Journal of Universal Computer Science, vol. 5, no. 9 (1999), 542-551
submitted: 10/6/99, accepted: 15/7/99, appeared: 28/9/99 © Springer Pub. Co.

Issues Related to Distributed Processing of Picture Languages

Padmanabhan Krishnan
Department of Computer Science
University of Canterbury, PBag 4800
Christchurch, New Zealand
E-mail: paddy@cosc.canterbury.ac.nz

Abstract: In this article we show that the parallel processing of pictures requires extending the notion of pictures with blank spaces. For the purposes of this article, parallel processing is defined in terms of a product construction.

Key Words: picture language, synchronisation, product language
Category: F.1.1

1 Introduction and Motivation

The term visual programming is used to mean two distinct aspects of programming. In the first context visual programming means representing, developing, manipulating programs using graphical objects. But the behaviour of the programs follow the usual semantics. Usually the graphical objects can be mapped onto some program fragment expressed in a particular language. For example, the article by [Usher and Jackson, 1998] discussing the use of Petri Nets to represent concurrent programs falls in this class. In the second context visual programming means programming where graphical objects are manipulated. The programs in this case need not be visual. An example of such work is presented in [Erwig and Meyer, 1995]. The volume [Marriott and Meyer, 1998] discusses various issues. For the purposes of this article we assume the second interpretation. That is, we are concerned with the processing of pictures (and not representing the programs which process the pictures as pictures themselves).

There are various issues related to visual computation including spatial representation (exact geometry), logical connectedness (via description logics), generating pictures using grammars or rewrite systems, using algebraic techniques etc. The proceedings [Marriott and Meyer, 1998] and the book [Glasgow et al., 1995] together provide a good overview.

The introductory tutorial [Giammarresi and Restivo, 1997] presents various models describing computation over pictures. These include four way automata, regular expressions involving horizontal and vertical composition, tessellation automata and tiling systems. These models are not interested in spatial layout but in logical connectedness. Thus pictures are represented as a two-dimensional matrix. Given a picture P, the symbol $P_{i,j}$ is dependent on the symbol $P_{i-1,j}$ and $P_{i,j-1}$. That is, a symbol is dependent on its row and column predecessor.

One of the most advanced theoretical models is that of a two dimensional online tessellation automata (2OTA). The class of 2OTA's is strictly more powerful than the class of four way automata and two dimensional right linear grammars.

The main aim of this article is to present a model of concurrent computation for two-dimensional or picture languages. We use the well understood concurrency model of shared variables (or synchronisation). Although two dimensional languages appear in the context of parallel processing or cellular automata, the theory of finite state recognisability is more or less based on sequential computation [Giammarresi and Restivo, 1997]. Even though models like 2OTA involve parallelism, there are some restrictions. For example, one only has a single wave front that propagates through the picture. There is no general notion of synchronisation and distributed execution. That is, the state at position (i, j) is computed by the transition relation $\delta(q_{i-1,j}, q_{i,j-1}, a_{i,j})$. Thus in the first step $q_{1,1}$ is calculated, in the second step $q_{1,2}$ and $q_{2,1}$ are calculated, in the third step $q_{2,2}, q_{1,3}$ and $q_{3,1}$ are calculated etc. The following example illustrates the key concepts.

Example 1. Consider an automaton over a single symbol (a) whose behaviour is described as follows. Let the set of states be {0,1,2,3} with 0 as the initial state and 1 as the only final state. The transitions of the automaton are
$\delta(0, 0, a) = \delta(2, 3) = 1$,
$\delta(0, 1, a) = \delta(0, 2, a) = \delta(2, 1, a) = \delta(2, 2, a) = 2$ and
$\delta(1, 0, a) = \delta(3, 0, a) = \delta(1, 3, a) = \delta(3, 3, a) = 3$.

Thus the state 1 is associated with the diagonal, with the state 2 associated with all elements above the diagonal and state 3 associated with all elements below the diagonal. An accepted picture requires that the final state (associated with the lowest, rightmost corner) be 1, only square pictures are accepted.

In the case of linear words, models such as product automata or asynchronous automata [Zielonka, 1987] are based on (a) combining words to form "larger" words and (b) synchronisation on common symbols. Computations with 2OTAs do not involve the combinations of pictures to form larger pictures. That is, there is no notion of multiple 2OTAs working in parallel.

Here we present a product like construction for two dimensional languages and 2OTA based on the notion of independence presented in [Mazurkiewicz, 1984] and [Zielonka, 1987]. The model we develop is an extension of the model for words. Thus the model when restricted to pictures of sizes (1,n) or (n,1) one gets the usual model for words.

This work can be viewed as generalising the notion of row/column composition of compatible pictures to parallel composition. That is, we are taking many pictures and constructing a larger picture that contains the information from the original pictures. This work can also be viewed as two automata working together on a single picture. The only related work is on combining two sets of commands to draw pictures so that the result is a connected picture [Ratoandromanana and Robilliard, 1994]. However, it is more to deal with sequential composition than concurrency and synchronisation. That is, [Ratoandromanana and Robilliard, 1994] consider the picture obtained via the sequential composition of commands associated with the drawing of p_1 and p_2.

To keep the presentation notationally simple, we assume only two agents. That is, we consider combining only two picture components. This can easily be extended to a n agent system. Although one could define a notion of horizontal independence distinct from vertical independence, we assume a single independence structure for the sake of simplicity. This allows us to generalise the behaviour of 2OTAs without drastically altering the alphabet structure.

As in [Zielonka, 1987] we associate the alphabets Σ_1 and Σ_2 with the two agents. We let Σ denote $\Sigma_1 \cup \Sigma_2$. For $a, b \in \Sigma$ we say aIB (a and b are *independent*) if and only if a and b belong only to distinct alphabet sets. That is, if a belongs to Σ_1 then a cannot belong to Σ_2 and b will belong only to Σ_2. If a symbol (say c) belongs to both Σ_1 and Σ_2 the two agents must synchronise on it. This is akin to performing consistent operations on shared variables.

Before we present the technical details in the next section, we present a few examples to motivate the various definitions. Consider the pictures $\begin{pmatrix} a\ b\ b \\ a\ a\ b \end{pmatrix}$ over $\{a, b\}$ and $\begin{pmatrix} b\ b\ c \\ c\ c\ b \end{pmatrix}$ over $\{b, c\}$. As the symbol b is common to both the alphabets, the combination of the two pictures should synchronise on a b. The symbols a and c are independent. If we perform a row-wise synchronised shuffle we obtain the rows $(a\ b\ b\ c)$ and $([a\ a\ c\ c]\ b)$ where $[w]$ represents the set of words obtained by exchanging the positions of adjacent independent symbols [Zielonka, 1987]. Thus $([a\ a\ c\ c]\ b)$ stands for the set containing the sequences $(a\ a\ c\ c\ b)$, $(a\ c\ a\ c\ b)$, $(a\ c\ c\ a\ b)$, $(c\ a\ a\ c\ b)$, $(c\ a\ c\ a\ b)$ and $(c\ c\ a\ a\ b)$. As pictures are rectangular, it is not possible to put together the two rows to obtain a larger picture. Thus under a rectangular definition of pictures, the combination of the two pictures is undefined as the resulting rows are of different sizes. If we perform a column-wise synchronisation, we obtain the columns $\emptyset$, $(b\ [c\ a])$ and $\emptyset$. Once again we cannot put these together to form a picture. One has to define the synchronised shuffle in the above case to be empty as the sizes do not tally.

However, insisting that all the shuffled rows or columns are of identical size is also not sufficient. Consider the picture $\begin{pmatrix} a\ b \\ b\ a \end{pmatrix}$ over $\{a, b\}$ and $\begin{pmatrix} c\ b \\ b\ c \end{pmatrix}$ over $\{c, b\}$. Now the row-wise shuffle can produce the rows $(a\ c\ b)$ and $(b\ a\ c)$ which can be combined to produce the picture $\begin{pmatrix} a\ c\ b \\ b\ a\ c \end{pmatrix}$. Similarly, the column-wise shuffle can yield the columns $(a\ c\ b)$ and $(b\ a\ c)$ which can be combined to obtain the picture $\begin{pmatrix} a\ b \\ c\ a \\ b\ c \end{pmatrix}$.

Both the resulting pictures are unsatisfactory. Recall that one of the aims was to maintain the original logical dependence when the two pictures were combined. The property of logical dependence is violated. In the picture obtained by row-wise shuffle the b in row 2 column 1 is positioned incorrectly. In the component pictures, the b is dependent on the a and the c in row 1 column 1 of the constituent pictures. However the c has been pushed to row 1 column 2 in the larger picture. Hence in the combined picture the b is no longer dependent on the c.

Similarly in the picture obtained by the column-wise shuffle, the b in row 1 column 2 is positioned incorrectly. Therefore, when we shuffle pictures a concatenation of shuffled rows or columns is not sufficient. We have to pay attention to the various spatial (i.e., two dimensional) dependencies that exist in the original pictures.

One could deem the composition of such pictures to be invalid, but that restricts the parallel composition to a very small subclass of pictures. We present a technique that is applicable to a reasonable set of pictures.

Our solution to obtain a satisfactory solution is to introduce ϵ (the empty symbol) in various positions. These ϵs will ensure that valid picture shuffles from the individual pictures are obtained . The shuffle in the second example can be defined to be $\begin{pmatrix} a\ c\ b\ \epsilon \\ \epsilon\ b\ a\ c \end{pmatrix}$ which satisfies the rectangular as well as the causality requirements. The first b in the second row has been pushed to the right and is now dependent on the c. Being the first non-empty symbol on the second row it has no left-hand predecessor. The column-wise shuffle can similarly be defined to be $\begin{pmatrix} a\ \epsilon \\ c\ b \\ b\ a \\ \epsilon\ c \end{pmatrix}$.

Note that the introduction of ϵ means that the logical dependence has to be defined with care. In both the row and column shuffle, the last c does not depend on ϵ (it actually depends on the b). The dependence on the ϵ has to be reflected (in different directions) to the dependence on the b.

But having one ϵ is not sufficient. There is a need to have row and column ϵs to record the fact that the ϵs were introduced due to row and column shuffle respectively. Consider the picture fragment $\begin{pmatrix} \cdot\ a\ b \\ c\ \epsilon\ d \\ e\ f\ \cdot \end{pmatrix}$. If the ϵ was introduced during a row shuffle, the symbol d has a causal dependency on the symbols c and b. It is this dependence on b that introduces the ϵ. The symbol f depends on e and c. Note that f cannot depend on d as that would have required the introduction of an ϵ between e and f. If the ϵ was introduced during a column shuffle, the symbol f depends on a and e (not on e and c). Again note that the dependence on e is what introduces the ϵ. The symbol d depends on a and b. This shows that it is important to know how the ϵ symbols were introduced. We use ϵ_r for those introduced during the row shuffle and ϵ_c for those introduced during the column shuffle.

In this paper we focus only on the row-wise shuffle. A theory for the column wise shuffle can be developed in a similar fashion. But we continue to write ϵ_r to differentiate it from ϵ and emphasise the fact that we are dealing with row-wise shuffle. It is easy to see that a given composite picture will not contain both ϵ_r and ϵ_c.

2 Technical Details

This section can be viewed as consisting of two parts. The first deals with language theoretic constructs while the second part is concerned with automata theoretic constructs. This section will conclude by presenting a relationship between the two views.

We define a shuffle of words which retains positional information. This positional information will be used to concatenate rows in a fashion that respects the logical dependencies.

As for product automata, synchronisation has to occur on common alphabets. Independent actions can occur in any order. This is defined below. Here ϵ stands for the empty string.

Definition 1. For $w_1 \in \Sigma_1^*$ and $w_2 \in \Sigma_2^*$, the *shuffle* denoted by $w_1 \ {}_0\|_0 \ w_2$ is defined inductively as follows.

1. $\epsilon \ {}_l\|_m \ \epsilon = \epsilon_r^{(l,m)^*}$
2. $\epsilon \ {}_l\|_m \ (b \cdot bs) = S \cup \{\epsilon_r^{(l,m)^*} \cdot w \mid w \in S\}$ provided $b \notin \Sigma_1$ where $S = \{b^{(l,m+1)} \cdot w' \mid w' \in (\epsilon \ {}_l\|_{m+1} \ bs)\}$
3. $(a \cdot as) \ {}_l\|_m \ \epsilon = S \cup \{\epsilon_r^{(l,m)^*} \cdot w \mid w \in S\}$ provided $a \notin \Sigma_2$ where $S = \{a^{(l+1,m)} \cdot w' \mid w' \in (as \ {}_{l+1}\|_m \ \epsilon)\}$
4. $(a \cdot as) \ {}_l\|_m \ (b \cdot bs) = S \cup \{\epsilon_r^{(l,m)^*} \cdot w \mid w \in S\}$ where

$$\begin{aligned} S = & \{a^{(l+1,m)} \cdot w \mid a \notin \Sigma_2, w \in (as \ {}_{l+1}\|_m \ (b \cdot bs))\} \cup \\ & \{b^{(l,m+1)} \cdot w \mid b \notin \Sigma_1, w \in ((a \cdot as) \ {}_l\|_{m+1} \ bs)\} \cup \\ & \{a^{(l+1,m+1)} \cdot w \mid a = b, w \in (as \ {}_{l+1}\|_{m+1} \ bs)\} \end{aligned}$$

The above definition is an extension of the usual definition of a synchronised shuffle of two languages. It permits the insertion of ϵ_r at arbitrary column locations. Hence combining empty words can yield a sequence of ϵ_rs. In general $\alpha \ {}_l\|_m \ \beta$ indicates that α will start from the l^{th} position and β will start from the m^{th} position in the shuffled words. Or in other words, α and β are the residues after removing $l-1$ symbols from and $m-1$ symbols the original words.

In a string a symbol of the form $a^{(p_1,p_2)}$ indicates that $p_1 - 1$ symbols from the left hand side and $p_2 - 1$ symbols from the right hand side have been consumed before consuming a. This will be taken into account while defining the concatenation of rows to obtain pictures. Given a symbol s of the form $a^{(p_1,p_2)}$ we use $pos(s) = (p_1, p_2)$ to indicate the position associated with the symbol. We use the term *positioned-word* and *positioned-picture* to indicate a word and a picture whose symbols have a position associated with them. If w is a positioned word we let $\overline{w}$ the word obtained by erasing the positions associated with all the symbols in w.

Example 2. Consider $a \cdot b \ {}_0\|_0 \ c \cdot b$ with a and c being independent. The result of the shuffle will contain $a^{(1,0)} \cdot c^{(1,1)} \cdot b^{(2,2)}$ and $c^{(0,1)} \cdot a^{(1,1)} \cdot b^{(2,2)}$. It shows that a is the first symbol in the first string while b is the second symbol in both the strings.

One could also insert arbitrary number of ϵ_r's with the appropriate position information. Hence $a^{(1,0)} \cdot c^{(1,1)} \cdot b^{(2,2)} \cdot \epsilon_r^{(2,2)}$ also belongs to the shuffle as does $a^{(1,0)} \cdot \epsilon_r^{1,0} \cdot c^{(1,1)} \cdot b^{(2,2)}$. The usefulness of the having ϵ_r at the end of the string was demonstrated via an example in section 1.

Similarly, the shuffle $a \cdot a_0\|_0 \ b \cdot c$ will contain the strings $b^{(1,1)} \cdot a^{(2,1)} \cdot c^{(2,2)}$ and $\epsilon_r^{(0,0)} \cdot b^{(1,1)} \cdot a^{(2,1)} \cdot c^{(2,2)}$.

When the shuffled rows are combined to form a picture, it is not always the case that the result is a well formed picture. For instance, the result may not be rectangular. Even if the result is rectangular, it may not satisfy all the logical constraints.

The following definition formalises the latter point. The rectangular requirement is as for standard pictures and is not reproduced here. That is, from now on we will assume that our pictures implicitly satisfy the requirements for being rectangular.

Definition 2. A rectangular positioned picture p is *well formed* if and only if the following conditions hold for every relevant i and j. For the following conditions we let $pos(p_{(i,j)})$ be (l_1, l_2) and $pos(p_{(i-1,j)})$ be (l_3, l_4).

1. If $p_{(i,j)} \in \Sigma_1 \cap \Sigma_2$ then $l_1 = l_3$ and $l_2 = l_4$.
2. If $p_{(i,j)} \in \Sigma_1 - \Sigma_2$ then $l_1 = l_3$.
3. If $p_{(i,j)} \in \Sigma_2 - \Sigma_1$ then $l_2 = l_4$.

In a picture without ϵ_rs the symbol $pos(p_{(i,j)})$ depends on $pos(p_{(i-1,j)})$. That is, the column positions must agree. The positions associated with symbols is an indication of of the column positions in a positioned word and picture. The dependence on row positions is automatically satisfied as we are performing the row shuffle. If a symbol belongs to the alphabet of both the agents, the column dependence of both agents has to be noted. If a symbol belongs to the alphabet of only one agent, only its column dependence has to be satisfied. The relevant column position of the other agent does not matter.

One can now verify that the examples considered in section 1 are valid (i.e., *well formed*) using the positional information.

Now to an impossible shuffle. Consider the shuffle of the pictures $\begin{pmatrix} a\ b \\ b\ a \end{pmatrix}$ over $\{a, b\}$ and $\begin{pmatrix} b\ c \\ c\ b \end{pmatrix}$ over $\{b, c\}$. While the row-wise shuffle essentially yields $(a\ b\ c)$ and $(c\ b\ a)$, these two rows cannot be merged. Even if we add ϵ_rs, the positional values are $((1,0)\ (2,1)\ (2,2))$ and $((0,1)\ (1,2)\ (2,2))$ respectively. These cannot be combined as the position associated with the b is $(1,2)$ and no such position can exist in the first row. As no other row-wise shuffles are possible, the shuffle of pictures is undefined. Intuitively this is not surprising. The b on the second row has to be both on the left (from the first picture) and the right (from the second picture) of the b on the first row. This is clearly not possible.

Based on the above definition the notion of a shuffle of pictures and hence languages can be described. As we are defining a row-wise shuffle, the number of rows in the two pictures must agree. The number of columns could be different. The number of columns in the resulting picture will depend on the specifics of the two pictures.

Definition 3. Given a $m \times n$ picture p_1 over Σ_1 and a $m \times n'$ picture p_2 over Σ_2 define their *parallel composition*
$p_1 \parallel p_2 = \{\ \overline{x} \mid\ x$ is well formed where $\forall\ 1 \le i \le m,\ x(i) \in p_1(i)\ {}_0\|_0\ \ p_2(i)\}$
Extend this to languages as follows.
$L_1 \parallel L_2 = \{\ p \mid\ p \in (p_1 \parallel p_2), p_1 \in L_1, p_2 \in L2\}$

The projection of a picture over $\Sigma_1 \cup \Sigma_2$ to Σ_1 or Σ_2 is defined in two stages. The first is to replace all symbols of the other alphabet with ϵ_r. The second stage is to treat the ϵ_rs as empty symbols and compact the various rows. Denote the projection of p into the i^{th} alphabet as $\pi_i(p)$.

Lemma 4. If $p \in (p_1 \parallel p_2)$ then $\pi_1(p) = p_1$ and $\pi_2(p) = p_2$.

This shows that the projection result for strings more or less carries over to pictures. The only extra work is removing the ϵ_rs. Note that definition 1 permits the insertion of arbitrary ϵ_rs. That is, ϵ_rs may be inserted even when not required.

The following lemma is useful in identifying the *essential* ϵ_rs. This provides an initial description towards a canonical form for pictures.

Lemma 5. Let a, c be symbols that are not ϵ_r. Also let p_1, p_2, p_1', p_2' belong to Σ^{**} and $w_1, w_2, w_1', w_2, , w_2''$ belong to Σ^*.

If the picture $\begin{pmatrix} & p_1 & \\ w_1 & a \ b & w_1' \\ w_2 & \epsilon_r \ c & w_2' \\ & p_2 & \end{pmatrix}$ belongs to a product language L such that the picture $\begin{pmatrix} & p_1' & \\ w_1 & a \ b & w_1' \\ w_2 & c & w_2'' \\ & p_2' & \end{pmatrix}$ does not belong to L for any p_1', p_2', w_2'' then the following holds. $\{b, c\} \subseteq \Sigma_i$ for some i.

Proof: Without loss of generality assume $b \in \Sigma_1$. Let the position associated with a in the accepted word be (x_1, x_2). If $a \in \Sigma_1$, the position associated with b will be $(x_1 + 1, x_2)$. Now if c belongs only to Σ_2, clearly c can be placed under the a as the only requirement is that the second component of the position of c be x_2. The other case of $a \in \Sigma_2$ yields a similar result. Hence c cannot belong only to Σ_2. $\square$

The above lemma identifies the situation where the c cannot be moved left over an ϵ_r. Therefore, c does indeed depend on b and not on the a due to which the ϵ_r is essential. If c depends on b they both belong to some common agent.

Thus far we have considered only the language aspects of computing with pictures. Now we discuss the automata model for parallel computation. This is done in two stages. The first is to extend the notion of run of a 2OTA to include pictures containing ϵ_r. As a notational convenience, if the original automaton is $\mathcal{A}$, we denote the extended automaton as $\mathcal{A}^{\epsilon_r}$. Recall that the run of 2OTA is map ρ where $\rho(i,j) = q$ where $(\rho(i-1,j), \rho(i,j-1)) \xrightarrow{p_{i,j}} q$. That is, the state associated a given position is derived from the states associated with the neighbouring positions.

Definition 6. Given $\mathcal{A} = (Q, \Sigma, \longrightarrow, q_0, F)$ define its run over a $m \times n$ picture $(p_{i,j})$ containing ϵ_r (i.e., the behaviour of $\mathcal{A}^{\epsilon_r}$) as follows.

The run $\rho : \{0..m\} \times \{0..n\} \longrightarrow Q$ is a function such that

1. $\rho(0, i) = q_0$ for $0 \leq i \leq n$.
2. $\rho(j, 0) = q_0$ for $0 \leq j \leq m$.
3. If $p_{i,j} = \epsilon_r$, $\rho(i,j) = \rho(i, j-1)$.
4. If $p_{i,j} \neq \epsilon_r$, $\rho(i,j) = q$ where $(q_1, q_2) \xrightarrow{p_{i,j}} q$ where $q_1 = \rho(i-1, j)$ and $q_2 = \rho(i, j-1)$

The run is accepting iff $\rho(m, n) \in F$.

The above definition is a slight modification of the definition used for 2OTA [Giammarresi and Restivo, 1997]. It makes it clear that the ϵ_r's are skipped. That is, the state does not change and $\rho(i,j)$ gets a copy of $\rho(i, j-1)$. It also shows the row-wise shuffle reflects the dependence along the appropriate column.

The product of two automata is now defined in the standard way and is given in definition 7.

Definition 7. Given 2OTAs $(Q_1,\Sigma_1, \longrightarrow_1, q_0^1, F^1)$ and $(Q_2,\Sigma_2, \longrightarrow_2, q_0^2, F^2)$ define a product automaton $(Q,\Sigma, \longrightarrow, q_0, F)$, where $Q = Q_1 \times Q_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$, $q_0 = (q_0^1, q_0^2)$, $F = F_1 \times F_2$.

The transition relation $(q_1, q_2) \xrightarrow{a} (q_1', q_2')$ is defined as follows.

If $a \in \Sigma_1 - \Sigma_2$ $q_2 = q_2'$ and $q_1 \xrightarrow{a}_1 q_1'$.
If $a \in \Sigma_2 - \Sigma_1$ $q_1 = q_1'$ and $q_2 \xrightarrow{a}_2 q_2'$.
If $a \in \Sigma_2 \cap \Sigma_1$, $q_1 \xrightarrow{a}_1 q_1'$ and $q_2 \xrightarrow{a}_2 q_2'$.

The following result describes the relationship between the language and automaton used in the shuffle.

Lemma 8. Given $\mathcal{A}_1$ accepting L_1 and $\mathcal{A}_2$ accepting L_2. $L_1 \parallel L_2$ is precisely accepted by $(\mathcal{A}_1 \times \mathcal{A}_2)^{\epsilon_r}$.

Proof: For every p belonging to $L_1 \parallel L_2$, there is a p_1 belonging to L_1 and p_2 belonging to p_2 such that p belongs to $p_1 \parallel p_2$. Therefore, there exist accepting runs ρ_i over p_i. We have to now construct an accepting run ρ over p. If $pos(p_{(i,j)})$ $= (l_1, l_2)$, then define $\rho(i, j)$ to be $(\rho_1(i, l_1), \rho_2(i, l_2))$. Now from Definition 7 it is easy to verify that ρ is an accepting run.

If p is accepted by $(\mathcal{A}_1 \times \mathcal{A}_2)^{\epsilon_r}$, there is an accepting run ρ. By appropriate projections on p and ρ pictures p_i with accepting runs ρ_i can be for $\mathcal{A}_i$. □

As expected, commuting of independent actions in a picture that is accepted does not always result in an acceptable picture. This is because the standard commuting takes only one dimension into account. Here one has to also take into account the second dimension.

Consider shuffling the pictures $\begin{pmatrix} a\ a \\ b\ b \end{pmatrix}$ and $\begin{pmatrix} c\ c \\ b\ b \end{pmatrix}$ where a and c are independent. The picture $\begin{pmatrix} a & c & a & c \\ \epsilon_r & b & \epsilon_r & b \end{pmatrix}$ belongs to the shuffle. However no picture with the first row $a\ a\ c\ c$ can be in the shuffle. This is because the first b cannot be placed under the second a as it does not depend on the second a but depends on the first c.

However a limited form of the commuting lemma (taking into account the two dimensions) can easily be proven. This then identifies a class of essentially equivalent pictures.

Lemma 9. Let a product language L contain the picture $\begin{pmatrix} p_1 \\ w_1\ a\ b\ w_2 \\ w_3\ c\ d\ w_4 \\ w_5\ e\ f\ w_6 \\ p_2 \end{pmatrix}$ where p_1, p_2 belong to Σ^{**}, $w_1, w_2, w_3, w_4, w_5, w_6$ belong to Σ^*, $\{a, c, e\} \subseteq \Sigma_1$ and $\{b, d, f\} \subseteq \Sigma_2$ with cId. There exists p_1', p_2' belonging to Σ^{**}, w_2', w_4', w_6' belonging to Σ^* such that $\begin{pmatrix} p_1' \\ w_1\ a\ \ b\ \epsilon_r\ \epsilon_r\ w_2' \\ w_3\ \epsilon_r\ d\ \ c\ \epsilon_r\ w_4' \\ w_5\ \epsilon_r\ \epsilon_r\ e\ \ f\ w_6' \\ p_2' \end{pmatrix}$ belongs to L

Proof: We show that the causality requirements are not violated for one particular case. The other cases follow a similar argument. The exhibition of the action b does not alter the first component of the composite state. As c depends only on the first component of the sate associated it with a, placing it under the ϵ_r adjacent to b does not violate any constraint. Similarly, f depends only on d and the switching of d and c does not violate any causality constraint. The sub-pictures p'_1, w'_2, w'_4 and p'_2 are derived from p_1, w_2, w_4 and p_2 by adding suitable ϵ_rs. □

The above lemma shows that if the context around independent symbols is also constructed from independent symbols, the symbols can be commuted. In the example considered earlier, the context around the as and cs involve b which depends on both a and c. Note that we are not commuting a and b or e and f.

The lemma 9 may be useful in defining a product notion for tiling systems which are equivalent to 2OTAs which is under investigation.

However, the result in lemma 9 is far from complete as in specific instances limited commutation can be achieved. For example, if the picture $\begin{pmatrix} a & b\, w_1 \\ \epsilon_r & c\, w_2 \\ & p \end{pmatrix}$ where aIb and $c \in \Sigma_1 \cap \Sigma_2$ is accepted, then the picture $\begin{pmatrix} b & a\, w_1 \\ \epsilon_r & c\, w_2 \\ & p \end{pmatrix}$ will also be accepted. This is because c is dependent on both a and b and there is nothing that causally determines a and b.

3 Conclusion

We have studied the effect of taking the product of two 2OTAs. The key result is that one has to introduce blank spaces (or ϵ_rs) for the results to be combined in a meaningful way. These blank spaces stretch (or distort) the pictures so that the logical connections are maintained. A limited form of commutation is presented. Further work is required to obtain a complete characterisation of commutativity.

References

[Erwig and Meyer, 1995] Erwig, M. and Meyer, B. (1995). Heterogeneous visual languages –integrating textual and visual programming. In *Proc. 1995 IEEE Symposium on Visual Languages*, pages 318–325. IEEE Computer Society Press.

[Giammarresi and Restivo, 1997] Giammarresi, D. and Restivo, A. (1997). Two-dimensional languages. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages: Beyond Words*, pages 215–267. Springer-Verlag.

[Glasgow et al., 1995] Glasgow, J., Narayanan, N. H., and Chandrasekaran, B. (1995). *Diagrammatic Reasoning: Cognitive and Computational Perspectives*. AAAI Press.

[Marriott and Meyer, 1998] Marriott, K. and Meyer, B., editors (1998). *Visual Language Theory*. Springer.

[Mazurkiewicz, 1984] Mazurkiewicz, A. (1984). Traces, histories, graphs : Instances of a Process Monoid. In *Mathematical Foundations of Computer Science*, volume LNCS 176. Springer Verlag.

[Ratoandromanana and Robilliard, 1994] Ratoandromanana, B. and Robilliard, D. (1994). Superposition in Picture Languages. In *CAAP-94*, volume LNCS 787, pages 322–334, Edinburgh. Springer Verlag.

[Usher and Jackson, 1998] Usher, M. and Jackson, D. (1998). A Concurrent Visual Language Based on Petri-Nets. In *Proc. 1998 IEEE Symposium on Visual Languages*, pages 72–73. IEEE Computer Society Press.

[Zielonka, 1987] Zielonka, W. (1987). Notes on Finite Asynchronous Automata. *RAIRO: Theoretical Informatics and Applications*, 21(2):101–135.