The Simple Generative Logic Grammar: A tool for teaching logical thinking through visual research in art and design.

Christian Jendreiko[1]

[1]HSD, University of Applied Sciences, Düsseldorf, Germany

Abstract

The Simple Generative Logic Grammar (SGLG) is a newly developed, very simply structured type of logic grammar designed for generating structured visual compositions. This paper examines its key features and explores its potential to play a central role in the educational framework Exploring Generative Logic (EGL) that I am currently developing. [1] At the heart of EGL is the idea of using the artistic creation of images composed of discrete elements to introduce students particularly in art and design to fundamental concepts in logic programming, formal grammars, and database systems in an accessible way.

Keywords

Generative Logic, Logical Thinking, Logic Programming, Prolog, Visual Research, Paul Klee, Teaching Prolog, Generative Design, Computer Art

1. Introducing SGLG: A Simple Generative Logic Grammar.

In my paper "Generative Logic: Teaching Prolog as Generative AI in Art and Design" [1] that I presented last year at the ICLP Education workshop, I briefly touched upon a simple, easy to use generative grammar that I developed and that we had used in my introductory course at the time to create what I did call "electronic mosaics." [1]

In the months following the presentation, through continued exploration and application of this grammar in concrete teaching scenarios, I increasingly realized its potential to become a central teaching tool within my educational framework, Exploring Generative Logic (EGL). [1] This framework, which I am currently developing at the HSD, is tailored especially for introducing Logic Programming and logical thinking to art and design students through visual research. It places the generative potential of Logic Programming into focus.

Therefore, I decided to take a closer look at this grammar in the present paper, referring to it provisionally Simple Generative Logic Grammar (SGLG).

This paper aims to pursue the following goals:

1. Giving a closer look at the properties of the grammar, providing a formal definition, and reflecting on the initial inspiration behind its design
2. Showing how the characteristics of this grammar can be effectively used in the context of EGL education as well as in visual research
3. Sharing recent experiences working with the grammar—particularly in my currently ongoing introductory course on Prolog, which is based on my EGL framework and where we use the SGLG as a central teaching tool
4. Providing an outlook on possible projects that combine visual research and education in logic programming within the framework of the EGL concept.

PEG 2025: 3rd Prolog Education Workshop, September, 2025, Rende, Italy.

christian.jendreiko@hs-duesseldorf.de (C. Jendreiko)

0009-0006-4302-6261 (C. Jendreiko)

2. SGLG: The key concepts.

The Simple Generative Logic Grammar (SGLG) is a simple non-recursive type of grammar designed to generate structured visual output in two dimensions —such as images, shapes, mosaics, or patterns.

The concept underlying the design of the SGLG is the idea that the total area of a 2-D visual object such as a picture can be composed of segments, each capable of containing a visual sign. On a more abstract level, the grammar can be understood as a tool that can be used to set up an elementary structure serving as a container structure.

The grammar is simple in two ways: The idea of composing an aesthetic object out of its parts is easy to understand and the grammar in that respect is easy to use and easy to handle.

It is also simple in the sense that, unlike many other visual grammars, it is non-recursive. This is also the reason why I do not draw on the extensive body of literature on visual grammars in this paper. Because what all of those grammars have in common is that they are recursive in nature.

These features makes SGLG an essential tool for learning and for visual research within the educational framework of Exploring Generative Logic (EGL).

What is also interesting from a visual research point of view: SGLG produces a distinct aesthetic, as the individual visual elements that make up the output remain perceptible as discrete, isolated objects.

3. SGLG: The Core Structure.

The Simple Generative Logic Grammar is defined by the following components:

G = V, Σ, P, S, M, L

Where:

V is a set of non-terminal symbols,
Σ is a set of terminal symbols,
P is a set of production rules,
S is the start symbol, representing the total area of visual output in 2-D,
M is a mapping from non-terminal symbols to sets of concrete terminal symbols (elements of Σ),
the non-terminals represent sub-areas of the total area of a 2-D visual object to be generated.
L is a set of specific layout-symbols,
L contains at least the symbol n that is used within production rules wherever necessary to ensure that the outputs of individual rules are arranged vertically. This enables the emergence of a two-dimensional structure.

3.1. Generating visual output in 2-D with the SGLG.

The concept underlying the design of the SGLG is the idea that the total area of a 2-D visual object such as a picture can be composed of segments, each capable of containing a visual sign. Two commented code examples of SGL grammars, along with their output, can be found in Chapter 8.

The 2-D visual object generation occurs in two main steps:

Step 1: Defining the Format and Structure.

In the first step, the grammar defines the total area of the 2-D visual output — including its size, shape, and the arrangement of sub-areas that define the segments composing the entire image.

The 2-D visual output is constructed row by row. Generation begins with the application of the production rules that generate these rows.

These production-rules are divided into two types:

Row Content Rules ($R\_row \subseteq P$):
These define how an individual row is composed from sub-area symbols. Each rule replaces a row symbol on the left-hand side with a sequence of sub-area symbols on the right-hand side.

Row Sequence Rules ($R\_seq \subseteq P$):
These define how the full 2-D visual object is composed from rows. Each rule replaces a higher-level row-sequence symbol with an ordered sequence of row symbols, effectively constructing the vertical structure of the visual object.

The start symbol S represents the overall area of the visual object.

This system constructs a fixed, two-dimensional structure: a predefined arrangement of sub-areas that built the overall area of the complete visual object.

Step 2: Assigning Visual Vocabulary.

Once the structure is defined, a visual vocabulary is introduced through the terminal symbols:

The Σ-set contains graphical symbols that serve as the terminal elements of the grammar. Each sub-area symbol, which represents one segment of the total area of the 2-D object, is associated — via the mapping M — with a subset of these terminal symbols. Thus, the sub-area symbols function as placeholders for variable visual content.

Each SGLG instance defines a single, fixed area structure — a spatial template for 2-D visual object generation. However, the visual vocabulary defined by the Σ- set can be modified dynamically: alternative graphical symbols may be added to or removed from subsets. Additionally, the mapping M, which links sub-area symbols to these subsets, can also be adjusted. The mapping may be nondeterministic, allowing each segment of the total area to take on different visual realizations during generation.

This flexibility enables a single structural template to generate a wide variety of visual objects — all belonging to the same structural class — making the system both structurally constrained and visually expressive.

The selection of the graphical symbols, their grouping into meaningful subsets of Σ, and the mapping of these subsets to sub-area symbols are entirely user-defined.

This allows for flexible interpretation and creative control over the visual content generated

by the grammar, making it a powerful yet easy-to-use tool for visual research.

3.2. How SGLGs differ from Context-Free Grammars.

Although SGLGs share superficial similarities with context-free grammars, they differ in two foundational ways:

1. Fixed, Non-Recursive Structure
A SGLG does not rely on recursion or unbounded rule application. The structure of the 2-D visual object —the number of rows and sub-areas— is fixed and explicitly defined. This contrasts with context-free grammars, where the generative power is often tied to recursion and potentially infinite depth or length.

2. Open and Dynamic Vocabulary
The terminal vocabulary Σ is not fixed in advance. The assignment of visual signs to sub-area symbols is user-defined and modular, allowing for reuse and reconfiguration. This flexibility supports ongoing visual experimentation and variation within a stable structural template.

4. SGLG: An educational tool for Cross-Domain Knowledge Integration.

Recent experiences in using SGLG stem from my current teaching in the summer semester, where I am employing the tool in an introductory Prolog course. The class consists of a moderate number of 12 students at both BA and MA levels; except for one, none have prior programming experience. Students use SGLG to construct image structures and populate them with graphical elements, which they assemble independently and assign to specific placeholders within a structural template. There is lively participation, with students actively presenting their own program designs and reporting on related topics.

4.1. SGLG as an Interface between Visual and Logical thinking.

The students' questions, comments, and program drafts clearly show that using SGLG helps them begin to connect visual thinking with logical reasoning. They start to develop a deeper understanding of how diverse domains of knowledge that initially seem unrelated can converge in the digital design process of image composition through the use of a formal grammar.

The grammar functions as an interface that mediates between visual and logical thinking. Through their work with SGLG, students begin to understand how concepts of logical thinking can serve as tools for visual research. Fundamental strategies of image construction are explored by engaging with the practice of describing images declaratively and by applying the generative power of analytical decomposition. Decomposing the image plane into meaningful segments and organizing graphic elements according to semantic or compositional criteria serves as a concrete implementation of the abstract principle of breaking down a whole into individual, logically structured units and reassembling it. In doing so, they naturally engage with further fundamental ideas from logic programming, computer science, and visual research within an interdisciplinary framework. From logic programming, they draw on core concepts such as logical inference, symbolic representation, and the operational semantics of Prolog. From computer science, they engage with formal grammars and rule-based systems, discovering how structure can drive aesthetic expression. All this reinforces analytical thinking through visual methods and deepens the understanding of how visual information can be structured logically. By combining these distinct domains, students develop an integrated practice in which analytical, technical, and aesthetic perspectives converge in a design process of digital image structures, balancing logical rigor with artistic intuition.

4.2. Making Prolog's Execution Mechanism Visible.

A key consideration for keeping the grammar structure as simple as possible is to make Prolog's execution mechanism transparent and easy to grasp. This mechanism can be studied effectively through the grammar because the sequence in which variations of an image structure are generated directly reflects Prolog's execution behavior, where backtracking, unification, and depth-first search interact. The execution mechanism becomes directly visible in the visual output, as the output is essentially the trace left by Prolog's reasoning process. The screen becomes a canvas for the Prolog engine's logic, turning abstract operations into concrete visual expression. This sparks students' interest in the execution mechanism of a Prolog system and opens the door to meta-programming. Students confront the execution process, which contrasts their intuitive expectations about when and how image variations should appear with the system's rigid, rule-based regime. This experience encourages a purposeful use of the system-conditioned properties and behaviors of a Prolog system—provided they become familiar with it first.

4.3. Exploring recursion with a non-recursive grammar.

My recent experiences in the current course shows to me, that SGLG seems to provide an ideal starting point for engaging with recursion. Although the grammar itself is non-recursive, students naturally discover recursion as a generative principle for image creation when they realize that an image within SGLG can be built from rows of rows. This does raise students' interest in taking the next step toward using a recursive grammar.

4.4. Experiencing the Generative Power of Prolog.

Through experimentation with SGLG, students experience firsthand the generative power of Prolog. In the simplest case, each subset of the symbol set Σ contains only one visual element. Here, the grammar yields a single output, as each symbol is placed into a unique image segment.However, once a single subset contains multiple elements, the full potential of Prolog as a generative engine becomes apparent. The system begins generating all valid combinations by systematically exploring every possibility. This use of the inference engine as a creative generator which is central to the concept of generative layout reveals Prolog's capacity for creative problem-solving and design iteration.

5. SGLG: A gateway to the classic concepts of generative art.

The SGLG provides an excellent starting point for engaging with classical concepts and methods for generating aesthetic objects from the pioneering days of computer art. This is primarily because the original idea behind designing the grammar was inspired by a specific class of early generative programs known for their ability to produce complex results despite being "of the simplest structure." [3]

In his classic book on fundamental principles of generative design, "Ästhetik als Informationsverarbeitung", Frieder Nake describes two programs of this class, both of which he wrote in 1969 in Toronto. [3] One he developed alone, the other in collaboration with Mary Gardner, who designed the symbols for the program. [3]

About the first program, he writes:
"Apparently, the generative aesthetics of this program are of the simplest kind: it places a fixed symbol at predetermined locations... all relevant decisions have been shifted to the phase prior to the generative aesthetics. This type of generative aesthetics is particularly common. It is characterized by the complete absence of random generators. The corresponding programs depend on a number of deterministic parameters." [3]

And about both programs he writes:
"Although both examples must be classified as 'low' from the standpoint of generative aesthetics, their outputs are surprisingly interesting. This is likely due to the symbol repertoires used as well as the transformations applied to the symbols: although these only determine the locations of the selected symbols, arrangements emerge that are complex enough to hold the viewer's interest for some time. In both examples, the symbol repertoires were determined by artists. I see this as an indication that generative aesthetics of the simplest structure, when based on an artistically chosen symbol repertoire, can lead to more stimulating aesthetic objects than when all components are exclusively set by programmers." [3]

According to Frieder Nake's account, a generative program of this class is characterized by the following four key features:

1) It is of the simplest structure.

2) It places fixed symbols at predetermined locations.

3) It is marked by the complete absence of random generators.

4) The symbol repertoires are defined by artists. [3]

It was immediately clear to me that a generative program with these properties could easily be modeled in the form of a logic grammar. This insight became the starting point for the development of the SGLG. Instead of using random generators to produce output variations, the inference engine is used as a generator.

As a consequence, SGLG is ideally suited to be used to experiment with these generative concepts from the pioneering era of computer art. This allows students to place their own approach within an art-historical context and to gain hands-on understanding of the fundamental concepts of this art form. The use of SGLG anchors students' own designs conceptually within the history of generative design and computer art.

6. Further Investigations: Visual research on the notes of Paul Klee.

At the beginning of the current seminar, it became evident to me that the abstract concept of image creation underlying SGLG aligns perfectly with the fundamental principles of image composition in Classical Modernism, particularly the idea that the total surface of a picture can be understood as a composition formed by its subdivision into sub-areas, each capable of containing a visual sign.

In this regard, SGLG is ideally suited for experimenting especially with the methods of picture construction outlined by Paul Klee, one of the most important figures of Classical Modernism. Klee articulated these methods in his Pedagogical Sketchbook and in the teaching notes for his seminar Elementare Gestaltungslehre der Fläche (Elementary Design Theory of the Plane), which he taught during his tenure as a professor at the Bauhaus in Weimar and Dessau between 1921 and 1931. [2]

Making these notes the object of study opens up a broad interdisciplinary field of visual research, in which visual inquiry and engagement with logical grammars and their practical applications intersect in exploring and experimenting with Paul Klee's ideas of image construction. In this context, SGLG serves not only as an educational tool but also as an instrument for deep visual research.

In my current introductory course, we took some cautious first steps into this field of research to explore which paths might open up and how far we could go. The initial impression: very far, along many branching paths. We will officially launch the project in the winter semester, and a report on our findings is planned for publication next year.

7. Conclusion: Connecting logical thinking with visual thinking using SGLG.

My recent experiences using SGLG in my introductory Logic Programming course have led to the following conclusions:

SGLG is an excellent tool for connecting visual thinking with logical thinking and for exploring the relationships between the two in the form-finding process. It serves as an eye-opener in multiple ways and can be effectively used to introduce fundamental concepts across the diverse fields that converge in generative design.

Students gain not only technical understanding of logic programming, formal grammars, and database systems—such as syntax, rules, and inference—but also cultivate deeper habits of logical abstraction, modular reasoning, and formalization.

The integrative power of the grammar stems from the idea of implementing fundamental principles of image composition in the form of a generative grammar within a logic programming context. This approach makes complex, abstract concepts more accessible and inclusive, offering a barrier-free entry point into interdisciplinary thinking.

The ultimate goal of my teaching approach, EGL, is to empower students to apply logical reasoning in both artistic creation and experimental form-finding—promoting a mode of thinking that is inherently interdisciplinary and critically relevant to both visual and computational fields.

Integrating SGLG as a central didactic tool appears to be a sound decision in moving closer to that goal.

8. SGLGs by Example: Grammar Codes and Visual Outputs.

```
%CJ, Simple Generative Logic Grammar I, 24.05.2025

%picture s composed of rows r1 to r5:

s --> r1,r2,r3,r4,r5,n.

%rows, composed of segments:

r1 --> a,a,a,a,a,n.
r2 --> a,b,b,b,a,n.
r3 --> a,b,c,b,a,n.
r4 --> a,b,b,b,a,n.
r5 --> a,a,a,a,a,n.

%assignment of segments to graphic signs
%(in this case: to 3 singleton subsets of ∑):

a --> [🟦].
b --> [🟩].
c --> [🟥].


%line break:
n --> ['\n'].


%output of generated picture:
p :- phrase(s, Ls), format("~s", [Ls]).

% Query: p.
```

Figure 1: Grammar Code Example 01: Basic grammar using singleton subsets of the Σ-set. The Σ-set consists exclusively of singleton sets.

Figure 2: Output of Example 01. The output is a square containing exactly one placement configuration of graphic elements from the Σ-set.

```
%CJ, Simple Generative Grammar II 24.05.2025
%with 2 sign-classes of 2

%picture s composed of rows r1 to r5:

s --> r1,r2,r3,r4,r5,n.

%rows, composed of sub-areas:

r1 --> a,a,a,a,a,n.
r2 --> a,b,b,b,a,n.
r3 --> a,b,c,b,a,n.
r4 --> a,b,b,b,a,n.
r5 --> a,a,a,a,a,n.

%assignment of segments to graphic signs
%(in this case: 2 singleton subsets of ∑
%and 1 subset with 2 members):


a --> [🟦] | [🟥].
b --> [🟩].
c --> [🟥].


%line break:
n --> ['\n'].

%output of generated picture:
p :- phrase(s, Ls), format("~s", [Ls]), fail.

% Query: p.
```

Figure 3: Grammar Code Example 02: Same grammar, but with one subset of the Σ-set containing two graphic elements.

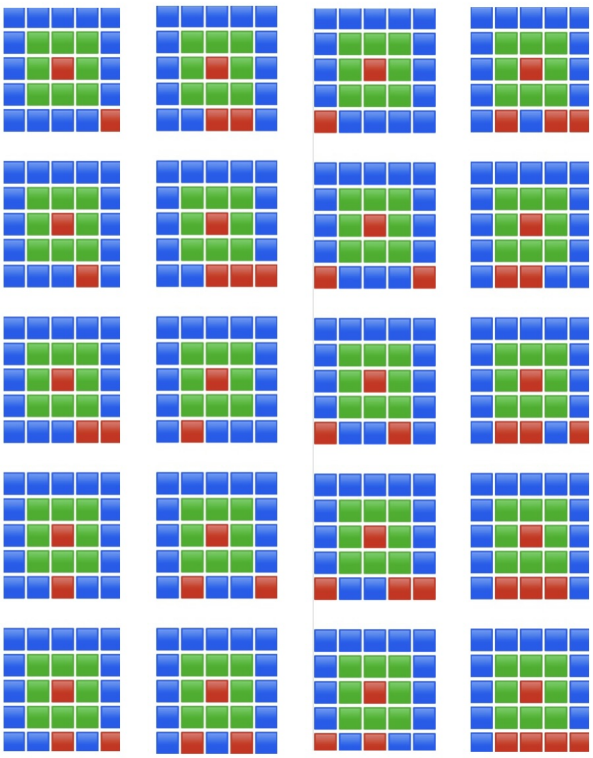

Figure 4: Grammar Code Example 02: Displayed here are the first 16 output-variations.

Figure 5: Two output variations were generated using the same Simple Generative Logic Grammar by replacing the Σ-set. The code was written by BA student Constantin Rieger in my Prolog introductory course at HSD, Summer Semester 2025.

Acknowledgments

I would like to express my deep gratitude to Frieder Nake and David Warren. This work has been profoundly shaped by extensive and enlightening conversations with both of them, which significantly contributed to the development of the ideas presented in this paper. I am also grateful to the PEG 2.0 education group for their generous exchange of ideas, which has been a continual source of inspiration. I am also grateful to the reviewers for their insightful feedback on the first version of the paper. Special thanks go to Carsten Heisterkamp for his valuable assistance in refining the manuscript. Finally, I would like to thank my students, whose engagement and insights often lead my ideas in directions I could not have anticipated.

Note on Translations

All translations from non-English sources were made by the author.

Declaration on Generative AI

During the preparation of this work, the author used ChatGPT in order to: translate text and, grammar and spelling check. After using this tool, the author reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] Jendreiko, Christian. "Generative Logic: Teaching Prolog as Generative AI in Art and Design". In: *Workshop Proceedings of the 40th International Conference on Logic Programming (ICLP-WS 2024) co-located with the 40th International Conference on Logic Programming (ICLP 2024)*. Ed. by Arias, J. et al. Vol. 3799. CEUR Workshop Proceedings. Dallas, TX, USA: CEUR-WS.org, Oct. 12, 2024. URL: https://ceur-ws.org/Vol-3799/paper9PEG2.0.pdf.

[2] Klee, Paul. *Unendliche Naturgeschichte. Form- und Gestaltungslehre Band 2*. Ed. by Spiller, Jürg. Stuttgart/Basel: Schwabe & Co, 1970.

[3] Nake, Frieder. *Aesthetik als Informationsverarbeitung*. Wien/New York: Springer, 1974.