Electronic Notes in Theoretical Computer Science 46 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume46.html 16 pages

A Three-Dimensional Uniquely Parsable Array Grammar that Generates and Parses Cubes

Katsunobu Imai [a,1], Yukio Matsuda [b], Chuzo Iwamoto [a,2], and Kenichi Morita [a,3]

[a] *Faculty of Engineering*
Hiroshima University
Higashi-Hiroshima, Japan

[b] *Sharp Corporation*
Osaka, Japan

Abstract

A uniquely parsable array grammar (UPAG) introduced by Yamamoto and Morita is a subclass of isometric array grammar (IAG) in which parsing can be performed without backtracking. Hence, we can use a UPAG as an efficient two-dimensional pattern recognition mechanism, if the pattern set is properly described by a UPAG. Furthermore, since a UPAG admits parallel parsing, it can also be used as a formal framework for parallel pattern recognition. In this paper, we show UPAG is also useful to generate and recognize three-dimensional patterns. We construct a UPAG that generate any size of cubes. This UPAG can recognize cubes in linear time of the length of its side by maximum parallel reduction. To construct the grammar, we have made a tool for designing three-dimensional IAG.

1 Introduction

An isometric array grammar (IAG) introduced by Rosenfeld [3] is a formal model of two-dimensional pattern generation. Until now, several subclasses of IAGs have been proposed and investigated. For example, there are a context-sensitive array grammar (CSAG), a context-free array grammar (CFAG), and a regular array grammar (RAG) that form a Chomsky-like hierarchy in IAGs. Although many research has been done in two-dimensional array grammar [6], it is in general very hard to parse two-dimensional languages based on these IAG frameworks. It has been shown that even for RAGs, the lowest subclass

[1] Email: imai@iec.hiroshima-u.ac.jp
[2] Email: iwamoto@iec.hiroshima-u.ac.jp
[3] Email: morita@iec.hiroshima-u.ac.jp

of the Chomsky-like hierarchy, the membership (i.e., recognition) problem is NP-complete [1].

A uniquely parsable array grammar (UPAG) is another subclass of IAGs proposed by Yamamoto and Morita [5]. It is a grammar that satisfies the following condition: for any superposition of right-hand sides of any two rewriting rules, the overlapping portions do not match except "context portions". Because of this condition, parsing can be done without backtracking. UPAGs also admit parallel parsing. That is, we can make any number of reverse applications of rewriting rules simultaneously, and it is assured that such parallel reduction leads exactly the same final result as a sequential reduction [5]. Thus, it can be used as a kind of formal framework for parallel pattern recognition.

On three-dimensional array grammar, Wang [7] introduced a universal 3-d array grammar (UAG) and he showed that some kind of patterns can be generated and parsed effectively by UAGs. But UAG uses a special type of parsing algorithm and it can't be regarded as a standard IAG. So we explore the straightforward extension of normal IAG, i.e., we allow that each rule has three-dimensional shapes. UAG uses rewriting rules of two-dimensional shapes and it is fairly easy to grasp there derivation processes, but if rewriting rules contain 'true' three-dimensional shapes, it becomes quite difficult to design rules and to illustrate there derivation processes.

So we have made a simple tool for designing IAGs. To show the tool is useful, we first design a context-free IAG that generates rectangular parallelepipeds. Next, to show UPAG is also useful to generate and parse three-dimensional patterns, we constructed a UPAG that generates cubes. In particular, it can be possible to recognize any size of cubes in linear time of the length of the side by the grammar.

2 Definition

Let Σ be a nonempty finite set of symbols and $d(= 2 \text{ or } 3)$ be the dimension of an array. A d-dimensional *word* over Σ is a d-dimensional finite connected array of symbols in Σ. The set of all words over Σ is denoted by Σ^{d+} (the empty word is not contained in Σ^{d+}).

Definition 2.1 An *isometric array grammar* (IAG) is a system defined by

$$G = (N, T, P, S, \#),$$

where N is a finite nonempty set of nonterminal symbols, T is a finite nonempty set of terminal symbols ($N \cap T = \emptyset$), $S (\in N)$ is a start symbol, $\# (\notin N \cup T)$ is a special blank symbol, P is a finite set of rewriting rules of the form $\alpha \to \beta$, where α and β are words over $N \cup T \cup \{\#\}$, and satisfy the following conditions:

(i) The shapes of α and β are geometrically identical.
(ii) α contains at least one nonterminal symbol.
(iii) Terminal symbols in α are not rewritten by the rule $\alpha \to \beta$.

(iv) The application of the rule $\alpha \to \beta$ preserves the connectivity of the host array.

If P also satisfies the following conditions, G is said to be *context-free*.

(i) The left-hand side of each rule in P only contains a nonterminal symbol and several # symbols.
(ii) The right-hand side of each rule in P contains no #.

A #-*embedded* array of a word $\xi \in (N \cup T)^{d+}$ is an infinite array over $N \cup T \cup \{\#\}$ obtained by embedding ξ in a d-dimensional infinite array of #s, and is denoted by $\xi_{\#}$. (Formally, a #-embedded array is a mapping $\mathbf{Z}^d \to (N \cup T \cup \{\#\})$, where $\mathbf{Z}$ is the set of all integers.) We say that a word η is *directly derived* from a word ξ in G if $\eta_{\#}$ can be obtained by replacing one of the occurrences of α in $\xi_{\#}$ with β for some rewriting rule $\alpha \to \beta$ in G. This is denoted by $\xi \underset{G}{\Rightarrow} \eta$. The reflexive and transitive closure of the relation $\underset{G}{\Rightarrow}$ is denoted by $\overset{*}{\underset{G}{\Rightarrow}}$. We say that a word η is *derived* from a word ξ in G if $\xi \overset{*}{\underset{G}{\Rightarrow}} \eta$. We write $\xi \overset{n}{\underset{G}{\Rightarrow}} \eta$ $(n = 0, 1, 2, \cdots)$ if there is a sequence of words $\zeta_0, \zeta_1, \ldots, \zeta_n \in (N \cup T)^{d+}$ that satisfy

$$\xi = \zeta_0 \underset{G}{\Rightarrow} \zeta_1 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} \zeta_n = \eta.$$

The *array language* generated by G (denoted by $L(G)$) is defined by

$$L(G) = \{\, w \mid S \overset{*}{\Rightarrow} w, \text{ and } w \in T^{d+} \,\}.$$

A rewriting rule $\alpha \to \beta$ is said to be *applicable* to ξ *at* $u \in \mathbf{Z}^d$, iff α occurs in $\xi_{\#}$ at the position u, where the *position* of an occurrence means the x–y coordinates of the leftmost symbol of its uppermost row of α $(d = 2)$ or x–y–z coordinates of the leftmost symbol of uppermost row of its bottommost pattern of α $(d = 3)$. If $\eta_{\#}$ is obtained by applying $\alpha \to \beta$ at u, we say η is directly derived from ξ by the rewriting with the *label* $L = [\alpha \to \beta, u]$. This is denoted by $\xi \overset{L}{\Rightarrow} \eta$. The label L itself is also said to be applicable to ξ.

Similarly, a rewriting rule $\alpha \to \beta$ is said to be *reversely applicable* to η *at* u, iff β occurs in $\eta_{\#}$ at the position u.

Let $\alpha \to \beta$ be a rule. The subarray of α whose symbols are not changed (i.e., rewritten to the same symbols) by the application of $\alpha \to \beta$ is called the *context portion* of α. The subarray of α all of whose symbols are rewritten to different symbols is called the *rewritten portion* of α. The context portion and the rewritten portion of β are also defined similarly.

Definition 2.2 Let $G = (N, T, P, S, \#)$ be an IAG. If P satisfies the following condition, G is called a *uniquely parsable array grammar* (UPAG).

The UPAG Condition:

(i) The right-hand side of each rule in P contains a symbol other than # and S.

(ii) Let $r_1 = \alpha_1 \to \beta_1$ and $r_2 = \alpha_2 \to \beta_2$ be any two rules in P (may be $r_1 = r_2$). Superpose β_1 and β_2 at all the possible positions variously translating them. For any superposition of β_1 and β_2, if all the symbols in overlapping portions of them match, then
   (a) these overlapping portions are contained in the context portions of β_1 and β_2, or
   (b) the whole β_1 and β_2 are overlapping, and $r_1 = r_2$.

Example 2.3 The pair of rewriting rules

$$aB \to ab, \quad Ca \to ca$$

satisfies the UPAG Condition, while the following does not.

$$\#B \to ab, \quad Ca \to ca$$

In [5] the following Lemmas have been shown.

Lemma 2.4 [Unique Parsability] [5] Let $G = (N, T, P, S, \#)$ be a UPAG. Let $\alpha \to \beta$ be any rewriting rule in P which is reversely applicable to $\eta \in (N \cup T)^{d+}$ at u. If

$$\eta \overset{n}{\Leftarrow} S,$$

then the following relation holds for some ζ:

$$\eta \overset{L}{\Leftarrow} \zeta \overset{n-1}{\Leftarrow} S.$$

Lemma 2.5 [Parallel Parsability] [5] Let $G = (N, T, P, S, \#)$ be a UPAG. Let $L_1, \cdots, L_m$ be different labels which are reversely applicable to $\eta \in (N \cup T)^{d+}$. If

$$\eta \overset{n}{\Leftarrow} S,$$

then the following relation holds for some ζ:

$$\eta \overset{\{L_1,\cdots,L_m\}}{\Longleftarrow} \zeta \overset{n-m}{\Leftarrow} S.$$

From the Lemma 2.5, the next corollary on maximum parallel reduction can be obtained.

Corollary 2.6 [Maximum Parallel Parsability] Let $G = (N, T, P, S, \#)$ be a UPAG. If

$$\eta \overset{n}{\Leftarrow} S,$$

then the following relation holds for some ζ:

$$\eta \Lleftarrow \zeta \overset{n-m}{\Leftarrow} S,$$

where m is the total number of labels reversely applicable to η.

All above conditions are agree with three-dimensional cases and each rewriting rule has a pair of three-dimensional figures.

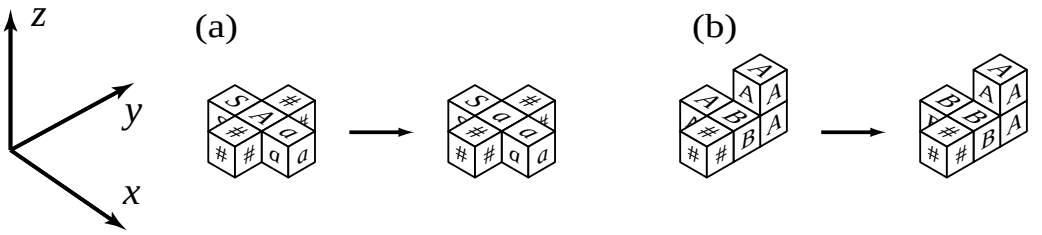


Fig. 1. Examples of three-dimensional IAG rules.

Example 2.7 Examples of rewriting rules of three-dimensional IAG are shown in fig. 1.

Rules (a) and (b) in fig. 1 are denoted as follows respectively.

$$(a)\ \begin{matrix} \_\#\_ \\ SAa \\ \_\#\_ \end{matrix} \rightarrow \begin{matrix} \_\#\_ \\ Sa\,a \\ \_\#\_ \end{matrix}$$

$$(b)\ \begin{matrix} \_A\ \ \_A \\ AB\ \ \_\_ \\ \_\#,\_\_ \end{matrix} \rightarrow \begin{matrix} \_A\ \ \_A \\ BB\ \ \_\_ \\ \_\#,\_\_ \end{matrix}$$

The rule (a) is the same as the two-dimensional case, while (b) is "true" three-dimensional one. Patterns stacked along z-axis is delimited by commas and '_' denotes that the position is not used as the context of the rule.

3 A tool for designing three-dimensional isometric array grammars

On designing three-dimensional array grammars, each rewriting rule has a complex structure and it is quite difficult to cope with such rules, in particular, checking the UPAG condition. Furthermore, simulating derivation processes and application of each rules are very difficult to perform without any software tools.

So we made a simple tool for designing IAG. Our tool is made with Macintosh Common Lisp and a three-dimensional graphic library. It has windows for manipulated patterns (fig. 2) and for rewriting rules (fig. 3), which make it easy to edit and apply rules interactively.

It also has a tool which checks whether a grammar is UPAG or not. So it is capable of designing three-dimensional IAGs with ease.

4 A context-free isometric array grammar that generates rectangular parallelepipeds

In this section, as an example of three-dimensional IAG, we show a context-free IAG that generates rectangular parallelepipeds.

Yamamto and Morita [4] designed a context-free IAG G_R that generates rectangles. We extend it and construct a three dimensional context-free IAG

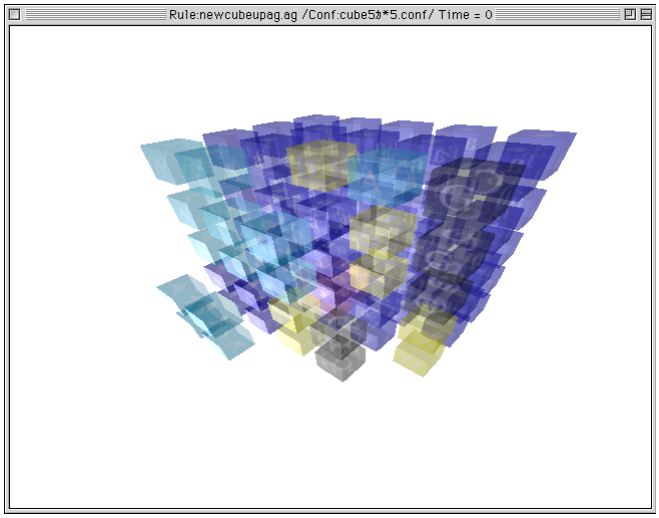


Fig. 2. A pattern window

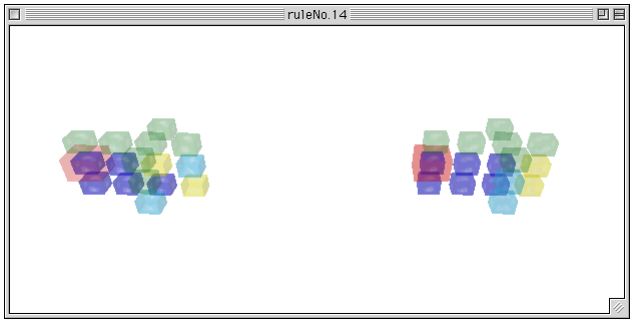


Fig. 3. A rule window

G_{RP} that generates rectangular parallelepipeds of symbol 'a's.

$G_{RP} = (V_{RP}, T_{RP}, P_{RP}, S, \#)$
$V_{RP} = \{S, U, R, L, I, J_1, J_2, J_3, J_4, J_5, J_6, J_7, J_8, J_9\}$
$T_{RP} = \{a\}$
P_{RP}, i.e., rewriting rules of G_{RP} are listed in Appendix A.

Fig. 4 shows the outline of a derivation process of G_R. In the process, the rule of fig. 5(1) is used to advance the length of rectangle and the rule of fig. 5(2) is used to terminate the generation.

G_{RP} uses the same approach to generating rectangular parallelepipeds. Fig. 6 shows the outline of a derivation process. First, a rectangle is generated by the same way as G_R. But this time, nine non-terminals $J_1, \cdots, J_9$ are embedded. These symbols denote the shape of each block and referring these symbols, it is possible to stack blocks of the same shape on the rectangle. After repeating this process of stacking blocks, it is interrupted by the same approach as G_R and the height is fixed.

G_{RP} generates any rectangular parallelepipeds of depth $3k+6$, width $2l+3$, and height $2m + 1 (k, l, m = 0, 1, 2 \ldots)$. Fig. 7 shows a derivation example of

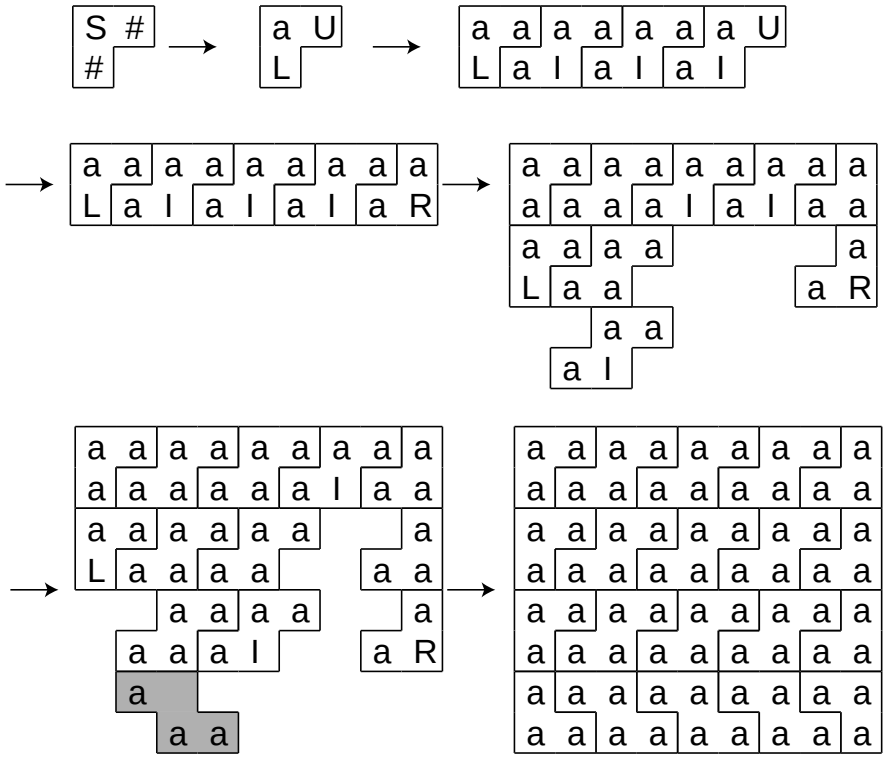


Fig. 4. A derivation example of G_R

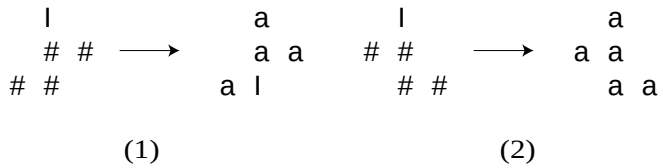


Fig. 5. The 'advancing' rule and the 'terminating' rule in G_R.

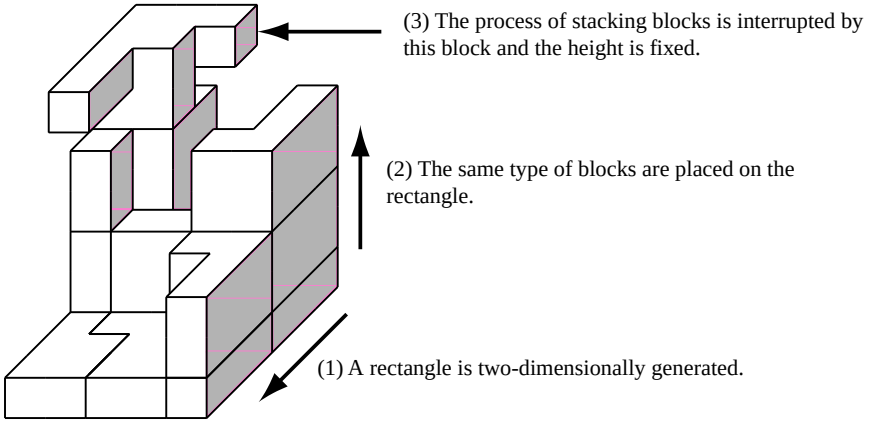


Fig. 6. The outline of derivation of a rectangular parallelepiped by G_{RP}

G_{RP}. To add appropriate non-terminal symbols, G_{RP} can be extended to generate any size of rectangular parallelepipeds.

5 A uniquely parsable array grammar that generates any size of cubes

In previous section, we show context-free IAG G_{RP} that generates rectangular parallelepipeds. But it is impossible to use G_{RP} to recognize that a pattern is a rectangular parallelepiped or not. So in this section, we show an example of three-dimensional UPAG.

By using the simulating tool described above, we constructed a UPAG G_{cube} that generates any cubes of symbol 'a's.

$G_{cube} = (N_{cube}, T_{cube}, P_{cube}, S, \#)$,
$N_{cube} = \{S, X, Y, Z, F, A, D, G, E\}$,
$T_{cube} = \{a\}$,
P_{cube}, i.e., rewriting rules of G_{cube} are listed in Appendix B.

G_{cube} generates and recognizes any cubes of the size $l(\geq 4)$. It is easy to extend G_{cube} to generates any size of cubes, i.e., including $l(< 4)$, by adding several rewriting rules.

G_{cube} generates a cube by the following method.

(i) generates an $l \times l$ square which consists of non-terminal symbol 'F' in x–z plane (fig. 8(1)).

(ii) generates rectangles of the size $l \times (l-1)$ l times (fig. 8(2,3,4)).

The outline of a generating process of G_{cube} is as follows.

Rewriting rules from (31) to (39) in Appendix B. are used to generate an 'F'-filled square and the other rules are used to generate $l \times (l-1)$ rectangles on x–y plane.

(i) Generating an 'F'-filled square:
The rule (32) is applied to advance the side of an 'F'-filled square in $-x$ direction. (31) is used to terminate its growth and to generate a symbol 'Z'. (33) is used to advance the side of the square in $-z$ direction (34) is used to place 'F' in $-x$ direction and each row is completed by (36). 'Z' is sent along the diagonal direction of the square by (35), and if 'Z' reaches to the other side, (37) is used to fix the height of the square and generate 'Y' at the bottom side of the square. (38) sends 'Y' along the bottomside in $-x$ direction and (39) completes the square.

(ii) Generating $l \times (l-1)$ rectangles:
Rules (11)–(20) and (21)–(30) are used for generating top and bottom rectangles respectively and (1)–(10) are used for generating the other rectangles.
$(1+10k)$ $(k = 0, 1, 2)$ advances the side of rectangles in $+z$ direction. This rule invokes each derivation processes of rectangles. $(2+10k)$,$(4+10k)$,$(5+10k)$ are used to advance rectangles in $-y$ and $+x$ direction and $(3+10k)$ terminates the growth in $+x$ direction.

$(6 + 10k)$ and $(7 + 10k)$ are used to 'count' the length of the side of each rectangle and the symbol 'G' is used as a marker and play almost the same roll as 'Z' in generating a 'F'-filled square. To keep the size of each rectangle should be $l \times (l - 1)$, $(8 + 10k)$ is used to terminate the growth in $-y$ direction and the symbol 'E' denotes that the growth is finished.

$(10 + 10k)$ sends 'E' in $+x$ direction and the generation process is completed by $(9 + 10k)$.

Wang [7] also show a UAG that generates and parses cubes. Although his grammar uses a special type of parsing algorithm, G_{cube} uses completely the same framework as two-dimensional IAG. Furthermore, it satisfies the UPAG condition and parallel parsing can be performed without backtracking.

By maximum parallel reduction, parsing l rectangles takes $7(l - 4) + 20$ steps and parsing an 'F'-square takes $3(l-4)+10$ steps. Thus the total parsing steps of a cube of the size l is $10(l - 4) + 30$ and linear to l. Fig. 9 shows a maximum parallel reduction example of G_{cube}.

6 Conclusion

In this paper, we apply IAG to three-dimensional pattern generations and recognitions. We constructed a context-free IAG G_{RP} that generates rectangular parallelepipeds and a UPAG G_{cube} that generates cubes. In particular, it is possible to recognize cubes in linear time of the length of their side by G_{cube}.

In the two-dimensional case, there are UPAGs that generate and recognize connected patterns [2]. Although such topological properties in the three-dimensional case are very difficult to describe with three-dimensional IAGs and UPAGs, they are interesting problems.

The derivation and parsing processes are hard to show on a paper. These examples can be seen as image files and QuickTime movies at the following addresses via WWW:

`http://www.iec.hiroshima-u.ac.jp/projects/ag/3d/.`

Acknowledgements: The authors thank Hiroyuki Aga (SONY Co.) for his comments. All software for simulations are made with Macintosh Common Lisp and its graphic libraries. These systems are maintained by engineers of Digitool, Inc. and John Wiseman (Neodesic, Inc.).

This work was supported in part by Grant-in-Aid for Scientific Research (C) No. 12680353 from JSPS, and by Electric Technology Research Foundation of Chugoku.

References

[1] Morita, K., Y. Yamamoto, and K. Sugata, "The complexity of some decision problems about two-dimensional array grammars," *Information Sciences*, **30** (1983), 241–262.

[2] Morita, K., K. Imai, Uniquely parasable array grammars for generating and parsing connected patterns, *Pattern Recognition Journal* **32** (1999), 269–276.

[3] Rosenfeld, A., *Picture Languages*, Academic Press, New York (1979).

[4] Yamamoto, Y., K. Morita, and K. Sugata,: An Isometric Context-Free Array Grammar That Generates Rectangles, *Trans. IECE of Japan*, **E 65**, No.12 (1982), 754–755.

[5] Yamamoto, Y., and K. Morita, Two-dimensional uniquely parsable isometric array grammars, *Int. J. Pattern Recognition and Artificial Intelligence*, **6** (1992), 301–313.

[6] Wang, P.S.P., Array grammars, patterns and recognizers, *Parallel Image Analysis and Processing*, P.S.P Wang (Ed.), Series in Computer Science **18**, World Scientific (1989).

[7] Wang, P.S.P., Three-dimensional sequential/parallel universal array grammars for polyhedral object pattern analysis, *Parallel Image Analysis and Processing*, K.Inoue et. al. (Ed.), Series in Machine Perception Artificial Intelligence **15**, World Scientific (1994), 563–576.

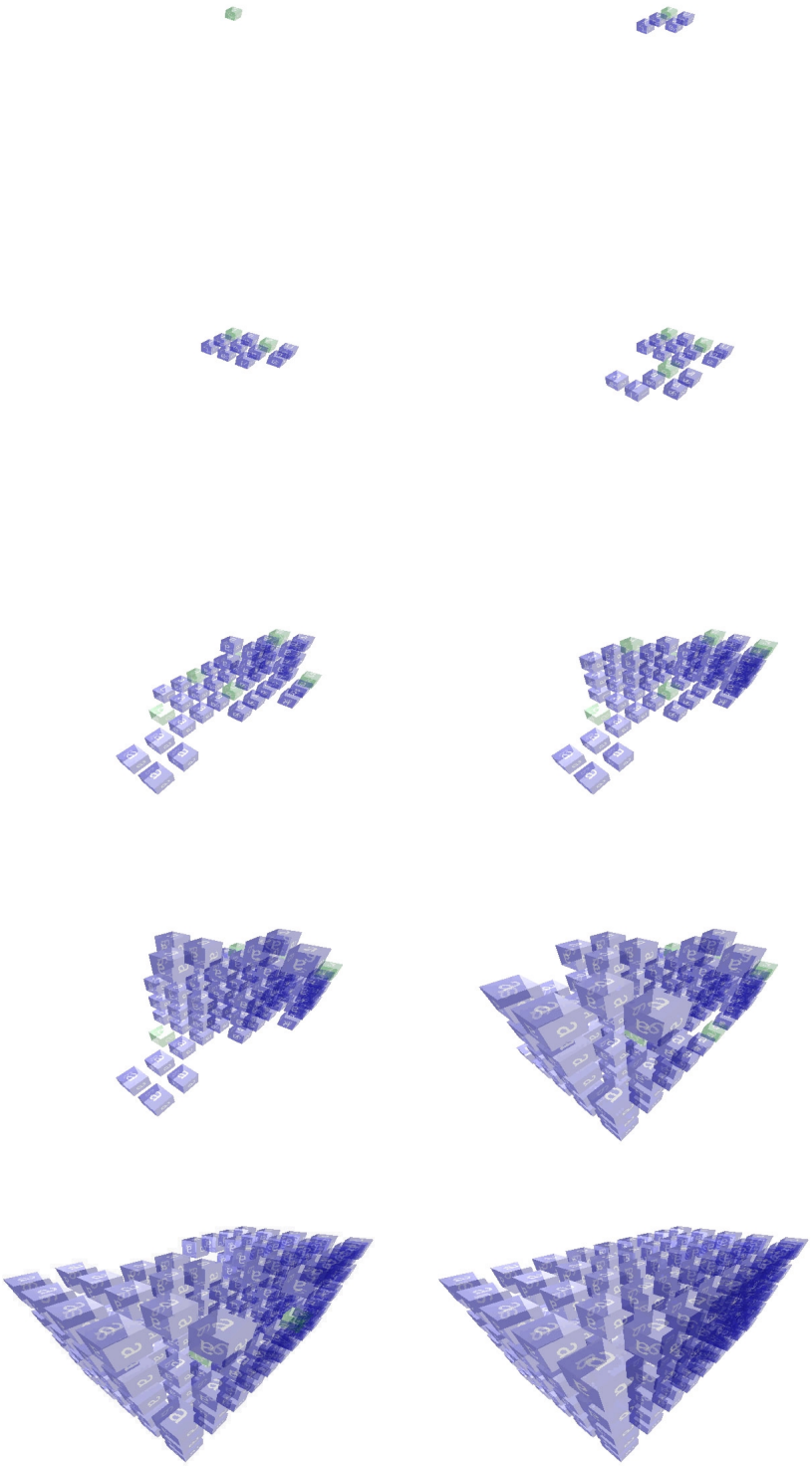


Fig. 7. A derivation example of G_{RP}

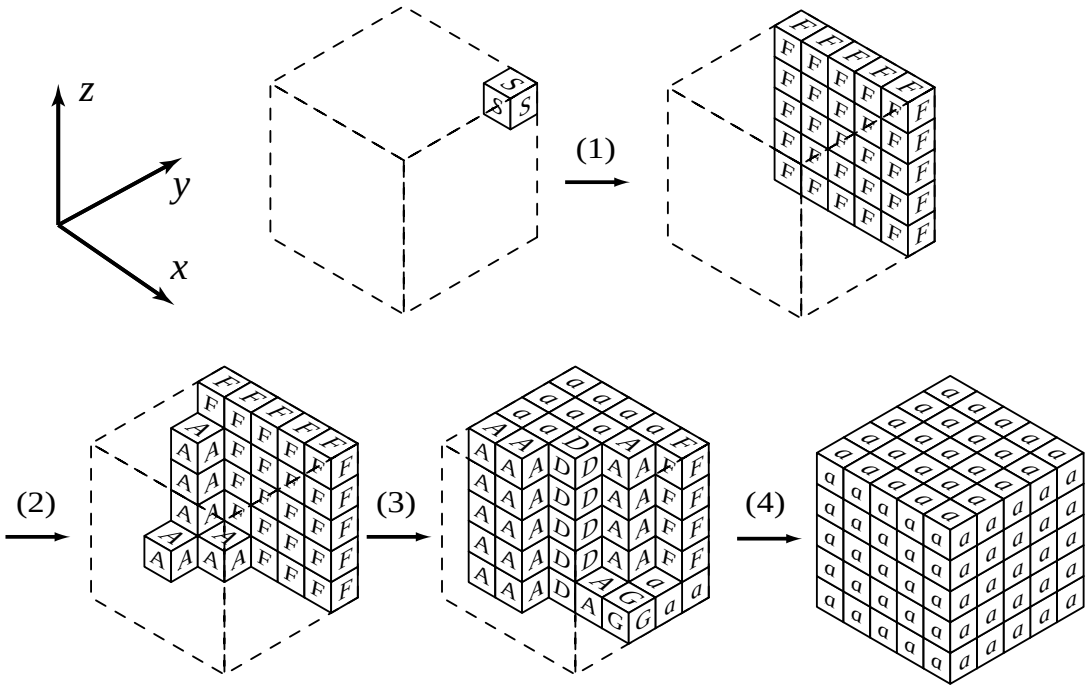


Fig. 8. The outline of derivation of a cube by G_{cube}

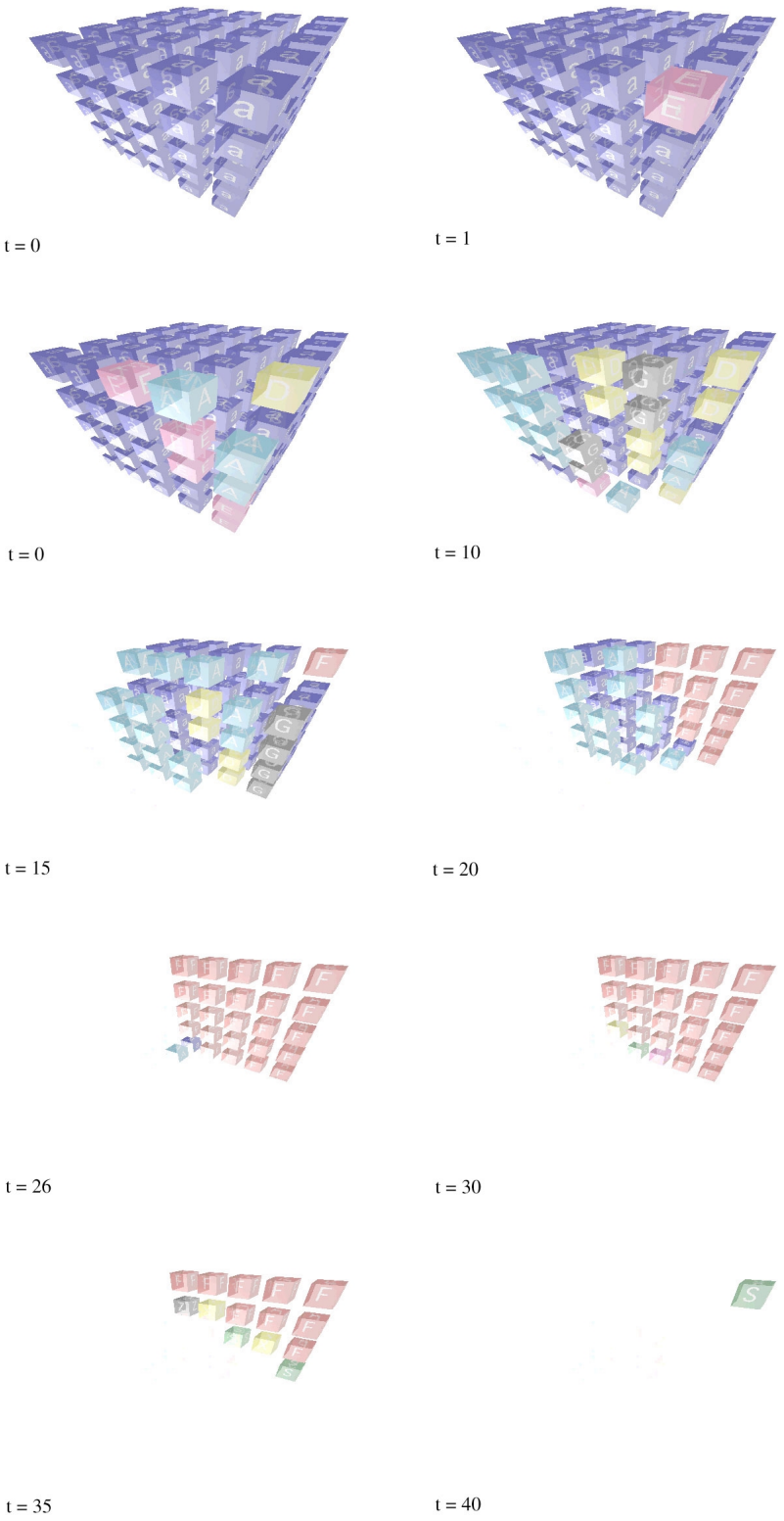
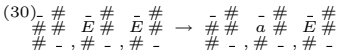
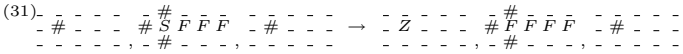


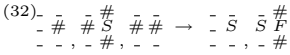
Fig. 9. A maximum parallel reduction example of G_{cube}

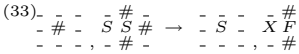
A Rewriting rules of G_{RP}

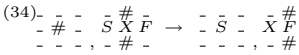
B Rewriting rules of G_{cube}

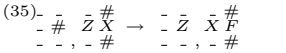
(30) 

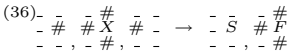
(31) 

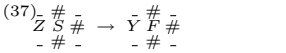
(32) 

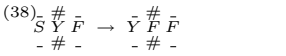
(33) 

(34) 

(35) 

(36) 

(37) 

(38) 

(39)