Grammars of Interlocking SL Blocks

Shen-Guan Shih[1,*]

[1] Department of Architecture, National Taiwan University of Science and Technology, 106 Taipei, Taiwan

[*] Corresponding author e-mail: sgshih@mail.ntust.edu.tw

Abstract

Grammar is a kind of abstract representations for defining how the composite whole can be derived from a hierarchy of mutually related parts. This paper discusses how grammars can be used as a means to assist the design and construction of large and complex compositions out of a simple building block.

An SL block is an octocube designed for making semi-interlocking structures extensible in three orthogonal directions. String re-write grammars are used to define languages of SL block compositions. It is expected to establish a mathematical basis for SL block compositions. Investigations and speculations through concepts, principles and notations of such a grammatical approach are proposed. A building frame form generator is devised to provide views towards how SL blocks can be systematically arranged to create architectural forms. With the grammatical approach, it might be possible to implement compilers for high level composition languages of SL block compositions.

Keywords: polycube, building block, grammar, interlocking.

1 Introduction and background

Combining great numbers of units to form large and complicated structures is a general principle in the development of natural and man-made objects. Grammar is a formalism to define syntactic structures of languages. This study proposes a strategy of using grammars to enable the compilation of high level representations to low level building block compositions.

Engineers, architects, product designers and puzzle designers look for interlocking configurations that can connect separated parts into strong and stable structures (Yong 2011; Weizmann et al. 2017; Fu et al. 2015; Song et al. 2012). Topological interlocking (Dyskin et al. 2003) opens up opportunities to the discovery of new and enchanted materials and structures by cellular units (Kanel-Belov et al. 2010; Estrin et al. 2011). This study proposes a system that may construct large varieties of forms with various structure behaviours by interlocking blocks of identical shapes. For architecture, it may contribute to the design of reusable building blocks that can be efficiently manufactured by means of mass production.

1.1 Semi-interlocking

Interlocking is an interesting issue for prefabricated constructions. Advances of digital fabrication technology drive researches towards automatic generation of interlocking parts for assembly (Song et al. 2012). Interlocking property of assembled structures can be classified by calculating the degrees of translational freedom for parts that are not to be dissected, and the network of relations for parts engagements (Fu et al. 2015). Among these researches, polycubes were often used as the basic elements for simplification and generality (Lo et al. 2009; Song et al. 2012). In this study, semi-interlocking is defined as the property of structures made up with units that are either locked topologically, or less preferably, with individual or groups of units held to the structure by friction and left with one single direction of translational freedom. A semi-interlocking structure may retain stability under forces from various directions, and allows at least one feasible sequence for assembling and disassembling.

1.2 The *SL* block, Conjugate pair, concatenation and *SL* strand

An *SL* block (Shih 2016, 2018) is a kind of octocube that can be used to build large variations of semi-interlocking structures without using any connectors or adhesives. **Figure 1a** shows the figure of an *SL* block, with three arrows shows the referred X, Y, Z axial directions and the rotational center at the intersection of arrows. Two *SL* blocks arranged into 180° Y axis rotational symmetry are called a conjugate pair, as shown in **fig. 1b**. Conjugate pairs of *SL* blocks can be sequentially concatenated to build a linked structure called an *SL* strand. Six types of concatenations are denoted as h, t, s, d, a and y. In fig. 2, the concatenating pair is shown as blue and the host pair that is to be concatenated is shown as transparent. Each type of concatenations would forward the free end of the strand into a specific direction and position. The *SL* strands may form closed loops if both ends meet at the same location like a snake that bites its own tail. All closed *SL* strands are semi-interlocking without using any adhesives between *SL* blocks.

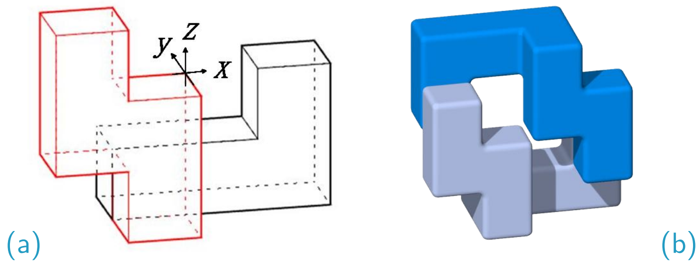

Figure 1: (a) an *SL* block, (b) a conjugate pair.

In fig. 2, axial rotations and translations of the corresponding transformations of each types of concatenations are denoted as $R_{x(angle)}$, $R_{y(angle)}$, $R_{z(angle)}$ and $T_{(xyz)}$. The six types of concatenations can be divided into two groups of three by whether the corresponding transformation consists of 180° rotation along x axis or not. Concatenations in both groups can be further characterized by 0°, -90° and 90° of rotations on z axis within the transformation. Translations for concatenations are strictly dependent upon rotations.

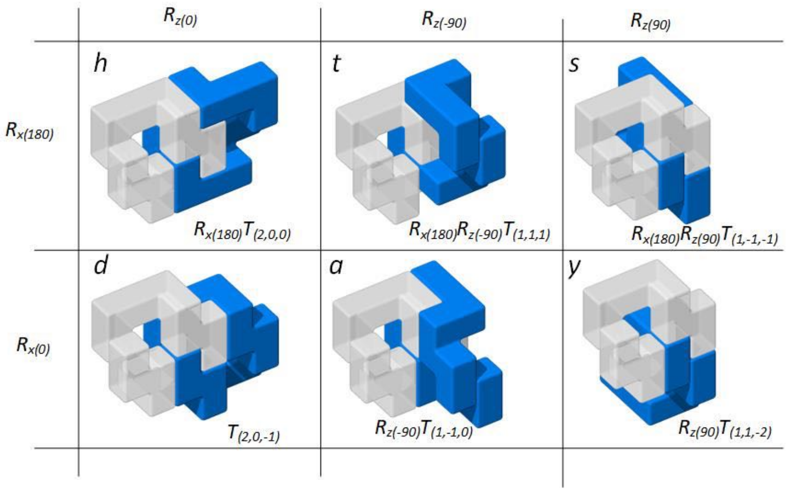

Figure 2: Six types of concatenations with their corresponding transformations.

2 Methods

2.1 String representation of *SL* strands

The *SL* strands can be represented as strings consisting of six letters that stand for the six types of concatenations. Concatenations are regarded as non-commutative multiplications. **Figure 3a** shows an open strand $hhhh$, or h^4, with the exponent stands for repetitive patterns. An open strand consists of one more pair of *SL* blocks than the number of concatenations. **Figure 3b** shows a closed *SL* strand of a^4. Consisting with just 4 conjugate pairs, it is the smallest closed *SL* strand. **Figure 3c** shows the strand of $(hha)^4$, which is formed by repeating concatenations (hha) 4 times. **Figure 3d** shows the strand of $hhahdhshthy$. In the figure, colors are used to distinguish types of concatenations between a pair and its preceding pair. The white blocks on the left are the initial pair.

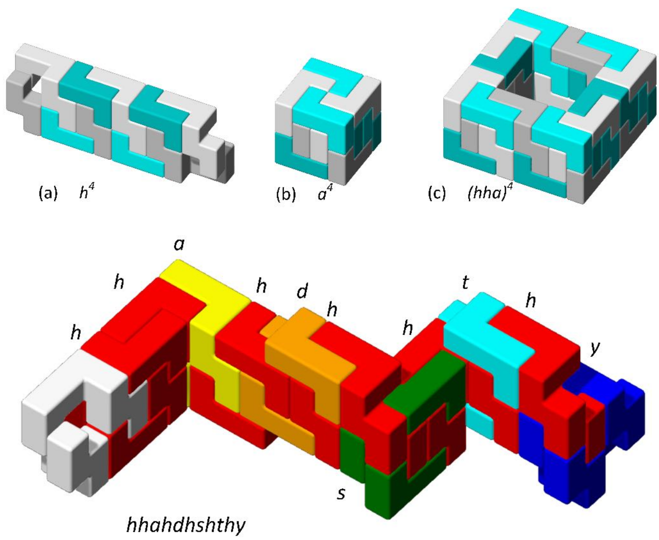

Figure 3: (a)an open strand, (b) and (c) two closed strands, (d) an open strand with colors showing the types of concatenations between a pair of *SL* blocks and their preceding pair.

2.2 *SL* strand grammars

A grammar consists of an initial variable and a set of rewrite rules, which are used to transform the initial variable into strings consisting of only terminals h, a, d, s, t and y, denoting the types of concatenations. For the definition of grammars, the symbol 1 is used to represent the multiplicative unity, or an empty concatenation. Denoted as bold capital letters, a variable represents an uncertain *SL* strand that has not been derived through rewrite rules derivations. A grammar defines the domain of its initial variable. The derivation of a variable based on rewrite rules in the grammar would uncover uncertainty and downsize the domain of the variable.

When the derivative process is fully uncovered, the initial variable can be evaluated to a specific *SL* strand. In this paper we use polynomials notation for grammars. Concatenation is regarded as noncommutative multiplication. Addition is regarded as “or” and adds derivative options to the variable on the left hand side of the rule. Rewrite rule options are added to make a summation to define the domain of rule. Rewrite rule options are added to make a summation to define the domain of a variable. Equal sign is used to associate a variable with its rewrite options on the right hand side. The multiplication is associative but not commutative. The addition is associative and commutative. The multiplication is distributive over addition. For example the following rewrite rule defines the domain of the variable X as $\{aa, aha, ahha\}$, and X can be assigned with any member of the set when it is evaluated.

$$X = a(1 + h + h^2)a = aa + aha + ahha$$

Rewrite rules can be recursive. The following rule defines a domain for the variable X as $(h + h^2 + h^3 + ... + h^n)$.

$$X = h(1 + X)$$

The grammar G_1 is a universal grammar that defines all possible *SL* strands without checking self-collision. In G_1, items separated by addition in the expression on the right hand side of the rule are options that can be selected for rewriting. The process may go on recursively until X is rewritten with the multiplicative unit 1 and the recursion terminates. **Figure 4** shows 8 strands from G_1, generated by random selections of rewrite options.

$$G_1 : X = (h + a + d + s + t + y)(X + 1)$$

The grammar G_2 defines closed strands with the shape of cube or elongated cubes of variable lengths. The language consists of palindromes with aa separating zero or even numbers of consecutive $h's$ on both sides and end both ends with an a. The simplest form in the language is a cube, denoted as the string $aaaa$ or a^4. Three strands generated by G_2 are shown in **fig. 5**. Conjugate pairs that are appended with a and h concatenations are shown in yellow and red respectively.

$$G_2 : S$$

$$1. S = aXa$$

$$2. X = hhXhh + aa$$

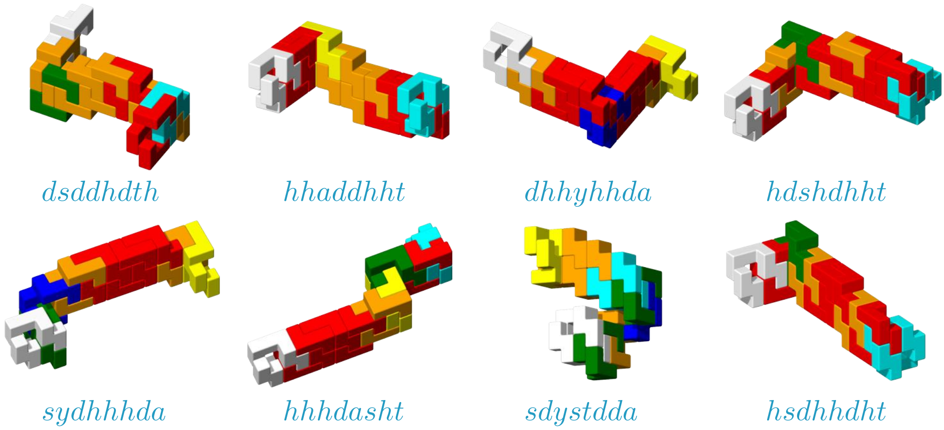

Figure 4: 8 strands from G1, with colors showing the type of concatenations between a pair of *SL* blocks and their preceding pair. White blocks are the initial pair. h: red, a: yellow, d: orange, s:green, t:cyan, y: blue.

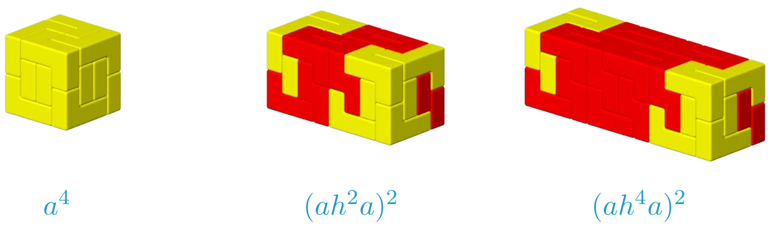

Figure 5: Three strands of G_2 with various lengths, with colors showing the type of concatenations between a pair and its preceding pair.

2.3 Fixed ending grammar

Every concatenation of *SL* block has a corresponding matrix that defines the transformational relationship between the concatenated and the concatenating *SL* pairs. The transformational relationship between the first and the last *SL* pairs of a strand can be derived by multiplying all concatenating matrices. Grammars that always generate strands with both ends at a consistent transformation are called fixed ending grammars. The transformation from the starting *SL* pair to the ending pair is called the end transformation of the strand. The grammar G_2 always generates closed strands. The grammar G_2 is a fix-ending grammar with identity matrix as its end transformation. Fixed ending grammars are of particular interest in this study. Designers may cut off a segment of a strand and insert the initial variable of a fixed ending grammar with an end transformation that matches the cut.

The grammar G_3 is a fixed ending grammar with end transformation equivalent to concatenations hh. **Figure 6** shows 3 strands generated by G_3, with starting pair in cyan and ending pair in blue. On the left hand side of the figure there shows the matrix for the end transformation of all strands generated by G_3.

$$
\begin{aligned}
&G_3 : X\\
&1.X = aYa\\
&2.Y = hhYhh + haah
\end{aligned}
$$

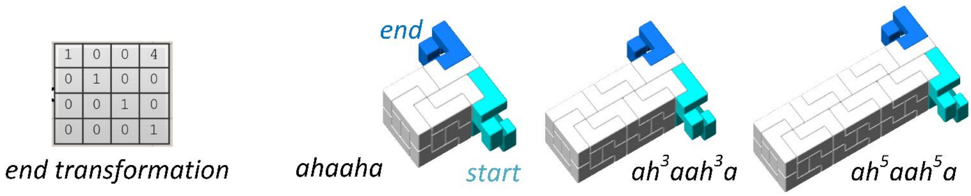

Figure 6: Fixed ending strands generated by G_3.

2.4 Inserting grammars to strand

Figure 7 shows the process of inserting the strand $ahaaha$ (Figure 7a) into the strand $(ah^6a)^2$ (**fig. 7b**) with the 5th and 6th pairs removed. The result of the insertion is $ah^3ahaahah^6a$, as shown in **fig. 7c**. The removed pairs are shown as red transparent blocks in **fig. 7b**.

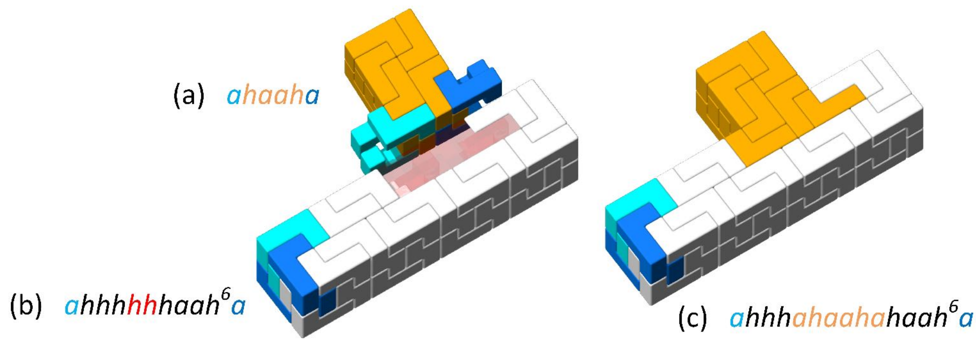

Figure 7: (a) the inserting strand, (b) the inserted strand, (c) the resulting strand.

The inserting strand can be replaced by a fixed ending grammar with the same ending transformation. The purpose is to incorporate uncertainty into *SL* strands construction when design decisions are incomplete. For grammar G_4, when the variable X is recursively substituted with the right hand side options of the rule, the domain of X is expanded endlessly as the following summation:

$$X = ah^{2n+1}a^2h^{2n+1}a + ah^{2(n-1)+1}a^2h^{2(n-1)+1}a + \ldots + ah^3a^2h^3a + aha^2ha$$

Since the addition is interpreted as "or", the value of X remains uncertain and variable within the domain defined by the infinite summation. Inserting the variable X into the strand $(ah^6a)^2$ with the contacting segment removed, the resulting strand becomes $ah^3aXaha^2h^6a$. When the uncertainty is uncovered and the value of X observed, the strand can be evaluated into one of the items in the following summation:

$$ah^3ah^{2n+1}a^2h^{2n+1}aha^2h^6a + ah^3ah^{2(n-1)+1}a^2h^{2(n-1)+1}aha^2h^6a + \cdots$$
$$+ah^3ah^3a^2h^3aha^2h^6a + ah^3aha^2haha^2h^6a$$

Figure 8 shows some possible outcomes of the strand after X is uncovered.

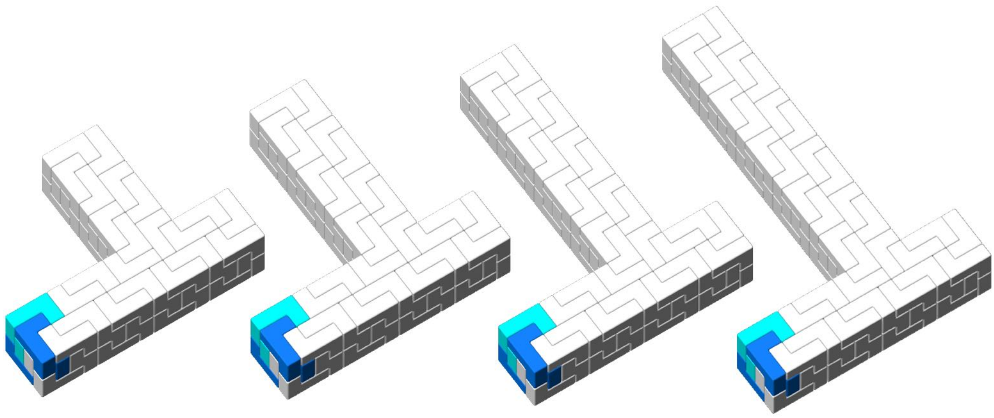

Figure 8: Some possible strands created by insertions.

2.5 Inserting a grammar to another grammar

The grammar G_4 is a fixed ending grammar with end transformation of $haah$. Some strands created by G_4 are shown on top of **fig. 9a**. The grammar G_5 always generates closed strands that are folded into elongated rectangles of various lengths. The definition of the variable S in G_5 consists of two rewrite options, of which $haah$ is the only option that would terminate the recursive rewriting for the derivation of S. It is apparent that the variable S must be uncovered to a strand that has a sub-strand as $haah$. Since $haah$ is the fix-ending transformation of G_4, it is possible to append the rewrite options of the variable S in G_5 with the rewrite options of the initial variable X of G_4. The grammar G_6 is defined by such a grammar insertion plus an additional rewrite option hSh for the variable Y, which enables mutual recursive derivations between the variables Y and S. **Figure 9b** shows 5 strands generated by G_6.

$G_4 : X$

$1.X = ahYhhah + hahhYha$

$2.Y = hhYhh + aa$

$G_5 : P$

$1.P = ahSha + aaaa$

$2.S = hhShh + haah$

$G_6 : P$

$1.P = ahSha + aaaa$

$2.S = hhShh + haah + ahYhhah + hahhYha$

$3.Y = hhYhh + aa + hSh$

Variable S' can be expanded to the following summation:

$$S' = h^{2n+1}aah^{2n+1} + h^{2(n-1)}aah^{(2n-1)} + ... + h^3aah^3 + haah$$

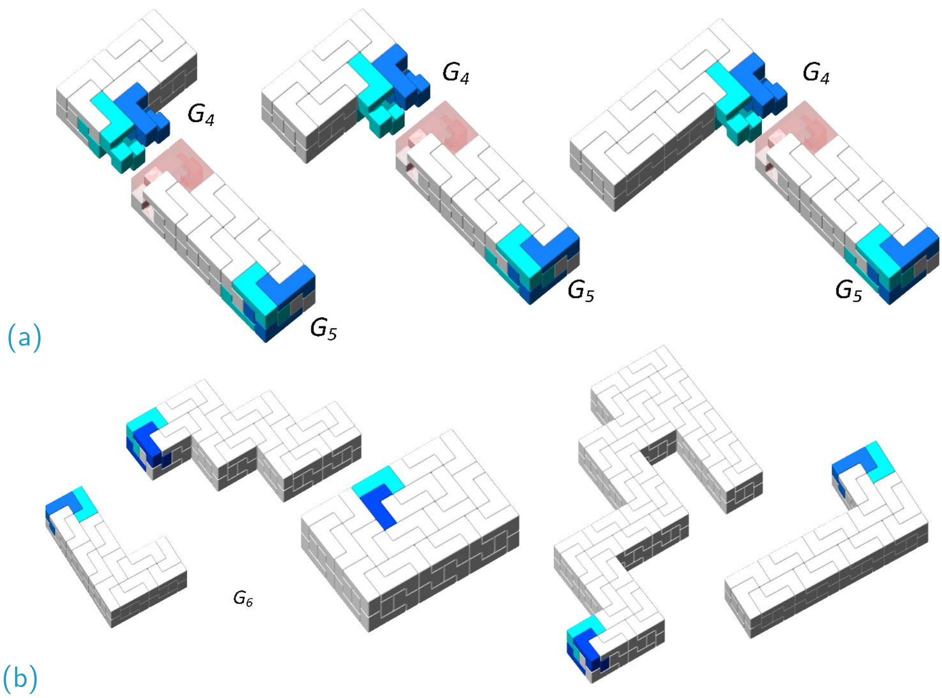

Figure 9: An example of grammar insertion: (a) strands created by inserting strands of G_4 to a strand of G_5; (b) Strands created by G_6.

$G_7 : T$

$1.T = aLa$

$2.L = hSh + aa$

$3.S = hLh + aShah + hahSa + aSaShh + aShhSa + hhSaSa + aSaSaSa$

In the grammar G_7, the variable S appears in rule 2 and rule 3. The second rewrite option aa for the variable L is the only one rewrite option in both rules that does not include any variables. The only way for S to be rewritten with a strand consists of the variable L is the first rewrite option hLh. It can be inferred that the variable L would be preceded and succeeded with the concatenation h when it is finally rewritten to aa. Therefore the sub-strand $haah$ would always appear at the time when the recursive derivation of the variable S terminates. Based on the definition of G_7, assuming a variable S' which inherit only the first rewrite option from rule 3 and the variable L, we may derive the following rules for S':

$$L' = hS'h + aa$$
$$S' = hL'h$$

SL strands cannot have branches sprout away from the main path, but can be folded to make branch-like forms, called pseudo-branching. The grammar G_7 generates pseudo-branching structures.

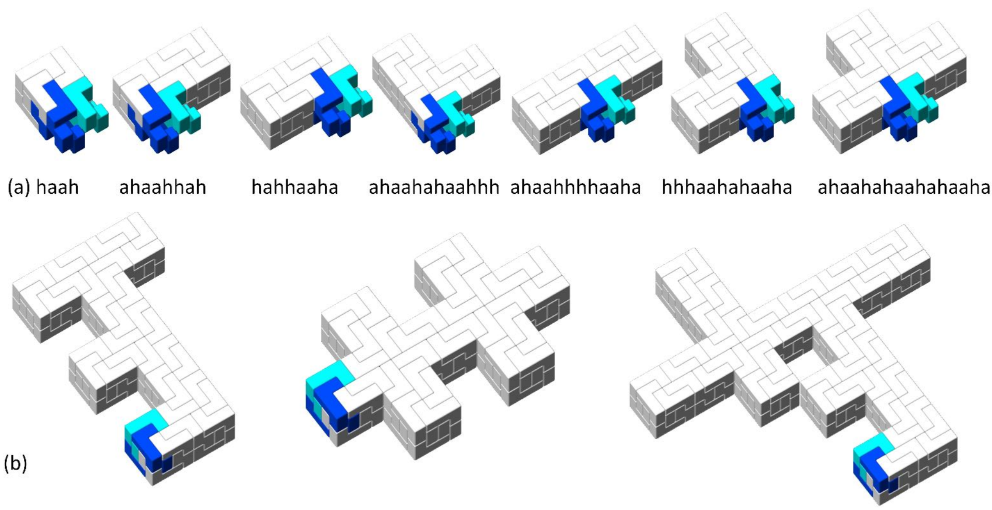

Figure 10: a. Shortest strands for rewrite options of S in G_7, b. Three strands derived with G_7.

The variable S' can be proven to be fixed ended with transformation $haah$. Let's add one more rewrite option $aS'hah$ to S'. With S' rewritten with its own end transformation $haah$, the strand $aS'hah$ becomes $ahaahhah$, which also has $haah$ as the end transformation. Therefore, the end transformation of S' remains unchanged after the new rewrite option is added. We may add all other rewrite options in S and find that the end-transformation would be consistently $haah$ for all possible derivations. **Figure 10a** shows the shortest possible derivations of all rewrite options for the rule 3 in G_7. **Figure 10b** shows some derived strand from G_7.

The grammar G_7 generates a language of all closed strands with pseudo-branches consist of only h and a concatenations.

2.6 Syntax-directed translation

The expressive power of context-free grammar is restricted by the syntactic structure of the parse tree. Information that is needed in deriving symmetric patterns cannot be sent across derivative branches in the parse tree. Syntax-directed translation is a means to derive information from a parse tree and use it in syntactic operations for analysis or generation. For example, it is impossible to define a context-free grammar that generates sets of strands with three or more correlated derivations such as $h^n ah^n ah^n$, or with two or more correlated derivations that are not nested such as $ah^n ah^m ah^n ah^m$. **Figure 11** shows two translations that map two input strands $(ah^2a)^2$ and $(ah^4a)^2$ derived by G_2 to output strands defined by G_9 and G_{10} through input grammar G_8. In the process of translation, G_8 is used to parse input strands. Output strands are generated by replacing rewrite options of the input grammar in the parse tree with corresponding rewrite options in the output grammars.

Input Grammar	Output Grammar 1	Output Grammar 2
$G_8 : X$	$G_9 : X$	$G_{10} : X$
$X = aY_1aY_2$	$X = aY_1aY_2$	$X = aY_1aY_2$
$Y_1 = hhY_1 + a$	$Y_1 = hhY_1hh + a$	$Y_1 = ssY_1ss + a$
$Y_2 = hhY_2 + a$	$Y_2 = hhY_2hh + a$	$Y_2 = ssY_2ss + a$

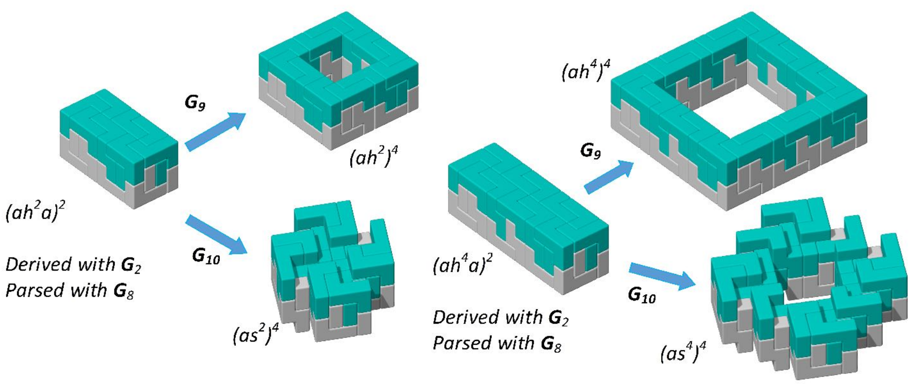

Figure 11: Examples of syntax-directed translations.

3 Results

A grammar that creates frame structures is defined in this section. A set of variables are defined as the following list for specific parts of the frame structure. The set of variables are combined to define G_{11} that is used to derive frame structures shown in **fig. 12**. **Figure 12a-c** show structures with columns, cantilevers, beams, and pitched girders. **Figure 12d** shows a photo of a model assembled with 8mm*16mm*12mm *SL* blocks. The model consists of two *SL* strands that are identical in shapes and with one in yellow and one in blue for colors. Both strands are derived by G_{11} with the option of having a base F at the bottom of the column.

$$
\begin{aligned}
&G_{11} : X \\
&1. X = (1 + F)Y \\
&2. Y = Ct(dBd + dHd + S + dhSd + P)sY + C
\end{aligned}
$$

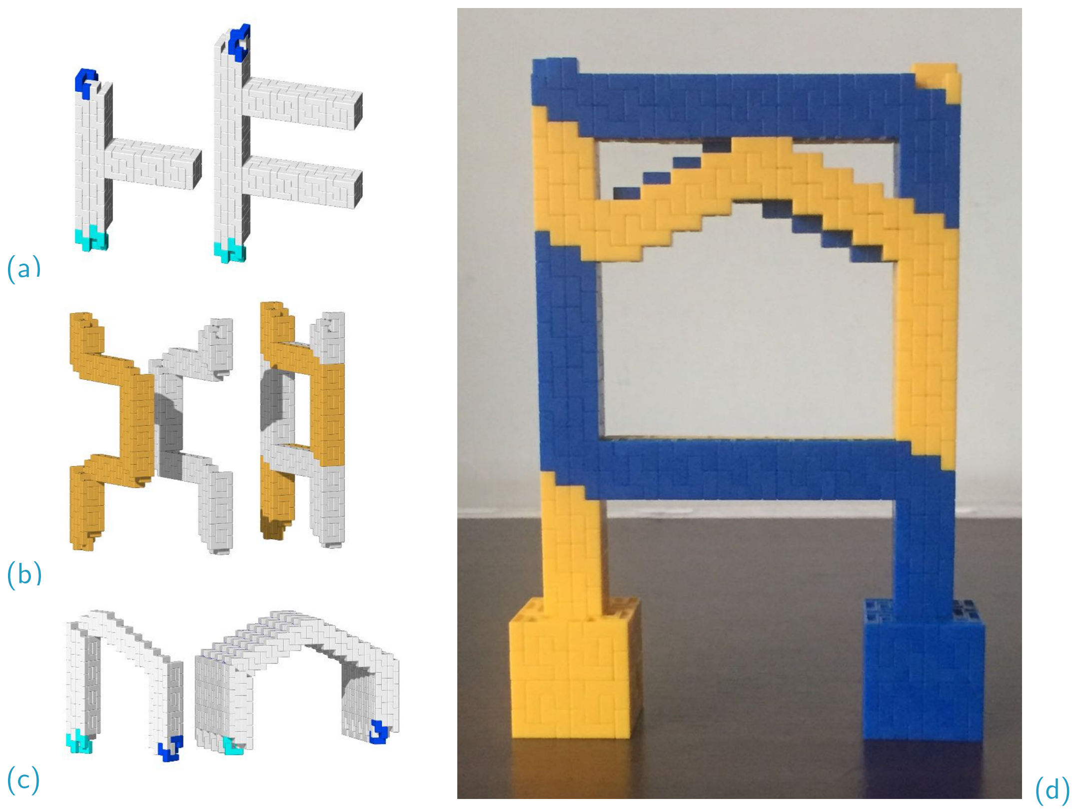

Figure 12: (a) Cantilever structures (b) Combining two strands to make frame structure (c) structures with sloped and pitched girders (d) a photo of an assembled model based on an extended version of G_{11}.

1. a helical strand for column with various height
   $C = tsC + ts$
2. a cantilever beam with various length
   $B = hhBhh + aa$
3. a half beam with various length
   $H = hHh + h$
4. a sloped lintel
   $S = dS + d$
5. a pitched girder
   $P = dPd + h$
6. a column foundation
   $F = ahhahhahhaadstdaF/ahhahhahhaads$

Figure 13 shows three grammars generating tree-like strands based on the same structure with a helical strand as the trunk and folding strands extruding from four directions as branches. All three grammars are devised by variable insertion described in prior sections. **Figure 14** shows an application of syntax-directed translation for a strand generated by the grammar G_{12}. In the translation process, the grammar G_{15} is used to parse the input strand shown in **fig. 14a**. The resulting parse tree is used to guide the derivations of G_{16} to generate the output strand shown in **fig. 14b**.

Input	Output
$G_{15}: X$	$G_{16}: X$
$X = thX + tahY_1aaY_2hdhX + th$	$X = thX + tatY_1aY_2tadhX + th$
$Y_1 = hhY_1 + 1$	$Y_1 = ssY_1ss + a$
$Y_2 = hhY_2 + 1$	$Y_2 = ssY_2ss + a$

4 Conclusion

It is attempted to establish a theoretical basis for *SL* block compositions. The polynomial notation for grammars is found to be convenient and revealing. The properties of end transformation and being fixed ending of grammars are very helpful for devising grammars that are meaningful and interesting for *SL* block compositions. Future works are needed for clearer definitions, theorems proving, and semantic checking based on syntactic operations.

$$
\begin{aligned}
G_{12}: \quad & X \\
X = \quad & thX + tahBhdhX + th \\
B = \quad & hhBhh + aa
\end{aligned}
$$

$$
\begin{aligned}
G_{13}: \quad & X \\
X = \quad & thX + tahLhdhX + th \\
L = \quad & hSh + aa \\
S = \quad & hLh + aShah + hahSa
\end{aligned}
$$

$$
\begin{aligned}
G_{14}: \quad & X \\
X = \quad & thX + tahLhdhX + th \\
L = \quad & hSh + aa \\
S = \quad & hLh + aShah + hahSa \\
& + aSaShh + aShhSa \\
& + hhSaSa + aSaSaSa
\end{aligned}
$$

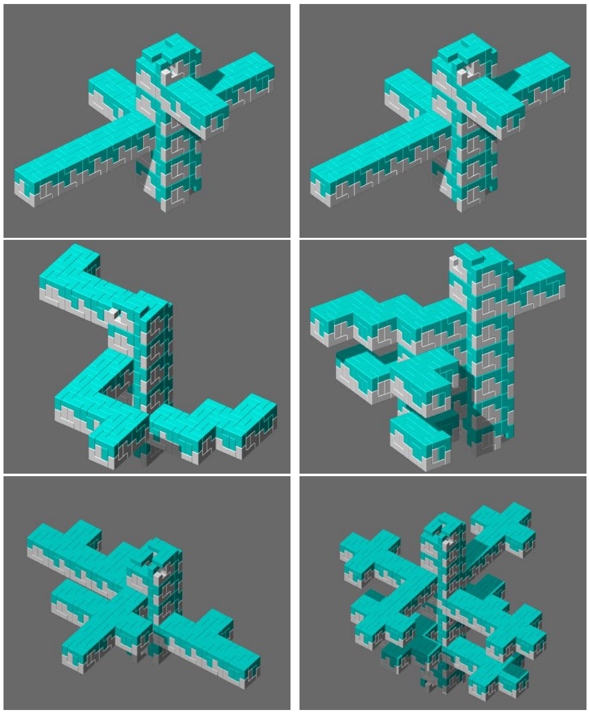

Figure 13: Three grammars generating tree-like strands.

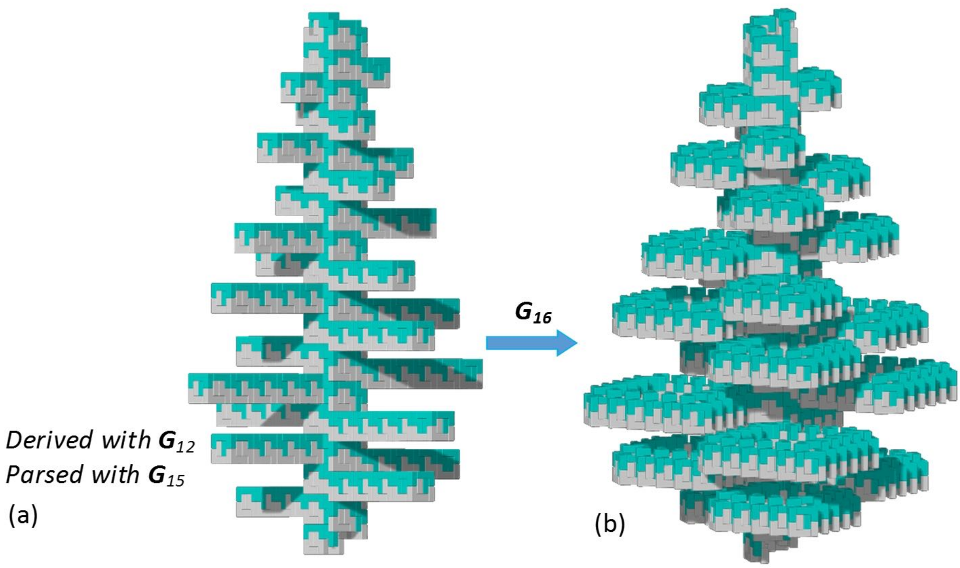

Figure 14: An application for syntax-directed translation. (a) the input strand of the translation, (b) the output strand of the translation.

References

Dyskin, A., V. Dyskin, Y. Estrin, A. Kanel-Belov, and E. Pasternak (2003). Topological interlocking of platonic solids: A way to new materials and structures. *Philosophical Magazine Letters*.

Estrin, Y., A. Dyskin, and E. Pasternak (2011). Topological interlocking as a material design concept. *Mater. Sci. Eng. C, Principles and Development of Bio-Inspired Materials 31*, 1189–1194.

Fu, C., X. Yan, L. Yang, P. Jayaraman, and D. Cohen-Or (2015). Computational interlocking furniture assembly. *Journal of ACM Transactions on Graphics (TOG), Proceeding of ACM SIGGRAPH 2015*.

Kanel-Belov, A., A. Dyskin, Y. Estrin, E. Pasternak, and I. Ivanov-Pogodaev (2010). Interlocking of convex polyhedra: towards a geometric theory of fragmented solids. *Moscow Mathematical Journal 10*(2), 337–342.

Lo, K.-Y., C.-W. Fu, and H. Li (2009). 3d polyomino puzzle. *ACM Trans. Graph. 28*(5).

Shih, S. (2016). *Advances in Architectural Geometry 2016*, Chapter On the Hierarchical Construction of SL Blocks – A Generative System that Builds Self-Interlocking structures. Hochschulverlag AG an der ETH Zürich.

Shih, S. G. (2018). The art and mathematics of self-interlocking sl blocks. In *Bridges 2018 Conference Proceedings*, pp. 107–114.

Song, P., C. Fu, and D. Cohen-Or (2012). Recursive interlocking puzzles. *ACM Trans. Graph. 31*(6).

Weizmann, M., O. Amir, and Y. J. Grobman (2017). Topological interlocking in architecture: A new design method and computational tool for designing building floors. *International Journal of Architectural Computing 15*(2), 107–118.

Yong, H. (2011). *Utilisation of topologically-interlocking osteomorphic blocks for multi-purpose civil construction*. Ph. D. thesis, University of Western Australia.