 5.2. Since e' has unique actions, for action a_w, there is only one block in the partitions of a_w-sites of s'_1 and s'_2: $\mathsf{Part}_{a_w}(s'_1)$ and $\mathsf{Part}_{a_w}(s'_2)$, and for remaining global actions a_y, a_b also. For action a_w partition set is: $\mathsf{Part}_{a_w}(s'_1) = \{D'_1\}$, $\mathsf{Part}_{a_w}(s'_2) = \{D'_4\}$, for action a_y: $\mathsf{Part}_{a_y}(s'_1) = \{D'_2\}$, $\mathsf{Part}_{a_y}(s'_2) = \{D'_5\}$, and, for action a_b: $\mathsf{Part}_{a_b}(s'_1) = \{D'_3\}$, $\mathsf{Part}_{a_b}(s'_2) = \{D'_6\}$.

Now we replace each action a_w, a_y and a_b in expression e' by action a to get expression $e = \mathsf{fsync}((aaa)^*aaa, (aaa)^*)$ defined over Σ. For blocks D'_i we get respective blocks D_i, as shown in Figure 5.1. And, pairing relation obtained for action a is: $\mathsf{pairing}(a) = \{(D_1, D_4), (D_2, D_5), (D_3, D_6)\}$.

5.5 Conclusion

In this chapter we defined unique sites property for expressions. We also defined expressions with pairing. Then we showed the correspondence between these expressions and product systems with separation of labels and product systems with matchings.

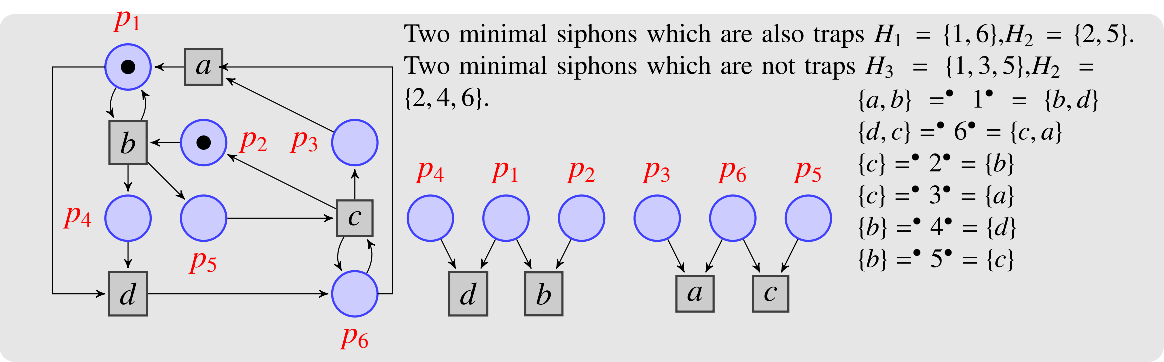
Combining these results with results in Chapter 4, we get expressions for free choice nets with unique cluster property and without it.

Chapter 6

Beyond Free Choice Nets

This thesis has dealt with 1-bounded labelled free choice nets and their connections to direct product representability. S-decomposability was a related condition we needed to use. In this small section we give some examples of non-free choice nets which we came up with while working on the results in this thesis.

6.1 Direct Product Representable but not S-decomposable Net



Two minimal siphons which are also traps $H_1 = \{1,6\}$,$H_2 = \{2,5\}$. Two minimal siphons which are not traps $H_3 = \{1,3,5\}$,$H_2 = \{2,4,6\}$.

$\{a,b\} = {}^\bullet 1^\bullet = \{b,d\}$
$\{d,c\} = {}^\bullet 6^\bullet = \{c,a\}$
$\{c\} = {}^\bullet 2^\bullet = \{b\}$
$\{c\} = {}^\bullet 3^\bullet = \{a\}$
$\{b\} = {}^\bullet 4^\bullet = \{d\}$
$\{b\} = {}^\bullet 5^\bullet = \{c\}$

Figure 6.1: Non S-decomposable but Direct Product Representable Net

Figure 6.1 shows a net with labels from a distributed alphabet $\Sigma = (\Sigma_1)$ (there is only one component in the alphabet). It is evident that this net is not S-decomposable. To see that its underlying unlabelled net also not S-decomposable, we verify that there is no sequential component which covers place p_3.

In the net given above, let us try to grow S-net around the place p_3. First add p_3. Then we have to add arcs (p_3, a) and (c, p_3). Then since transition a has only one outgoing transition to p_1 we have to add arc (a, p_1), which in turn adds arcs (p_1, b) and (p_1, d) and (b, p_1) also. Because of d, we have to add (d, p_6), which again forces an arc (p_6, a), which is not desired, because that introduces synchronization for transition a(with addition of place p_6 there will be two pre-places p_6 and p_3 for transition a).

Alternative idea of building a component for a place itself [GR92] does not give us sequential systems representing processes, as that place might not have a token in it. Also, in this approach its surrounding transitions are taken care of in the expressions which we can not write in our syntax.

But the net is certainly direct product representable since once can have a sequential system which repeatedly executes the sequence *bdca*.

Hence, having a 1-bounded net which is live, distributed choice, even satisfying the unique cluster property and direct product representable, but not free choice, does not imply that it is S-decomposable.

6.2 S-decomposable but not direct product representable

We already know that Zielonka's net (Figure 1.3) is a 1-bounded net is not direct product representable. It is clear that it is S-decomposable. It also satisfies the distributed choice property but not the unique cluster property. The free choice nets we gave in Figures 1.1 and 4.4 satisfied the unique cluster property but not the distributed choice property.

The net in Figure6.2 is 1-bounded, S-decomposable, satisfying both the distributed choice and unique cluster properties, but is not free choice and not direct product representable.

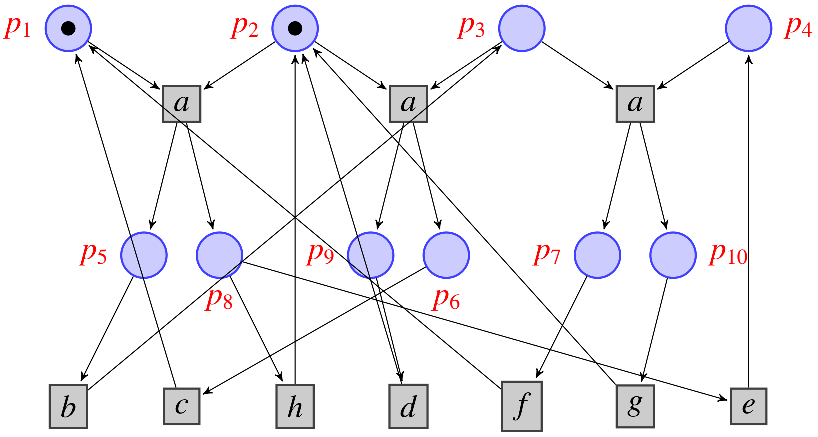


Figure 6.2: 1-bounded, S-decomposable, DCP, UCP but not Direct Product representable

Given below is the only possible S-decomposition of the net shown in Figure 6.2. One

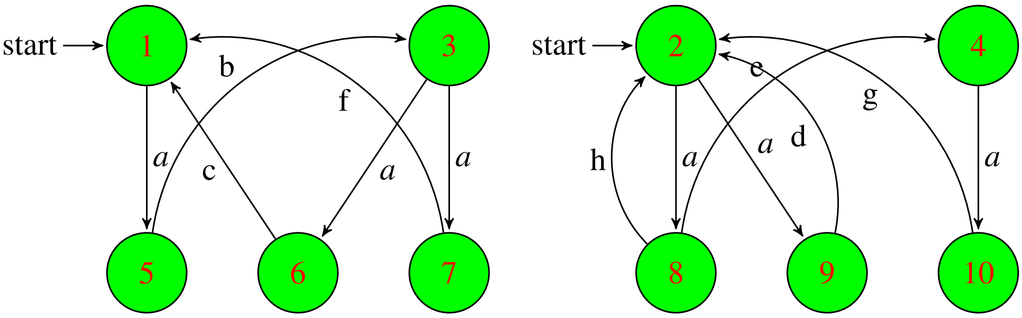


Figure 6.3: S-decomposition of net given in Figure 6.2

can see that word *abeacg* is in the language of the product system given in Figure 6.3 but not in the language of net of Figure 6.2.

6.3 Extending Hack's theorem

For any net to be S-decomposable its underlying net, which is unlabelled, should be S-decomposable in the sense of Hack [Hac72]. We discuss this issue next.

Hack showed that class of live and 1-bounded, unlabelled, free choice nets are S-decomposable. Hack's theorem relies on the important fact that in live and 1-bounded free choice nets minimal siphons are maximal traps and these are S-components of free choice net. The net in Figure 6.1 has a minimal siphon which is not a trap.

The net in Figure 6.4 is 1-bounded, deadlock-free and has a maximal trap of places $\{q_1, q_2, q_5, q_7\}$ which is not a siphon.

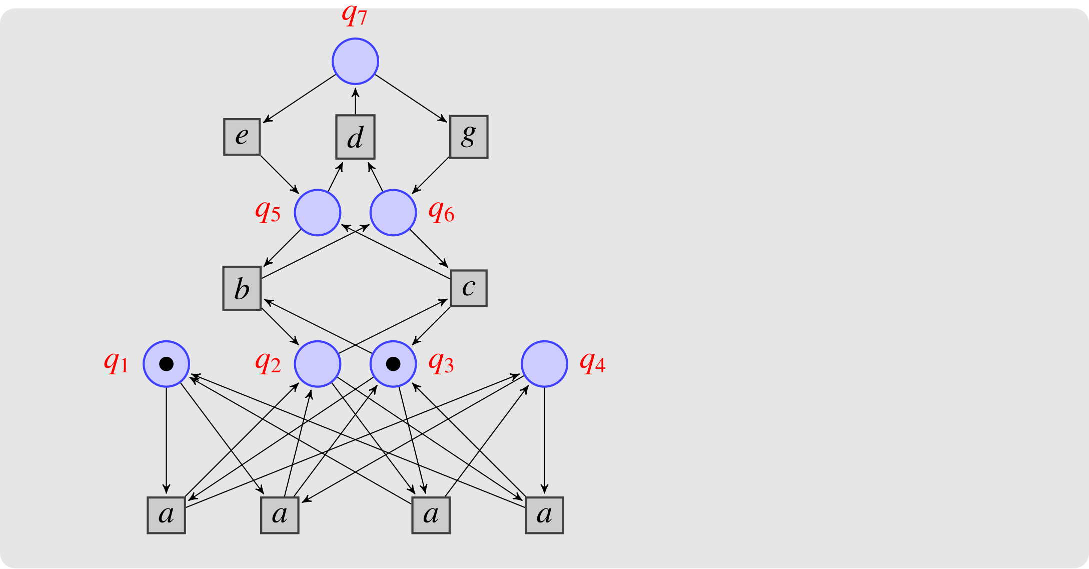


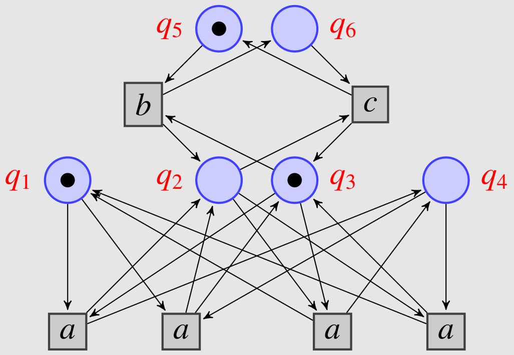
Figure 6.4: Deadlock-free net

We consider a stronger property than deadlock-freedom: the controlled siphon (CS) property of Barkaoui and Predat-Peyre [BPP96]. Figure 6.5 shows a net which is 1-bounded, satisfies the CS property, but where the places $\{q_1, q_2, q_3, q_4\}$ form a minimal siphon but not a trap.

6.4 Meta Free Choice Nets

The following definition is based on extending the idea of a free choice cluster.

Definition 95 (meta free choice)**.** *Let $g = (S_g, T_g, F_g)$ be a cluster of net N. A set of places $S_p \subseteq S_g$ is called* similar*, if*



Note that all siphons are marked now at initial marking $M_0 = \{q_1, q_3, q_5\}$. And, if we draw reachability graph we can see that all of them remain marked at all reachable markings. But the system is not live as there are two a-labelled transitions in the system which are dead. Minimal siphons are $\{q_5, q_6\}, \{q_1, q_2, q_5\}, \{q_3, q_4, q_6\}, \{q_1, q_2, q_3, q_4\}$.

Figure 6.5: CS property satisfying net

- $\forall p, q \in S_p, |p^\bullet| = |q^\bullet|$.

- $\forall p, q \in S_p,\ p^\bullet \cap q^\bullet = \Phi$, *and*

- $\biguplus_{\forall p \in S_p} p^\bullet = T_g$.

*A **cluster-cover** X_g of cluster $g = (S_g, T_g, F_g)$, is a partition of S_g, if it's each cell B is similar. A **block** is a cell which is similar.*

*A cluster $g = (S_g, T_g, F_g)$ is called a **meta free choice cluster(MFC)** if there exists a cluster-cover for set of places S_g. A net $N = (S, T, F)$ is called a meta free choice net if each cluster of N is a meta free choice cluster.*

Please note that it is possible for a cluster to have more than one cluster-cover. We make the following observations from definitions of fc cluster and MFC cluster.

Proposition 96. *Any fc cluster $g = (S_g, T_g, F_g)$ is a MFC cluster where number of blocks equals number of places S_g.*

So all FC clusters are MFC clusters but not all MFC clusters are FC clusters.

If a MFC net N is given to us then we assume that for each cluster its cluster-cover is also given. Because of Proposition 96, we have following corollary:

Corollary 97. *Class of free choice nets is strictly included in the class of MFC nets.*

Now we try to generalize the distributed choice condition using labels and meta free choiceness.

Definition 98 (generalized distributed choice)**.** *Let $g = (S_g, T_g, F_g)$ be a LFC cluster. Let $X = \{B_1, B_2, \ldots, B_l\}$ be cluster-cover of S_g. Then for each letter $a \in \Sigma$, Let T_g^a denote the subset of a-labelled transitions of T_g. For block B_i of similar places, let $B_i^a = \{p_1^a, p_2^a, \ldots, p_m^a\}$ be the places of B_i, which have at least one a-labelled transition in its post-transitions. For any place p, let $|p^\bullet|_a$ denote the number of a-labelled transitions in post-transitions of place p. For all $i \in \{1, 2, \ldots, l\}$, let $x_i^a = gcd\{|p^\bullet|_a$ such that $p \in B_i^a\}$, and, $a_i^s = T_g^a / x_i^a$.*

Then, $\forall a \in \Sigma,\ \forall i \in \{1, 2, \ldots, l\},\ x_i^a = a_1^s \times a_2^s \times \ldots \times a_{i-1}^s \times a_{i+1}^s \times \ldots \times a_l^s$.

We know that places of a block are assigned to one agent. Only thing to figure out is, for each place here in the block, what are the transitions in its post or rather how many transitions having some label are in its post.

This we get when we compute x_i^a for i-th block and for label a. Let $B_i^a = \{p_1^a, p_2^a, \ldots, p_m^a\}$ be the places of B_i, which have at least one a-labelled transition in its post-transitions. Now wlog we arrange number of post-transitions, which are a-labelled, in a tuple as shown below: $(|p_1^{a\bullet}|_a, |p_2^{a\bullet}|_a, \ldots, |p_m^{a\bullet}|_a)$.

Now take gcd, x_i^a of this tuple:
$(|p_1^{a\bullet}|_a, |p_2^{a\bullet}|_a, \ldots, |p_m^{a\bullet}|_a) = x_i^a(y_1^a, y_2^a, \ldots, y_m^a)$.

We claim that y_j^a is the number of a-labelled transitions in the post of place p_j^a.

Here we give an example to illustrate generalized distributed choice. Blocks of MFC shown in Figure 6.6 are given as $B_1 = \{p_1\}$, $B_2 = \{p_2, p_3\}$, and $B_3 = \{p_4, p_5\}$. So for label a we have $(|p_1^\bullet|_a) = 4(1)$, $(|p_2^\bullet|_a, |p_3^\bullet|_a) = 2(1, 1)$, and $(|p_4^\bullet|_a, |p_5^\bullet|_a) = 2(1, 1)$.

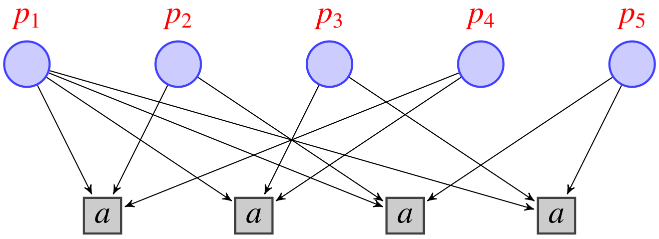


Figure 6.6: locally decomposable meta free choice cluster

Hence, parameters are:

$x_1^a = 4, x_2^a = 2,\ x_3^a = 2$ and, $a_1^s = 1, a_2^s = 2,\ a_3^s = 2$.

And, equations $x_1^a = a_2^s \times a_3^s,\ x_2^a = a_1^s \times a_3^s,\ x_3^a = a_1^s \times a_2^s$ are valid.

Above cluster is divided locally into product systems as given in the Figure 6.7.

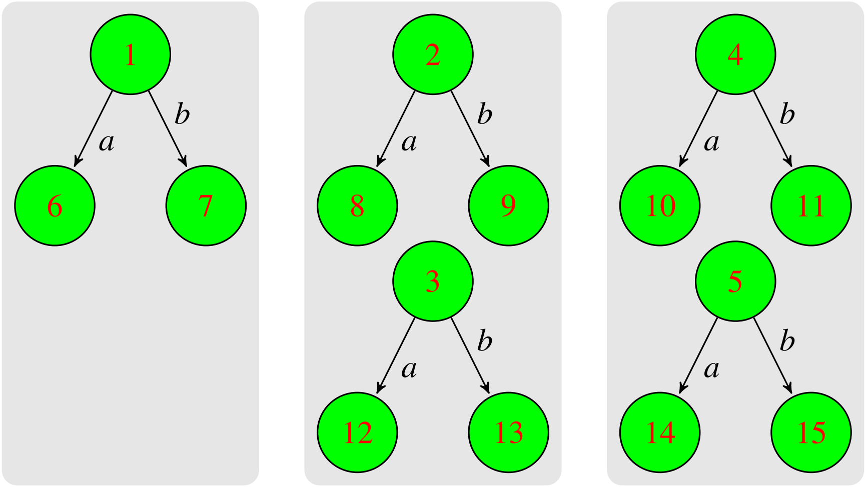


Figure 6.7: S-decomposition of MFC of Figure 6.6

So, when we construct MFC-nets from product systems , we need to start with product systems, in which each automata (each agent) behaves deterministically on global actions.

This labelling condition allows us the distribution of cluster into a product system locally at the cluster level.

However the quest for extending Hack's theorem is still elusive.

Chapter 7

Conclusions

We have given various classes of nets along with the corresponding product systems and expressions, which characterize them.

Correspondence between formalisms for various subclasses

In following Table we summarize correspondence between three formalisms for various subclasses.

Expressions	Product Systems	Labelled 1-bounded Nets
connected-T-expression	T-dag	acyclic T-net system
ω-T-expression	T-Product, structurally cyclic	T-net system, structurally cyclic
connected-FC-expression	FC-dag	acyclic FC net system
ω-FC-expression	FC-Product, structurally cyclic	
product expression, unique sites	FC-product, separation of labels	FC net system, unique cluster property
product expression, equal choice pairing, consistency of pairing,	product system conflict equivalent matching, consistency with matching	FC net system

Table 7.1: Correspondence between expressions, product automata and labelled 1-bounded S-decomposable distributed free choice Petri nets

Resources required to check various properties

We collect below upper bounds for the resources required for checking various properties of the expressions and product systems used in this thesis.

For product systems checking whether given matching is conflict-equivalent is in PTIME and checking if the product system is consistent with the given matching is in PSPACE by Proposition 64. And, for product expressions checking whether given matching is equal-choice is in PTIME and checking if the product expression is consistent with the given pairing is in PSPACE by Proposition 85.

Expressions/ Product systems	equal choice/ conflict equivalence	Deadlock	Equivalence	Emptiness
connected-T-expression or T-dag	Trivial	PTIME(d1)	PTIME(l1)	PTIME(e1)
connected-FC-expression or FC-dag	PTIME(c1)	NP(d2)	coNP(l2)	coNP(e2)
product expression (with pairing) or product system (with matching)	PTIME	PSPACE(d3)	PSPACE(l3)	PSPACE(e3)

Table 7.2: Resources required for checking various properties

(c1) By Lemma 35.

(d1) By Lemma 37.

(d2) By Lemma 37.

(d3) First convert to product system with matching and use similar algorithm as in the proof of Proposition 64

(l1) By Theorem 38.

(l2) By Theorem 38. Proof for product systems in which each component is acyclic is given in [SJ09].

(l3) First convert to product system with matching and use [SHRS96].

(e1) By Corollary 39.

(e2) By Corollary 39.

(e3) First convert to product system with matching and use similar algorithm as in the proof of Proposition 64 or using (*l*3).

For free choice nets which are S-decomposable 1-bounded and satisfying distributed choice property, checking language equivalence and emptiness can be done by first converting it into direct product representation(all translations for each corresponding class are in polynomial time), then apply algorithms of product systems with complexities shown above. And, using Proposition 67 distributed choice property for nets is checkable in PTIME.

Future work

We have given direct product representation for labelled 1-bounded free choice nets having distributed choice property. These nets are assumed to be S-decomposable and labelled with a distributed alphabet. One direction of research is to relax the condition of distributed choice property and deal with the full class of labelled free choice nets.

Another aspect that can be the object of investigation is to assume that free choice nets considered are labelled but not necessarily S-decomposable (i.e., S-decomposition is not given a priori). Then the question is to extend Hack's theorem for labelled free choice nets.

Axiomatization of equivalence for the different classes of expressions is another goal which can be pursued.

Bibliography

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

[Arn98] André Arnold. Synchronized products of transition systems and their analysis. In Jörg Desel and Manuel Silva, editors, *ICATPN*, volume 1420 of *LNCS*, pages 26–27. Springer, 1998.

[BPP96] Kamel Barkaoui and Jean-François Pradat-Peyre. On liveness and controlled siphons in petri nets. In Jonathan Billington and Wolfgang Reisig, editors, *Application and Theory of Petri Nets*, volume 1091 of *LNCS*, pages 57–72. Springer, 1996.

[BRR87] Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors. *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, September 1986*, volume 254 of *LNCS*. Springer, 1987.

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

[BS83] Eike Best and Michael W. Shields. Some equivalence results for free choice nets and simple nets and on the periodicity of live free choice nets. In *CAAP*, pages 141–154, 1983.

[BV84] Eike Best and Klaus Voss. Free choice systems have home states. *Acta Informatica*, 21:89–100, 1984.

[CEP95] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. *Theor. Comput. Sci.*, 147(1&2):117–136, 1995.

[CHEP71] Fred Commoner, Anatol W. Holt, Shimon Even, and Amir Pnueli. Marked directed graphs. *J. Comput. Syst. Sci.*, 5(5):511–523, 1971.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[CMT99] Ilaria Castellani, Madhavan Mukund, and P.S. Thiagarajan. Synthesizing distributed transition systems from global specification. In C. Pandu Rangan, Venkatesh Raman, and R. Ramanujam, editors, *FSTTCS*, volume 1738 of *LNCS*, pages 219–231. Springer, 1999.

[cpn] Cpn tools. http://cpntools.org.

[DE95] Jörg Desel and Javier Esparza. *Free choice Petri nets*. Cambridge Univ Press, 1995.

[Des92] Jörg Desel. A proof of the rank theorem for extended free choice nets. In *Application and Theory of Petri Nets*, pages 134–153, 1992.

[DR95] Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1995.

[ES92] Javier Esparza and Manuel Silva. A polynomial-time algorithm to decide liveness of bounded free choice nets. *Theoret. Comp. Sci.*, 102(1):185–205, 1992.

[Fri02] Jeffrey EF Friedl. *Mastering regular expressions*. O'Reilly Media, Inc., 2002.

[GH08] Hermann Gruber and Markus Holzer. Finite automata, digraph connectivity, and regular expression size. In *ICALP(2)*, pages 39–50, 2008.

[GM06] Blaise Genest and Anca Muscholl. Constructing exponential-size deterministic zielonka automata. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP(2)*, volume 4052 of *LNCS*, pages 565–576. Springer, 2006.

[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. *Fundam. Inform.*, 4(2):427–498, 1981.

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

[Hac76] Michel Henri Theódore Hack. Petri net languages. Technical Report TR-159, MIT, 1976.

[HMU03] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley, 2003.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[Jan86] Matthias Jantzen. Language theory of petri nets. In Brauer et al. [BRR87], pages 397–412.

[JvdAB+13] Kurt Jensen, Wil van der Aalst, Gianfranco Balbo, Maciej Koutny, and Karsten Wolf, editors. *Transactions on Petri Nets and other models of concurrency, based on the Advanced Course on Petri Nets, held in Rostock, September 2010*, volume 7480 of *LNCS*. Springer, 2013.

[Kle56] Stephen C. Kleene. Representation of events in nerve nets and finite automata. In Claude E. Shannon and John McCarthy, editors, *Automata studies*, pages 3–41. Princeton, 1956.

[LMP11] Kamal Lodaya, Madhavan Mukund, and Ramchandra Phawade. Kleene theorems for product systems. In Markus Holzer, Martin Kutrib, and Giovanni Pighizzini, editors, *Descriptional Complexity of Formal Systems*, volume 6808 of *LNCS*, pages 235–247. Springer, 2011.

[Lod06] Kamal Lodaya. Product automata and process algebra. In Paritosh Pandya and Dang van Hung, editors, *Proc. 4th Softw. Engg. Formal Meth., Pune*, pages 128–136. IEEE, 2006.

[LRR03] Kamal Lodaya, D. Ranganayakulu, and K. Rangarajan. Hierarchical structure of 1-safe petri nets. *Advances in Computing Science–ASIAN 2003*, pages 173–187, 2003.

[LW00] K. Lodaya and P. Weil. Series-parallel languages and the bounded-width property* 1. *Theoretical Computer Science*, 237(1-2):347–380, 2000.

[Maz77] Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. Technical Report DIAMI-PB-78, Aarhus University, 1977.

[Maz86] Antoni Mazurkiewicz. Trace theory. *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pages 278–324, 1986.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

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

[Moh98] Swarup Kumar Mohalik. *Models for concurrency: local presentations for finite state distributed systems*. University of Madras, 1998.

[MR97] Swarup Mohalik and Ramaswamy Ramanujam. Assumption-commitment in automata. In S. Ramesh and G. Sivakumar, editors, *FSTTCS*, volume 1346 of *LNCS*, pages 153–168. Springer, 1997.

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

[MS97] Madhavan Mukund and Milind A. Sohoni. Keeping track of the latest gossip in a distributed system. *Distrib. Comp.*, 10(3):137–148, 1997.

[Muk02] Madhavan Mukund. From global specifications to distributed implementation. In Caillaud B., Darendeau P., Lavango L., and Xie X., editors, *Synthesis and Control of Discrete Event Systems*, pages 19–34. Springer, 2002.

[Muk11] Madhavan Mukund. Automata on distributed alphabets. In Deepak D'Souza and Priti Shankar, editors, *Modern Applications of Automata Theory*. World Scientific, 2011.

[Mur89] Tadeo Murata. Petri nets: properties, analysis, and applications. *IEEE Transactions on circuits and systems*, 77(4):541,580, 1989.

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

[Och85] Edward Ochmański. Regular behaviour of concurrent systems. *Bulletin of the EATCS*, 27:56–67, 1985.

[pep] Pep tool. http://peptool.sourceforge.net.

[Pet62] Carl Adam Petri. *Kommunikation mit automaten.* Institut für Instrumentelle Mathematik, Bonn, 1962.

[Pet63] Carl Adam Petri. Fundamentals of a theory of asynchronous information flow. In Cicely M. Popplewell, editor, *Proc. IFIP Congress 1962, Munich*, pages 386–390. North-Holland, 1963.

[Pet76] James L. Peterson. Computation sequence sets. *J. Comput. Syst. Sci.*, 13(1):1–24, 1976.

[Pet81] James L Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, 1981.

[Pha14] Ramchandra Phawade. Article: Direct product representation of labelled free choice nets. *International Journal of Computer Applications*, 99(16):1–8, 2014.

[PL14] Ramchandra Phawade and Kamal Lodaya. Kleene theorems for labelled free choice nets. In Daniel Moldt and Heiko Rölke, editors, *International Workshop on Petri Nets and Software Engineering (PNSE'14)*, number 1160 in CEUR Workshop Proceedings, pages 75–89, Aachen, 2014. CEUR-WS.org.

[PL15] Ramchandra Phawade and Kamal Lodaya. Kleene theorems for synchronous products with matching. *Transactions on Petri nets and other models of concurrency*, X, 2015. To Appear.

[Rei85] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

[RR98] Wolfgang Reisig and Grzegorz Rozenberg, editors. *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *LNCS*. Springer, 1998.

[Sak09] Jacques Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[SHRS96] Sandeep K. Shukla, Harry B. Hunt(III), Daniel J. Rosenkrantz, and Richard Edwin Stearns. On the complexity of relational problems for finite state processes (extended abstract). In Friedhelm Meyer auf der Heide and Burkhard Monien, editors, *ICALP*, volume 1099 of *LNCS*, pages 466–477. Springer, 1996.

[SJ09] Zdenek Sawa and Petr Jancar. Hardness of equivalence checking for composed finite-state systems. *Acta Inf.*, 46(3):169–191, 2009.

[Thi95] P.S. Thiagarajan. A trace consistent subset of ptl. In Insup Lee and Scott A. Smolka, editors, *Proc. 6th Concur, Philadelphia*, volume 962 of *LNCS*, pages 438–452. Springer, 1995.

[Thi02] P.S. Thiagarajan. Regular event structures and finite petri nets: A conjecture. In Wilfried Brauer, Hartmut Ehrig, Juhani Karhumäki, and Arto Salomaa, editors, *Formal and Natural Computing*, volume 2300 of *LNCS*, pages 244–256. Springer, 2002.

[TV84] P.S. Thiagarajan and Klaus Voss. In praise of free choice nets. In *European Workshop on Applications and Theory in Petri Nets*, pages 438–454, 1984.

[Yen06] Hsu-Chun Yen. Introduction to petri net theory. In Zoltán Ésik, Carlos Martín-Vide, and Victor Mitrana, editors, *Recent Advances in Formal Languages and Applications*, volume 25 of *Studies in Computational Intelligence*, pages 343–373. Springer, 2006.

[Zie87] Wieslaw Ziełonka. Notes on finite asynchronous automata. *ITA*, 21(2):99–135, 1987.