Polynomial Time Algorithm for Inferring Subclasses of Parallel Internal Column Contextual Array Languages

Abhisek Midya[1(✉)], D.G. Thomas[2], Alok Kumar Pani[3], Saleem Malik[1], and Shaleen Bhatnagar[1]

[1] Information Technology, Alliance University, Bangalore 562106, India
abhisekmidyacse@gmail.com, baronsaleem@gmail.com, shaleenbhatnagar@gmail.com

[2] Department of Mathematics, Madras Christian College, Chennai 600059, India
dgthomasmcc@yahoo.com

[3] Computer Science and Engineering, Christ University Faculty of Engineering, Bangalore 560074, India
alok.kumar@christuniversity.in

Abstract. In [2,16] a new method of description of pictures of digitized rectangular arrays is introduced based on contextual grammars, called parallel internal contextual array grammars. In this paper, we pay our attention on parallel internal column contextual array grammars and observe that the languages generated by these grammars are not inferable from positive data only. We define two subclasses of parallel internal column contextual array languages, namely, k-uniform and strictly parallel internal column contextual languages which are incomparable and not disjoint classes and provide identification algorithms to learn these classes.

Keywords: Parallel internal column contextual array grammars · k-uniform · Identification in the limit from positive data

1 Introduction

In theoretical computer science, formal language theory is one of the fundamental areas. This study has its origin in Chomskian grammars. Contextual grammars which are different from Chomskian grammars, have been studied in [3,13] by formal language theorists, as they provide novel insight into a number of issues central to formal language theory. In a total contextual grammar, a context is adjoined depending on the whole current string. Two special cases of total contextual grammars, namely internal and external are very natural and have been extensively investigated. (External) Contextual grammars are introduced by S. Marcus in 1969 [13] with a linguistic motivation in mind. An external contextual grammar generates a language starting from a finite set of strings (the base) and iteratively adjoining to its contexts outside the current string.

In other families of contextual grammars, such as internal contextual grammars [13], the contexts are adjoined inside the current string.

There has been a great interest in adapting the techniques of formal string language theory for developing methods to study the problem of picture generation and description, where pictures are considered as connected, digitized finite arrays in the two-dimensional plane [15]. Recently, extensions of string contextual grammars to array structures and hyper graphs have been made in [1,2,6–8,11,12,14,16].

On the other hand, Grammatical Inference refers to the method of inferring a grammar (and possibly a target language) from data. Data can be text or informant. The difference between text and informant is that a text gives only positive examples (all strings do belong to the same language) where informant is both positive and negative examples. A learning procedure is an algorithm which is executed on a never-ending stream of inputs. The inputs are grammatical strings/arrays, taken from a target language which is in a known class of languages. The task is to identify a grammar that generates the target language. At each point in the process, any string is given as an input to the algorithm. After each input the algorithm produces a guess at the grammar which is eventually correct and could be unaltered when additional inputs are given. This model of learning is Gold's model of identification in the limit from positive data [5]. It is proved that no super finite language(it contains all finite languages and at least one infinite language) can be learn-able in the limit from positive examples. Hence, regular, context free, context sensitive grammars are not learn-able in the limit from positive examples only.

In this paper, we have introduced two subclasses of parallel internal column contextual array grammar, called, strictly parallel internal column contextual array grammar (SPICCAG), k-uniform parallel internal column contextual array grammar (k-UPICCAG) in order to find out identification algorithms. Our learning strategy is based on Gold's model.

2 Definition and Examples

If V is a finite alphabet, then V^* is the set of all strings including the empty string λ. An image or a picture over V is a rectangular $m \times n$ array of elements of V or in short $[a_{ij}]_{m\times n}$, the set of all images including the empty array Λ is denoted by V^{**}. A picture language or two dimensional language over V is a subset of V^{**}. In this paper Λ denotes any empty array. The notion of column concatenation is as follows: if X and Y are two arrays where

$$X = \begin{bmatrix} a_{1,j} & \dots & a_{1,k} \\ a_{2,j} & \dots & a_{2,k} \\ \dots & \dots & \dots \\ a_{l,j} & \dots & a_{l,k} \end{bmatrix}, Y = \begin{bmatrix} b_{1,m} & \dots & b_{1,n} \\ b_{2,m} & \dots & b_{2,n} \\ \dots & \dots & \dots \\ b_{l,m} & \dots & b_{l,n} \end{bmatrix} then, X\Phi Y = \begin{bmatrix} a_{1,j} & \dots & a_{1,k} & b_{1,m} & \dots & b_{1,n} \\ a_{2,j} & \dots & a_{2,k} & b_{2,m} & \dots & b_{2,n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{l,j} & \dots & a_{l,k} & b_{l,m} & \dots & b_{l,n} \end{bmatrix}$$

If L_1, L_2 are two picture languages over an alphabet Σ, the column concatenation $L_1\Phi L_2$ of L_1, L_2 is defined by $L_1\Phi L_2 = \{X\Phi Y \mid X \in L_1, Y \in L_2\}$. If X is an array, the set of all subarrays of X is denoted by $sub(X)$. We now recall the notion of column array context [2,16].

Definition 1. *Let V be an alphabet. A column array context c over V is of the form*

$$c = \left[\begin{smallmatrix} u_1 \\ u_2 \end{smallmatrix}\right] \psi \left[\begin{smallmatrix} v_1 \\ v_2 \end{smallmatrix}\right]$$

$\in V^{**}\psi V^{**}$ where u_1, u_2 are arrays of sizes $1 \times p$, and v_1, v_2 are arrays of sizes $1 \times q$, for some $p, q \geq 1$ and ψ is a special symbol not in V.

The next definition deals with parallel internal column contextual operation.

Definition 2. *Let V be an alphabet, C be a finite subset of $V^{**}\psi V^{**}$ whose elements are the column array contexts and $\varphi : V^{**} \to 2^C$ be mapping, called choice mapping.*

$$\textit{For an array } X = \begin{bmatrix} a_{1,j} & \cdots & a_{1,k} \\ a_{2,j} & \cdots & a_{2,k} \\ \cdots & \cdots & \cdots \\ a_{l,j} & \cdots & a_{l,k} \end{bmatrix},$$

$j \leq k, a_{ij} \in V$, we define $\hat{\varphi} : V^{**} \to 2^{V^{**}\psi V^{**}}$ such that $L\psi R \in \hat{\varphi}[X]$, where

$$L = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_l \end{bmatrix}, R = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_l \end{bmatrix},$$

and

$$c_i = \left[\begin{smallmatrix} u_i \\ u_{i+1} \end{smallmatrix}\right] \psi \left[\begin{smallmatrix} v_i \\ v_{i+1} \end{smallmatrix}\right] \in \varphi \left[\begin{smallmatrix} a_{i,j} & \dots a_{i,k} \\ a_{i+1,j} & \dots a_{i+1,k} \end{smallmatrix}\right],$$

with $c_i \in C, (1 \leq i \leq l-1)$, not all need to be distinct.

Given an array $X = [a_{ij}]$ of size $m \times n$, $a_{ij} \in V, X = X_1 \Phi X_2 \Phi X_3$ where

$$X_1 = \begin{bmatrix} a_{1,1} & \cdots & a_{1,p-1} \\ a_{2,1} & \cdots & a_{2,p-1} \\ \vdots & \vdots & \vdots \\ a_{m1} & \cdots & a_{m,p-1} \end{bmatrix}, X_2 = \begin{bmatrix} a_{1,p} & \cdots & a_{1,q} \\ a_{2,p} & \cdots & a_{2,q} \\ \vdots & \vdots & \vdots \\ a_{m,p} & \cdots & a_{m,q} \end{bmatrix}, X_3 = \begin{bmatrix} a_{1,q+1} & \cdots & a_{1,n} \\ a_{2,q+1} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots \\ a_{m,q+1} & \cdots & a_{m,n} \end{bmatrix}$$

and $1 \leq p \leq q \leq n$, we write $X \Rightarrow Y$ if $Y = X_1 \Phi L \Phi X_2 \Phi R \Phi X_3$ such that $L\psi R \in \hat{\varphi}[X_2]$. Here L and R are called left and right contexts respectively. We say that Y is obtained from X by parallel internal column contextual operation $(\Rightarrow_{in})$.

Now we consider the notion of parallel internal column contextual array grammar [2,16].

Definition 3. *A parallel internal column contextual array grammar is an ordered system $G = (V, A, C, \varphi)$ where V is an alphabet, A is a finite subset of V^{**} called the axiom set, C is a finite subset of $V^{**}\psi V^{**}$ called column array contexts, $\varphi : V^{**} \to 2^C$ is the choice mapping which performs the parallel internal column contextual operation. When φ is omitted we call G as a parallel internal contextual array grammar without choice.*

For any $X, Y \in V^{**}, X \Rightarrow Y$ if and only if $X = X_1 \Phi X_2 \Phi X_3, Y = X_1 \Phi L \Phi X_2 \Phi R \Phi X_3$ with $L\psi R \in \hat{\varphi}[X_2]$. We denote by $\Rightarrow^*$ the reflexive transitive closure of $\Rightarrow_{in}$. Then the parallel internal column contextual array language generated by the parallel internal column contextual array grammar G is defined as the set $L_{in}(G) = \{Y \in V^{**} / \exists X \in A \text{ such that } X \Rightarrow^* Y\}$.

3 Subclasses of Parallel Internal Column Contextual Array Grammars

In this paper our main focus is on designing an identification algorithm to infer parallel internal column contextual array grammar. According to Gold model [5], no superfinite class of languages is inferable from positive data only. A class of languages that consists of all finite languages and atleast one infinite language, is called a super finite class of languages.

Proposition 1. *The class of parallel internal column contextual array languages* ($PICCAL$)*, is not inferable from positive data only.*

Proof. In the case of string languages, the class of internal contextual languages, is not inferable from positive data only [4]. From this fact, we can conclude Theorem 1.

As we know that the class ($PICCAL$) is not inferable from positive data only, it is natural to look for subclasses of these languages which can be identified in the limit from positive data only. We now define strictly parallel internal column contextual array grammar ($SPICCAG$) and k-uniform parallel internal column contextual array grammar ($k-UPICCAG$).

Definition 4. *A strictly parallel internal column contextual array grammar* ($SPICCAG$) *is a* 6 *tuple* $G = (V, X, C, \varphi, P, A)$ *where*

- V *is the alphabet.*
- X *is a finite subset of* V^{**}*, called selector set and* C *is a finite subset of* $V^{**}\psi V^{**}$*, called context set.*
- $\varphi : V^{**} \to 2^C$ *is a choice mapping.*
- P *is a finite set of parallel internal column contextual rules of the form,* $\varphi[x_i] = L_i\psi R_i$ *where* $L_i, R_i \in C$ *are the ith left and right context of ith selector* $x_i \in X$*,* L_i, R_i *have same number of rows.*
- $first[L_i] \neq first[R_i]$ *where* $first[W]$ *denotes the first column of* W *and* L_i *is not a subarray of* R_i *and vice versa.*
- A *is a finite subset of* V^{**}*, called the axiom set.*
- *for each selector, there is exactly one rule.*

The language generated by strictly parallel internal column contextual array grammar ($SPICCAG$) *is called a strictly parallel internal column contextual array language (SPICCAL) which is* $L_{sin}(G) = \{Y \in V^{**} \mid Q \Rightarrow^* Y, Q \in A\}$.

3.1 Example

Let $G = (V, X, C, \varphi, P, A)$ be a strictly parallel internal column contextual array grammar ($SPICCAG$) where $V = \{a, b\}$,

$$X = \left\{ \begin{bmatrix} a & b \\ b & a \end{bmatrix}, \begin{bmatrix} a & b \\ a & b \end{bmatrix}, \begin{bmatrix} b & a \\ b & a \end{bmatrix} \right\}, C = \left\{ \begin{bmatrix} a \\ b \end{bmatrix} \psi \begin{bmatrix} b \\ a \end{bmatrix}, \begin{bmatrix} a \\ a \end{bmatrix} \psi \begin{bmatrix} b \\ b \end{bmatrix}, \begin{bmatrix} b \\ b \end{bmatrix} \psi \begin{bmatrix} a \\ a \end{bmatrix} \right\}$$

φ is a choice mapping

$$P = \left\{\varphi \begin{bmatrix} a & b \\ b & a \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} \psi \begin{bmatrix} b \\ a \end{bmatrix}, \varphi \begin{bmatrix} a & b \\ a & b \end{bmatrix} = \begin{bmatrix} a \\ a \end{bmatrix} \psi \begin{bmatrix} b \\ b \end{bmatrix}, \varphi \begin{bmatrix} b & a \\ b & a \end{bmatrix} = \begin{bmatrix} b \\ b \end{bmatrix} \psi \begin{bmatrix} a \\ a \end{bmatrix}\right\},$$

$$A = \left\{ Q = \begin{bmatrix} a & a & b & b \\ a & a & b & b \\ b & b & a & a \\ b & b & a & a \end{bmatrix} \right\}$$

Here for each rule $\varphi[x_i] = L_i \psi R_i$, $first(L_i) \neq first(R_i), i \geq 1$, so it does satisfy Definition 4. Clearly, $L_{sin}(G) = \left\{ \begin{bmatrix} (a^n \ b^n)_m \\ (b^n \ a^n)_m \end{bmatrix} \mid n \geq 2, m = 2 \right\}$ Here $a^n = aaa...a$ (n times) and $a_m = \begin{matrix} a \\ \vdots \\ a \end{matrix}$, m rows are there. A simple derivation of a member of $L_{sin}(G)$ is as follows,

$$Q = \begin{bmatrix} a & a & b & b \\ a & a & b & b \\ b & b & a & a \\ b & b & a & a \end{bmatrix} \Rightarrow \begin{bmatrix} a & a & a & b & b & b \\ a & a & a & b & b & b \\ b & b & b & a & a & a \\ b & b & b & a & a & a \end{bmatrix} = \begin{bmatrix} (a^3 \ b^3)_2 \\ (b^3 \ a^3)_2 \end{bmatrix} \in L_{sin}(G).$$

Definition 5. *A k-uniform parallel internal column contextual array grammar is a 6-tuple* $(k - UPICCAG)$, $k \geq 1, G = (V, X, C, \varphi, P, A)$ *where*

- V *is the alphabet.*
- X *is a finite subset of* V^{**}, *called selector set and* C *is a subset of* $V^{**}\psi V^{**}$, *called context set.*
- $\varphi : V^{**} \rightarrow 2^C$ *is a choice mapping.*
- P *is a finite set of parallel internal column contextual array rules of the following form,* $\varphi[x_i] = L_i \psi R_i$ *where* $L_i, R_i \in C$ *are the ith left and right context of ith selector* $x_i \in X$, L_i, R_i *have same number of rows.*
- A *is the finite subset of* V^{**}, *called axiom set. Each member of* A *is an axiom which contains mk number of columns, for some* $m \geq 1$ *and we put the following restrictions.*

If the rule is $\varphi[x] = L\psi R$ *then,*

- $|x| = |L| = |R| = k$, *where* $|W|$ *denotes the number of columns in an array* W.
- *for each selector, there is exactly one rule.*

The language generated by $k - UPICCAG$ *is called a k-uniform parallel internal column contextual array language(k-UPICCAL) which is* $L_{k-uin}(G) = \{Y \in V^{**} \mid Q \Rightarrow^* Y, Q \in A\}$.

3.2 Example of 2-UPICCAG

$G = (V, X, C, \varphi, P, A)$ is a 2-UPICCAG where $V = \{a, b\}$,

$$X = \left\{ \begin{bmatrix} a & b \\ b & b \end{bmatrix}, \begin{bmatrix} b & b \\ a & b \end{bmatrix} \right\}, C = \left\{ \begin{bmatrix} a & b \\ b & a \end{bmatrix} \psi \begin{bmatrix} a & b \\ b & a \end{bmatrix}, \begin{bmatrix} b & a \\ a & b \end{bmatrix} \psi \begin{bmatrix} b & a \\ a & b \end{bmatrix} \right\},$$

φ is a choice mapping,

$$P = \left\{ \varphi \begin{bmatrix} a & b \\ b & b \end{bmatrix} = \begin{bmatrix} a & b \\ b & a \end{bmatrix} \psi \begin{bmatrix} a & b \\ b & a \end{bmatrix}, \varphi \begin{bmatrix} b & b \\ a & b \end{bmatrix} = \begin{bmatrix} b & a \\ a & b \end{bmatrix} \psi \begin{bmatrix} b & a \\ a & b \end{bmatrix} \right\},$$

$A = \left\{ Q = \left[\begin{smallmatrix} a & b & a & b \\ b & b & b & b \\ a & b & a & b \end{smallmatrix}\right] \right\}$. Here, $|x| = |L| = |R| = 2, m = 2$, number of columns in $A = mk = 4$. So it satisfies Definition 5. Clearly

$$L_{k-uin}(G) = \left\{ A, \begin{bmatrix} (ab)^{n-1} & ab & (ab)^{n-1} & ab \\ (ba)^{n-1} & bb & (ba)^{n-1} & bb \\ (ab)^{n-1} & ab & (ab)^{n-1} & ab \end{bmatrix} \mid n \geq 2 \right\}$$

$$\text{For instance, } \begin{bmatrix} a & b & a & b \\ b & b & b & b \\ a & b & a & b \end{bmatrix} \Rightarrow \begin{bmatrix} a & b & a & b & a & b & a & b \\ b & a & b & b & b & a & b & b \\ a & b & a & b & a & b & a & b \end{bmatrix} \in L_{k-uin}(G).$$

Now, if we consider a = black box and b = white box, we get a nice rectangular picture.

Theorem 1. *$\mathcal{L}_{\mathcal{SPICCAG}}$ is incomparable with $\mathcal{L}_{\mathcal{K}-\mathcal{UPICCAG}}$ and not disjoint.*

Proof. We prove this theorem using following lemmas whose proofs are omitted. □

Lemma 1. $\mathcal{L}_{\mathcal{SPICCAG}} - \mathcal{L}_{\mathcal{K}-\mathcal{UPICCAG}} \neq \phi$

Lemma 2. $\mathcal{L}_{\mathcal{K}-\mathcal{UPICCAG}} - \mathcal{L}_{\mathcal{SPICCAG}} \neq \phi$

Lemma 3. $\mathcal{L}_{\mathcal{K}-\mathcal{UPICCAG}} \cap \mathcal{L}_{\mathcal{SPICCAG}} \neq \phi$

4 Identification of Subclasses of Parallel Internal Column Contextual Array Languages

In this section, we propose an algorithm to infer $SPICCAG$ from positive data only. We recall the notion of an insertion rule. The insertion operation is first considered by Haussler in [9] and based on the operation, insertion systems are introduced by L. Kari in [10]. Informally, if a string α is inserted between two parts w_1 and w_2 of a string $w_1 w_2$ to get $w_1 \alpha w_2$, we call the operation as insertion.

This algorithm takes finite sequences of positive examples in the different time interval or all together. Our goal is to find out $SPICCAG$ G, such that $IP \subseteq L(G)$ where IP is the input set of arrays. The algorithm works in the following way. After receiving the first set of arrays as an input, based on the size(actually based on number of columns), firstly the algorithm determines the axiom, then it defines 2D insertion rules in order to find out context and selector from input example. After that, insertion rules are converted into 1-sided[1] contextual rules which will be a guess about the unknown grammar. Then we will convert 1-sided contextual rule into 2-sided contextual rule to take care of over generalization. Then updates with new contextual rules if the next input array cannot be generated by the existing contextual rules. All the guessing will be done in a flexible way in the sense that the correction is done at every instance. Finally we will find the parallel internal column contextual rules according to Definition 2.

In this paper we consider single axiom A and finite selector set. Now, we present our algorithm with a description for better understanding.

[1] In an 1-sided contextual rule either left context is Λ or right context is Λ.

5 Pseudocode of Our Algorithm

```
1: $axiom \leftarrow Find - Smallest(IPS)$
2: $inser \leftarrow Generate - Inser(axiom, IP_i)$
3: $1 - Sided - Contextual - Rule \leftarrow \{\}$
4: $1 - Sided - Correct - Rule \leftarrow \{\}$
5: $2 - Sided - Correct - Rule \leftarrow \{\}$
6: $Parallel - Rule \leftarrow \{\}$
7: $Table \leftarrow \sqcap$
8: $1 - Sided - Contextual - Rule.push[Convert - into - Contextual - Rule(inser)]$
9: $IPS \leftarrow Remove(IPS, IP_i)$
10: for $(1 - Sided - Contextual - Rule_i \in \{1 - Sided - Contextual - Rule\})$ do
11:     for $(IP_i \in \{IPS\})$ do
12:         $S \leftarrow Check - Contextual - Rule(1 - Sided - Contextual - Rule_i, IP_i)$
13:         if $S = 1$ then
14:             $1 - Sided - Correct - Rule.push[1 - Sided - Contextual - Rule_i]$
15:         if $S = 0$ then
16:             $1 - Sided - Correct - Rule.push[Correction - Contextual - Rule(1 - Sided - Contextual - Rule_i, IP_i)]$
17: for $(1 - Sided - Correct - Rule_i \in \{1 - Sided - Correct - Rule\})$ do
18:     for $(IP_i \in \{IPS\})$ do
19:         $Table.insert[Find - Nof - App - of - EachRule - in - EachMember(1 - Sided - Correct - Rule_i, IP_i)]$
20: if $TableRow_i = TableRow_j$ then
21:     $2 - Sided - Correct - Rule.push[Merge(1 - Sided - Correct - Rule_i, 1 - Sided - Correct - Rule_j)]$
22: for $(2 - Sided - Correct - RULE_i \in \{2 - Sided - Correct - Rule\})$ do
23:     $Parallel - Rule.push(2 - Sided - Correct - Rule_i)$
```

In the next few subsections we will explain all the steps of our pseudocode in detail.

5.1 Finding Axiom - Pseudocode-Step: 1

axiom ← Find − Smallest(IPS): It finds the smallest array from the IPS (input set). The output of the function will be considered as an axiom.

In order to find out the axiom, the number of columns of each array is evaluated, the array with the smallest number of columns, will be considered as the axiom. Also a new input array will be compared with the existing axiom based on the number of columns, and the smaller one will be considered as an axiom and Let the single axiom be denoted by A.

5.2 Defining Insertion Rule and Converting It into Contextual Rule - Pseudocode-Step: 2, 8, 9

- $\mathbf{insr} \leftarrow \mathbf{Generate} - \mathbf{Inser}(\mathbf{axiom}, \mathbf{IP_i})$**:** It generates the insertion rule from axiom and member of input set (IP_i). The output of the function will be stored in *insr* as an insertion rule.
- $\mathbf{1{-}Sided{-}Contextual{-}Rule.push} \leftarrow [\mathbf{Convert} - \mathbf{into} - \mathbf{Contextual{-}Rule}(\mathbf{inser})]$**:** It converts *insr* into $1 - Sided - Contextual - Rule$ and store that.
- $\mathbf{IPS} \leftarrow \mathbf{Remove}(\mathbf{IPS}, \mathbf{IP_i})$**:** It removes the current input member IP_i from IPS.
- We now shortly describe about the intuitive idea of the parts 1–4. We try to identify the selectors from the axiom and contexts from examining input.
- Let the format of 2D insertion rule be LIR where $L, I, R \in V^{++}$ are left context, inserted portion, and right context respectively. Axiom and examining array are respectively

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ a_{2,1} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix}, E = \begin{bmatrix} a_{1,1} & \cdots & a_{1,p} \\ a_{2,1} & \cdots & a_{2,p} \\ \vdots & \vdots & \vdots \\ a_{m,1} & \cdots & a_{m,p} \end{bmatrix}$$

Let the initial insertion rule be LIR and from the axiom we can have the following consideration:

Part 1:

$$L = \begin{bmatrix} a_{1,1} \\ a_{2,1} \\ \vdots \\ a_{m,1} \end{bmatrix}, R = \begin{bmatrix} a_{1,2} & \cdots & a_{1,n} \\ a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots \\ a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

Check whether any $I = [I_{i,j}]_{m \times r}$ where $r \le p$ exists with $LIR \in sub(E)$ or not. If yes then fix that $I = [I_{i,j}]_{m \times r}$ and go to part 3, else go to part 2.

Part 2: Remove the last column of the right context R and the rule becomes LIR where

$$L = \begin{bmatrix} a_{1,1} \\ a_{2,1} \\ \vdots \\ a_{m,1} \end{bmatrix}, R = \begin{bmatrix} a_{1,2} & \cdots & a_{1,n-1} \\ a_{2,2} & \cdots & a_{2,n-1} \\ \cdots & \cdots & \cdots \\ a_{m,2} & \cdots & a_{m,n-1} \end{bmatrix}$$

Check whether any $I = [I_{i,j}]_{m \times r}$ where $r \le p$ exists with $LIR \in sub(E)$ or not. If yes then fix that $I = [I_{i,j}]_{m \times r}$ and go to part 3, else go to part 2 recursively, until $L = \begin{bmatrix} a_{1,1} \\ a_{2,1} \\ \vdots \\ a_{m,1} \end{bmatrix}, R = \begin{bmatrix} a_{1,2} \\ a_{2,2} \\ \vdots \\ a_{m,2} \end{bmatrix}$, and then go to part 4.

Part 3 - Conversion of 2D insertion rule into 1 sided 2D contextual rule: Here L^{IN}, I^{IN}, R^{IN} are left context, inserted portion, and right context for insertion rule respectively. On the other hand, L^{IC}, x^{IC}, R^{IC} are left context, selector, and right context for internal contextual rule respectively.

$(LIR)^{IN} \rightarrow (\hat{\varphi}[x] = L\psi R)^{IC}$ where $x^{IC} = L^{IN}, L^{IC} = \Lambda, R^{IC} = I^{IN}$. Once we get a selector and associated context with it, we have the following conditions for each 2D insertion rule:

- **Condition 1:** If $(|L|+|I|+|R|)^{IN} = |E|$, it implies that on this current axiom A, only one rule has been applied and we obtain the rule.
- **Condition 2:** If $(|L| + |R|)^{IN} \leq |A|$, then we remove L^{IN} from axiom A, and obtain a new temporary axiom, also consider R^{IN} as a L^{IN} for the next insertion rule. Also we remove $(LI)^{IN}$ as a subarray from the examining input E and obtain a new temporary input. Now we continue our procedure with this temporary axiom and temporary examing input in the same way.
- **Condition 3:** If $(|L|+|I|+|R|)^{IN} \leq |E|$ but $(|L|+|R|)^{IN} = |A|$, then it can be understood that some part of the examining input is still left to scan, and that is considered directly as the left context L^{IC} of the first selector x^{IC}_{first} or right context R^{IC} of the last selector x^{IC}_{last}. We define new rule internal contextual rule.
- $(\hat{\varphi}[x] = L\psi R)_{new}$ where $L_{new} = L^{IC}, R_{new} = \Lambda, x_{new} = x^{IC}_{first}$, another rule can be $(\hat{\varphi}[x] = L\psi R)_{new}$ where $L_{new} = \Lambda, R_{new} = R^{IC}, x_{new} = x^{IC}_{last}$. It should be noted that these particular rules will not be considered for updation and correction.

Part 4: At that moment, existing first column of R will be concatenated with existing L.

$$L = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ \vdots & \vdots \\ a_{m,1} & a_{m,2} \end{bmatrix}, R = \begin{bmatrix} a_{1,3} & \cdots & a_{1,n} \\ a_{2,3} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots \\ a_{m,3} & \cdots & a_{m,n} \end{bmatrix}, \text{ go to part 1 until } L = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ a_{2,1} & \cdots & a_{2,n} \\ \vdots & \vdots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix},$$

in that case defining insertion rule is not possible. We may need to define insertion rule with the current examining array, if we are still unable to define insertion rule, then we will conclude that the choosen axiom is wrong. It is a negative example as we are dealing with single axiom.

So in this section, we get the selectors from axiom and contexts from examining input. Later on for new input, we may need to guess (next section).

5.3 Making Correction and Updating Rules - Pseudocode-Step: 10–16

- $\mathbf{S \leftarrow Check-Contextual-Rule(1-Sided-Contextual-Rule_i, IP_i)}$: It checks the correctness of $1 - Sided - Contextual - Rule_i$ for IP_i. If S is true then the correct $1 - Sided - Contextual - Rule_i$ will be pushed onto set $\{1 - Sided - Correct - Rule\}$ or it goes for correction.
- $\mathbf{Correction-Contextual-Rule(1-Sided-Contextual-Rule_i, IP_i)}$: In that case we need to go for correction of the rule in such a way so that our new corrected rule can take care of new inputs and as well as previous inputs.
- Let the initial rule be $\hat{\varphi}[x_i] = L_i\psi R_i$ where L_i, R_i are ith left and right context of the ith selector x_i. Here x_{i+1} is also introduced because we will make the correction using x_{i+1}.

Proposition 2. *In case of correction, we deal with only 1-sided contextual rules where left context is always empty and selector is not the last one. (see condition*

3 of Subsect. 5.2) We will try to find the rule as a subarray from the examining input.

Let the examining input be $E = \begin{bmatrix} a_{i,1} & \cdots & a_{i,p} \\ a_{i+1,1} & \cdots & a_{i+1,p} \\ \vdots & \vdots & \vdots \\ a_{m,1} & \cdots & a_{m,p} \end{bmatrix}$. We can represent the examining input in the following format $E = P\Phi x_i \Phi Q \Phi x_{i+1} \Phi Z$. where P, Z are the rest of the part of string and they can be empty also, Q is the inserted subarray portion. Now we present the examining input in 2D form.

$$E = P\Phi \begin{bmatrix} a_{i,k} & \cdots & a_{i,\alpha} \\ a_{i+1,k} & \cdots & a_{i+1,\alpha} \\ \vdots & \vdots & \vdots \\ a_{m,k} & \cdots & a_{m,\alpha} \end{bmatrix} \Phi Q \Phi \begin{bmatrix} a_{i,j} & \cdots & a_{i,\beta} \\ a_{i+1,j} & \cdots & a_{i+1,\beta} \\ \vdots & \vdots & \vdots \\ a_{m,j} & \cdots & a_{m,\beta} \end{bmatrix} \Phi Z$$

Now we need to check the contexts. R must be matched with Q. $R = R_i \Phi R_{i+1} \Phi ... \Phi R_w$ where $1 \leq i \leq w$, and R_i presents the ith column of array. $Q = Q_i \Phi Q_{i+1} \Phi ... \Phi Q_z$ where $1 \leq i \leq z$, and Q_i presents the ith column of array.

Here we are making an analysis to find out the partially equal part (as a prefix/suffix) between $R_1 \Phi R_2 \Phi ... \Phi R_w$ and $Q_1 \Phi Q_2 \Phi ... \Phi Q_z$ and we have shown the correction part for one rule, in the same way can make the correction for other rules. In Theorems 3 and 4, we obtain the common-prefix and common-suffix part between R and Q.

Theorem 2. *If the analysis starts with equality such that $Q_1 = R_1, Q_2 = R_2 \Phi ... \Phi Q_f = R_s$, and $Q_{f+1} \neq R_{s+1}$ or $f = z$ or $s = w$, then we can have four different types of errors which are stated in terms of following lemmas.*

Lemma 4. *If ($f = z$ and $s = w$) then it implies that matching is correct, so no need to make any correction for this rule and the rule is correct.*

Lemma 5. *If ($f = z$ and $s < w$) then we infer the following two new rules.*

- *$Rule_{i'} : \hat{\varphi}[x_{i'}] = L_{i'} \psi R_{i'}$ where $R_{i'} = Q_1 \Phi Q_2 \Phi ... \Phi Q_f, L_{i'} = \Lambda, x_{i'} = x_i$.*
- *$Rule_{(i+1)'} : \hat{\varphi}[x_{(i+1)'}] = L_{(i+1)'} \psi [R_{(i+1)'}$ where $L_{(i+1)'} = R_{s+1} \Phi R_{s+2} \Phi ... \Phi R_w, R_{(i+1)'} = \Lambda, x_{(i+1)'} = x_{(i+1)}$.*

Lemma 6. *If ($f < z$ and $s = w$) then we infer the following two new rules.*

- *$Rule_{i'} : \hat{\varphi}[x_{i'}] = L_{i'} \psi R_{i'}$ where $R_{i'} = R_1 \Phi .. \Phi R_w, L_{i'} = \Lambda, x_{i'} = x_i$.*
- *$Rule_{(i+1)'} : \hat{\varphi}[x_{(i+1)'}] = L_{(i+1)'} \psi R_{(i+1)'}$ where $L_{(i+1)'} = Q_{f+1} \Phi Q_{f+2} \Phi ... \Phi Q_z$, $R_{(i+1)'} = \Lambda, x_{(i+1)'} = x_{(i+1)}$.*

Lemma 7. *If ($f < z$ and $s < w$) then we infer the following three new rules.*

- *$Rule_{i'} : \hat{\varphi} x_{i'} = L_{i'} \psi R_{i'}$ where $R_{i'} = Q_1 \Phi Q_2 \Phi ... \Phi Q_f, L_{i'} = \Lambda, x_{i'} = x_i$.*
- *$Rule_{(i+1)'} : \hat{\varphi}[x]_{(i+1)'} = L_{(i+1)'} \psi R_{(i+1)'}$ where $L_{(i+1)'} = R_{s+1} \Phi ... \Phi R_w, R_{(i+1)'} = \Lambda, x_{(i+1)'} = x_{i+1}$.*
- *$Rule_{(i+2)'} : \hat{\varphi}[x]_{i+2} = L_{(i+2)'} \psi R_{(i+2)'}$ where $L_{(i+2)'} = Q_{f+1} \Phi ... \Phi Q_z$, $R_{(i+2)'} = \Lambda, x_{(i+2)'} = x_{i+1}$.*

Theorem 3. *If the analysis starts with inequality such that* $Q_1 \neq R_1$*, but* $Q_z = R_w, Q_{z-1} = R_{w-1}\Phi ...\Phi Q_f = R_s$*, and* $Q_{f-1} \neq R_{s-1}$ *then we can have three different types of errors which can be seen in the following lemmas.*

Lemma 8. *If* $(s = 1, f > 1)$ *then we infer the following two new rules.*

- $Rule_{i'} : \hat{\varphi}[x_{i'}] = L_{i'}\psi R_{i'}$ *where* $L_{i'} = R_1\Phi R_2\Phi ...\Phi R_w, R_{i'} = \Lambda, x_{i'} = x_{i+1}$.
- $\hat{\varphi}[x_{(i+1)'}] = L_{(i+1)'}\psi R_{(i+1)'}$ *where* $R_{(i+1)'} = Q_1\Phi Q_2\Phi ...\Phi Q_{f-1}, L_{(i+1)'} = \Lambda, x_{(i+1)'} = x_i$.

Lemma 9. *If* $(s > 1)$ *then we infer the following three new rules.* $Rule_{i'} : \hat{\varphi}[x_{i'}] = L_{i'}\psi R_{i'}$ *where* $L_{i'} = R_s\Phi R_{s+1}\Phi ...\Phi R_w, R_{i'} = \Lambda, x_{i'} = x_{i+1}$. $Rule_{(i+1)'} : \hat{\varphi}[x_{(i+1)'}] = L_{(i+1)'}\psi R_{(i+1)'}$ *where* $R_{(i+1)'} = Q_1\Phi Q_2\Phi ...\Phi Q_{f-1}, L_{(i+1)'} = \Lambda, x_{(i+1)'} = x_i$. $Rule_{(i+2)'} : \hat{\varphi}[x_{(i+2)]'}] = L_{(i+2)'}\psi R_{(i+2)'}$ *where* $L_{(i+2)'} = \Lambda, R_{(i+2)'} = R_1\Phi R_2\Phi ...\Phi R_{s-1}, x_{(i+2)'} = x_i$.

Lemma 10. *If* $Q_z \neq R_w$ *then we infer the following two new rules.*

- $Rule_{i'} : \hat{\varphi}[x_{i'}] = L_{i'}\psi R_{i'}$ *where* $R_{i'} = R_1\Phi R_2\Phi ...\Phi R_w, L_{i'} = \Lambda, x_{i'} = x_i$.
- $Rule_{(i+1)'} : \hat{\varphi}[x_{(i+1)'}] = L_{(i+1)'}\psi R_{(i+1)'}$ *where* $R_{(i+1)'} = Q_1\Phi Q_2\Phi ...\Phi Q_z$, $L_{(i+1)'} = \Lambda, x_{(i+1)'} = x_i$.

In this section, we must notice that we have different rules with same selectors. According to Definitions 4 and 5, for each selector there must be one rule. As we are inferring 1-sided contextual rule, it does not satisfy our Definitions 4 and 5. In the next section we will convert 1-sided contextual rule into 2-sided contextual rule in order to take care of over generalization and Definitions 4 and 5.

5.4 Controlling over Generalization - Pseudocode-Step: 17–21

- $\mathbf{Table.insert[Find - Nof - App - of - EachRule - in - EachMember}$ $\mathbf{(1 - Sided - Correct - Rule_i, IP_i)]:}$ It finds out the application of each rule on each member of the input and insert that record into the table.
- $\mathbf{2 - Sided - Correct - Rule.push[Merge(1 - Sided - Correct - Rule_i,}$ $\mathbf{1 - Sided - Correct - Rule_j)]:}$ In this case if we find that ith row ($Table\ Row_i$) and jth row ($TableRow_j$) is same then we merge these two rules ($1 - Sided - Correct - Rule_i, 1 - Sided - Correct - Rule_j$) and store as a $2 - Sided - Correct - Rule$.
- In this section we determine the number of applications of each rule to generate the given input set. It will be presented in table. We put priority in applying rules where left context is empty and context is smaller in size. If it is found that without using any rule we can generate the full input set then we can ignore that rule.
- Actually all the rules are 1-sided where left contexts or right contexts are empty that generate more elements. Thus, to control this over generalization, we check that how many times each rule is applied in each member of the input set. Rules which are applied equal number of times in each member, those can be merged into one rule based on condition (discussed in Lemmas 11 and 12).

- Also in this way we satisfy our required condition for $SPICCAG$ (Definition 4), that is, for each selector atmost one rule is applicable.

Lemma 11. *If consecutive selectors are x_i, x_j with $(j - i) = 1$ and left contexts(right contexts) are empty in a set of rule then we can get 1-sided or 2-sided internal contextual rule after merging them.*

Proof. Let x_i, x_j denote ith and jth selector, R_i, R_j be ith and jth right context and L_i, L_j are ith and jth left context.

- **case 1:** If x_i, x_j are such that $(j - i) = 1$ and if $R_i = R_j = \Lambda$ then rule becomes $\hat{\varphi}[x_i] = L_i \psi R_i$ where $R_i = L_j$.
- **case 2:** If x_i, x_j are such that $(j - i) = 1$ and if $L_i = L_j = \Lambda$ then the rule becomes $\hat{\varphi}[x_i] = L_i \psi R_i$ where $L_j = R_i, x_i = x_j$.

Lemma 12. *If consecutive selectors are x_i, x_j with $(j - i) = 1$ and left contexts of ith rule and right context of jth rule are empty then we can get 1-sided internal contextual rule after merging them.*

Proof. Let x_i, x_j denote ith and jth selector, R_i, R_j are left contexts of ith rule and right context of jth rule respectively.

If x_i, x_j are such that $(j - i) = 1$ and if $L_i = R_j = \Lambda$ then the rule becomes $\hat{\varphi}[x_i] = L_i \psi R_i$ where $R_i = R_i \Phi R_j$

5.5 Parallalization Contextual Array Rules - Pseudocode-Step: 22, 23

- $\mathbf{Parallel - Rule.push(2 - Sided - Correct - Rule_i)}$**:** It converts the $2-Sided-Correct-Rule_i$ into parallel rule and push onto set $\{Parallel-Rule\}$. If we get a rule $\hat{\varphi}[x] = L\psi R$ where

$$x = \begin{bmatrix} a_{i,k} & \cdots & a_{i,\alpha} \\ a_{i+1,k} & \cdots & a_{i+1,\alpha} \\ \vdots & \vdots & \vdots \\ a_{m,k} & \cdots & a_{m,\alpha} \end{bmatrix}, L = \begin{bmatrix} a_{i,j} & \cdots & a_{i,k-1} \\ a_{i+1,j} & \cdots & a_{i+1,k-1} \\ \vdots & \vdots & \vdots \\ a_{m,j} & \cdots & a_{m,k-1} \end{bmatrix}, R = \begin{bmatrix} a_{i,\alpha+1} & \cdots & a_{i,n} \\ a_{i+1,\alpha+1} & \cdots & a_{i+1,n} \\ \vdots & \vdots & \vdots \\ a_{m,\alpha+1} & \cdots & a_{m,n} \end{bmatrix}$$

According to Definition 2, we can have $(m-1)$ parallel rules $\varphi[Px_i] = PL_i \psi PR_i$ where Px_i, PL_i, PR_i are respectively selector, left context, right context.

$$Px_i = \begin{bmatrix} a_{i,k} & \cdots & a_{i,\alpha} \\ a_{i+1,k} & \cdots & a_{i+1,\alpha} \end{bmatrix}, PL_i = \begin{bmatrix} a_{i,j} & \cdots & a_{i,k-1} \\ a_{i+1,j} & \cdots & a_{i+1,k-1} \end{bmatrix}, PR_i = \begin{bmatrix} a_{i,\alpha+1} & \cdots & a_{i,n} \\ a_{i+1,\alpha+1} & \cdots & a_{i+1,n} \end{bmatrix}$$

where $1 \le i \le m - 1$.

Remark 1. The above algorithm can also be used to identify a $k - UPICCAG$. A modification required in the algorithm is that, k is also given along with the positive presentation as an input to the algorithm.

In this case, at the time of defining insertion rule (Sect. 5.2), we need to focus on the size of selectors and contexts in terms of number of columns as k is given as an input. Defining insertion rule should be done in the following way, $LIR \in sub(E)$ where $|I| = |L| = |R| = k$ and also $|A| = mk, L, I, R \in V^{++}$.

6 Correctness of the Algorithm and Characteristic Sample

The correctness of the algorithm can be noticed in view of the fact that the specific properties of the subclasses considered allow the positive examples. The correctness of the algorithm can be seen by considering a characteristic sample for a target language. Also it can be seen that the algorithm runs in polynomial time in the sum of the size of the examples given. (discussed in Sect. 7). The correctness of the algorithm, can be seen by considering a characteristic sample for a target $SPICCAL$. Let L be an $SPICCAL$. A finite set IPS is called a characteristic sample of L if and only if L is the smallest $SPICCAL$ containing IPS.

7 Running Time Complexity of Our Algorithm

In this section we show the running time of our algorithm to infer the column contextual rules.

Theorem 4. *The running time complexity of the given pseudocode in Sect. 5, is polynomial in the size of the input set, that is,* $SumofSize(IPS)$ *where* $IPS = \{IP_i, IP_{i+1}, ..., IP_k\}$.

Proof. proof is omitted.

8 Conclusion and Future Work

In this paper we present a polynomial time algorithm to infer subclasses of parallel internal column contextual array languages from positive examples only. Here we deal with only column contextual rules. In the form of future direction of this work, we can deal with column and row contextual rules together, that is, parallel internal array contextual languages.

References

1. Chandra, H., Martin-Vide, C., Subramanian, K.G., Van, D.L., Wang, P.S.P.: Parallel contextual array grammars and trajectories. In: Chen, C.H., Wang, P.S.P. (eds.) Handbook of Pattern Recognition and Computer Vision, 3rd edn., pp. 55-70 (2004)
2. Chandra, H., Subramanian, K.G., Thomas, D.G.: Parallel contextual array grammars and languages. Electron. Notes Discrete Math. **12**, 106–117 (2003)
3. Ehrenfeucht, A., Paun, G., Rozenberg, G.: Contextual grammars and formal languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Language, vol. 2, pp. 237–293 (1997)
4. Emerald, J.D., Subramanian, K.G., Thomas, D.G.: Inferring subclasses of contextual languages. In: Oliveira, A.L. (ed.) ICGI 2000. LNCS, vol. 1891, pp. 65–74. Springer, Heidelberg (2000). doi:10.1007/978-3-540-45257-7_6

5. Gold, E.M.: Language identification in the limit. Inf. Control **10**, 447–474 (1967)
6. Fernau, H., Freund, R., Holzer, M.: Representations of recursively enumerable array languages by contextual array grammars. Fundamenta Informatica **64**, 159–170 (2005)
7. Fernau, H., Freund, R., Siromoney, R., Subramanian, K.G.: Contextual array grammars with matrix and regular control. In: Câmpeanu, C., Manea, F., Shallit, J. (eds.) DCFS 2016. LNCS, vol. 9777, pp. 98–110. Springer, Cham (2016). doi:10.1007/978-3-319-41114-9_8
8. Fernau, H., Freund, R., Siromoney, R., Subramanian, K.G.: Non-isometric contextual array grammars with regular control and local selectors. In: Durand-Lose, J., Nagy, B. (eds.) MCU 2015. LNCS, vol. 9288, pp. 61–78. Springer, Cham (2015). doi:10.1007/978-3-319-23111-2_5
9. Haussler, D.: Insertion and iterated insertion as operations on formal languages. Ph.D. Thesis, University of Colorado, Boulder (1982)
10. Kari, L.: Contextual insertions/deletions and computability. Inf. Comput. **1**, 47–61 (1996)
11. Krithivasan, K., Balan, M.S., Rama, R.: Array contextual grammars. In: Martin-Vide, C., Paun, G. (eds.) Recent Topics in Mathematical and Computational Linguistics, pp. 154-168 (2000)
12. Lalitha, D., Rangarajan, K., Thomas, D.G.: Petri net generating hexagonal arrays. In: Aggarwal, J.K., Barneva, R.P., Brimkov, V.E., Koroutchev, K.N., Korutcheva, E.R. (eds.) IWCIA 2011. LNCS, vol. 6636, pp. 235–247. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21073-0_22
13. Marcus, S.: Contextual grammars. Revue Roumane de Mathematiques Pures et Appliques **14**(10), 1525–1534 (1969)
14. Rama, R., Smitha, T.A.: Some results on array contextual grammars. Int. J. Pattern Recogn. Artif. Intell. **14**, 537–550 (2000)
15. Rosenfield, A., Siromoney, R.: Picture languages - a survey. Lang. Design **1**, 229–245 (1993)
16. Subramanian, K.G., Van, D.L., Chandra, P.H., Quyen, N.D.: Array grammars with contextual operations. Fundamenta Informaticae **83**, 1–18 (2008)