UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Paloma Zankely Alves de Oliveira Cione

Um algoritmo para geração de projetos arquitetônicos utilizando gramática de formas

Florianópolis
2024

Paloma Zankely Alves de Oliveira Cione

Um algoritmo para geração de projetos arquitetônicos utilizando gramática de formas

Trabalho de Conclusão de Curso do Curso de Graduação em Ciências da Computação do Campus Florianópolis da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Ciências da Computação.
Orientador: Prof. Dr. Rafael de Santiago

Florianópolis
2024

Ficha de identificação da obra

A ficha de identificação é elaborada pelo próprio autor.

Orientações em:

http://portalbu.ufsc.br/ficha

Paloma Zankely Alves de Oliveira Cione

Um algoritmo para geração de projetos arquitetônicos utilizando gramática de formas

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Ciências da Computação" e aprovado em sua forma final pelo Curso de Graduação em Ciências da Computação.

Florianópolis, Julho de 2024.

---

do Curso

Banca Examinadora:

Prof. Dr. Rafael de Santiago
Orientador

Prof. Dr Maicon Rafael Zatelli
Avaliador
UFSC

Prof. Dr Alvaro Junio Pereira Franco
Avaliador
UFSC

RESUMO

A constante digitalização e automatização de métodos diversos impacta a sociedade em múltiplos setores, influenciando também a arte e outros processos criativos. Gramáticas de formas, na teoria da computação, são uma classe de gramáticas que possuem um conjunto de regras que geram formas geométricas em suas produções. Com isto, nas últimas décadas, gramáticas de formas são consideradas uma ferramenta poderosa para o processo de criação na arquitetura, se tornando parte do conjunto do design paramétrico, em que recursos são moldados com base em um algoritmo. Nesse contexto, o objetivo principal deste trabalho é criar um algoritmo que reflita em criações arquitetônicas, como casas, prédios, e outros edifícios, se baseando em um estilo específico definido, analisando sua complexidade computacional e possíveis barreiras tecnológicas que possam ser encontradas.

Palavras-chave: Gramáticas de forma, Arquitetura, Design paramétrico, Automatização, Geração Procedural

LISTA DE FIGURAS

LISTA DE TABELAS

LIST OF ALGORITHMS

SUMÁRIO

1 INTRODUÇÃO

Com a sua criação em meados de 1970 (BONACIC; GIPS; STINY, 1978), gramáticas de formas foram introduzidas originalmente como uma maneira de analisar e sintetizar a arte, entendendo e identificando seus padrões e formas. Desde então, a utilização de gramáticas de formas foi ampliada e é altamente eficaz na produção de *design* e geração de formas com o intuitivo criativo, estético e visual.

Para o *design* e a arte autogerada, é suficiente e desejável a elaboração de formas que se entrelaçam e se sobrepõem, criando novas formas a partir das anteriores e assim criando infinitos *designs* que podem ter as mais diversas faces. Porém, em um projeto arquitetônico, é necessária a limitação de possibilidades e uma melhor definição destas. Por exemplo, como se é dito em (PAUWELS *et al.*, 2015), a demarcação de paredes e cômodos suficientemente grandes e funcionais. Podemos então definir que projetos arquitetônicos como casas e prédios tem sua diferença dos *designs* comuns gerados por gramáticas de forma baseada na funcionalidade de tais estruturas, por isso, desde sua criação e até o presente momento, diferentes gramáticas que analisam e descrevem estilos arquiteturais foram desenvolvidas, visando também a possibilidade do entendimento genérico sobre a aplicação dessas regras.

1.1 OBJETIVOS

Nas seções abaixo estão descritos o objetivo geral e os objetivos específicos deste TCC.

1.1.1 Objetivo Geral

O objetivo principal do projeto é o desenvolvimento de um algoritmo que produza de maneira automatizada as derivações de uma gramática de formas, criando projetos arquitetônicos de acordo com as regras da gramática, de modo que essa ferramenta impacte e auxilie tanto a criatividade quanto a inovação na nossa realidade.

1.1.2 Objetivos Específicos

1. desenvolver uma implementação de um algoritmo que gere produções de uma gramática de formas;
2. expandir o algoritmo para que este consiga usar as limitações de projetos arquitetônicos;
3. conseguir criar formas que condizem com um estilo arquitetônico previamente definido;
4. avaliar o desempenho do algoritmo de forma a verificar se a aplicação é viável;

5. disponibilizar código-fonte e aplicação de maneira acessível ao público interessado;

1.2 METODOLOGIA

A metodologia de pesquisa aplicada neste projeto é feita pela seguinte maneira:

1. revisão bibliográfica sistemática referente aos temas-chave, visando entender melhor a implementação e os passos necessários.
2. desenvolvimento de um algoritmo que gere as produções de uma gramática de formas, adaptando-o para a geração de projetos arquitetônicos definidos pelas gramáticas de entrada;
3. desenvolver experimentos e coletar os resultados
4. comparar e analisar os resultados a fim de melhorar a aplicação;
5. descrever e reportar os resultados obtidos;
6. redigir relatórios parciais referentes às disciplinas de Introdução ao TCC e TCC I;
7. redigir a monografia e apresentá-la para banca avaliadora no fim da disciplina de TCC II.

1.3 ESTRUTURA DO TRABALHO

Além do presente capítulo, que trata da introdução e apresentação do projeto, este trabalho está estruturado em cinco capítulos, a serem explicados a seguir.

O capítulo 2 apresenta os conceitos básicos para o entendimento do trabalho. De uma forma superficial, são expostos os conceitos de gramáticas, gramáticas de formas, transformações lineares de formas, gramáticas de formas na teoria da computação e como modelo computacional.

No capítulo 3 são apresentados os trabalhos relacionados à gramáticas de formas na arquitetura.

No capítulo 4 é apresentado um pseudo-código referente ao algoritmo base criado para geração de gramáticas de formas, que será a fonte do nosso objetivo final para a geração de projetos arquitetônicos.

O capítulo 5 apresenta conclusões desse trabalho, e planeja o desenvolvimento futuro.

2 FUNDAMENTAÇÃO TEÓRICA

Esse capítulo irá apresentar conceitos fundamentais para o entendimento do trabalho e a base teórica que possibilitam que os objetivos sejam alcançados

2.1 GRAMÁTICAS

Como definido por Chomsky (CHOMSKY, 1956), uma gramática é um mecanismo que define o conjunto de regras de uma linguagem, que dita também sua estrutura, de forma a permitir que apenas determinadas combinações sejam válidas, isto é, sejam consideradas sentenças e parte da linguagem especificada. A produção de elementos definidos nestas regras forma uma gramática generativa, capaz de formar sentenças completas. Gramáticas também são um mecanismo importante na representação de linguagens potencialmente infinitas de forma finita, apenas com um conjunto de regras que podem gerar toda a linguagem.

Definição formal de gramática: G = (N, T, P, S)

- N: conjunto de variáveis não-terminais
- T: conjunto de variáveis terminais ($N \cap T = \emptyset$)
- P: conjunto finito de produções $P \subseteq (N \cup T = \emptyset)^+ \rightarrow (N \cup T = \emptyset)^*$
- S: símbolo inicial $S \in N$

⟨SENTENCE⟩ → ⟨NOUN-PHRASE⟩⟨VERB- PHRASE⟩
⟨NOUN-PHRASE⟩ → ⟨CMPLX-NOUN⟩ | ⟨CMPLX- NOUN⟩⟨PREP-PHRASE⟩
⟨VERB-PHRASE⟩ → ⟨CMPLX-VERB⟩ | ⟨CMPLX- VERB⟩⟨PREP-PHRASE⟩
⟨PREP-PHRASE⟩ → ⟨PREP⟩⟨CMPLX-NOUN⟩
⟨CMPLX-NOUN⟩ → ⟨ARTICLE⟩⟨NOUN⟩
⟨CMPLX-VERB⟩ → ⟨VERB⟩ | ⟨VERB⟩⟨NOUN-PHRASE⟩
⟨ARTICLE⟩ → a|the
⟨NOUN⟩ → boy|girl|flower
⟨VERB⟩ → touches|likes|sees
⟨PREP⟩ → with

Figura 1 – Exemplo de gramática. Fonte: (SIPSER, 2013)

Na Figura 1, a partir do símbolo inicial **<SENTENCE>**, é possível gerar cadeias que pertencem a linguagem:

- a boy sees
- the boy sees a flower

- a girl with a flower likes the boy

2.1.1 Classificação de gramáticas: Hierarquia de Chomsky

A hierarquia de Chomsky denomina uma classificação de gramáticas e linguagem introduzida em meados de 1950 por Noah Chomsky (CHOMSKY, 1956) em seu trabalho de gramáticas em estruturas de frases. Essa classificação pode ser vista como uma hierarquia de níveis de computabilidade e complexidades e portanto se tornou fundamental tanto no campo da linguística quanto no da computação.

- Tipo 0: Irrestrita
  - $(N \cup T)^+ \rightarrow (N \cup T)^*$
    Gramáticas irrestritas incluem todas as gramáticas formais, e não possuem nenhuma restrição. Gramáticas desse tipo geram exatamente todas as linguagens que podem ser reconhecidas por uma Máquina de Turing.
- Tipo 1: Sensível ao Contexto
  - $(N \cup T)^+ \rightarrow (N \cup T)^+$
  - $|\alpha| \leq |\beta|$
    As regras de uma gramática sensível ao contexto definem que um conjunto positivo de não terminais ou terminais possa gerar também um conjunto positivo de não terminais ou terminais, desde que o lado esquerdo seja menor que o lado direito. Uma gramática desse tipo é reconhecida por um Autômato Linearmente Limitado.
- Tipo 2: Livres de Contexto
  - N $\rightarrow (N \cup T)^+$
    As gramáticas livres de contexto são definidas por regras de forma que um não terminal possa gerar uma cadeia de não terminais ou terminais de qualquer tamanho positivo. Essas linguagens são a base para a estrutura de frase da maioria das linguagens de programação e são reconhecidas por um Autômato de Pilha Não Determinístico.
- Tipo 3: Regulares
  - N $\rightarrow$ a|aN, a $\in T$
    Uma gramática regular restringe suas regras para que se tenha um único não-terminal ao lado esquerdo, e, ao lado direto, um único terminal que pode ou não ser seguido por um único não terminal. Linguagens regulares são reconhecidas por um Autômato Finito.

2.2 GRAMÁTICAS DE FORMAS

No mesmo sentido do mecanismo de geração utilizado nas gramáticas generativas, as gramáticas de forma são representações visuais e geométricas de gramáticas, com regras definidas espacialmente ao invés de textualmente, utilizadas para transformar os símbolos iniciais e atingir diferentes formas.

Tendo sua definição como uma quádrupla : G = <V_t,V_m,R,I> onde:

- V_t é um conjunto finito de formas.
- V_m é um conjunto finito de formas de maneira que $V_t \cap V_m = \emptyset$.
- R é um conjunto finito de regras
- I é uma forma inicial e subconjunto de V_t e V_m

As formas contidas em V_t são geradas a partir do conjunto de produtores V_m de acordo com as regras R que serão aplicadas durante sua produção a partir da forma inicial I.

Uma regra é descrita como A $\rightarrow$ B, dividindo-se em lado esquerdo e lado direito.

Uma regra pode ser aplicada quando a sua parte esquerda (A), que no caso, é uma forma, é detectada em uma outra forma C.

Essa detecção pode ocorrer via translação, rotação, escalonamento, entre outros processos lineares.

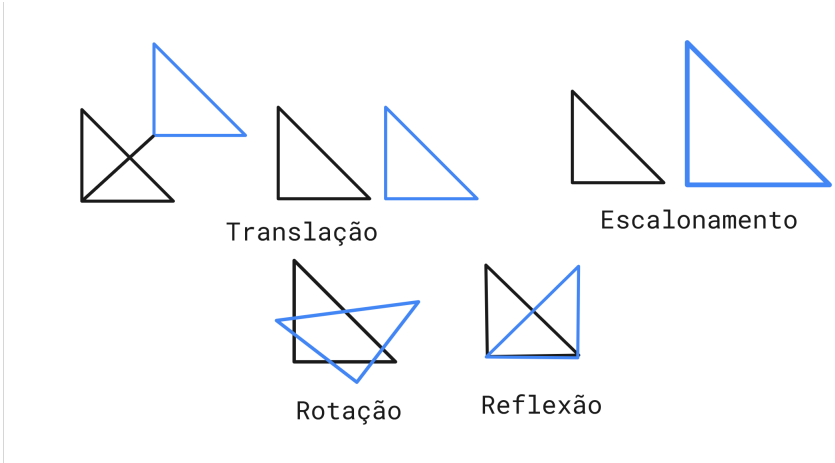

Figura 2 – Transformações lineares de formas 2D.

A forma (A) encontrada em C será então será substituída pela forma B, com suas transformações.

Exemplo de gramática de forma:

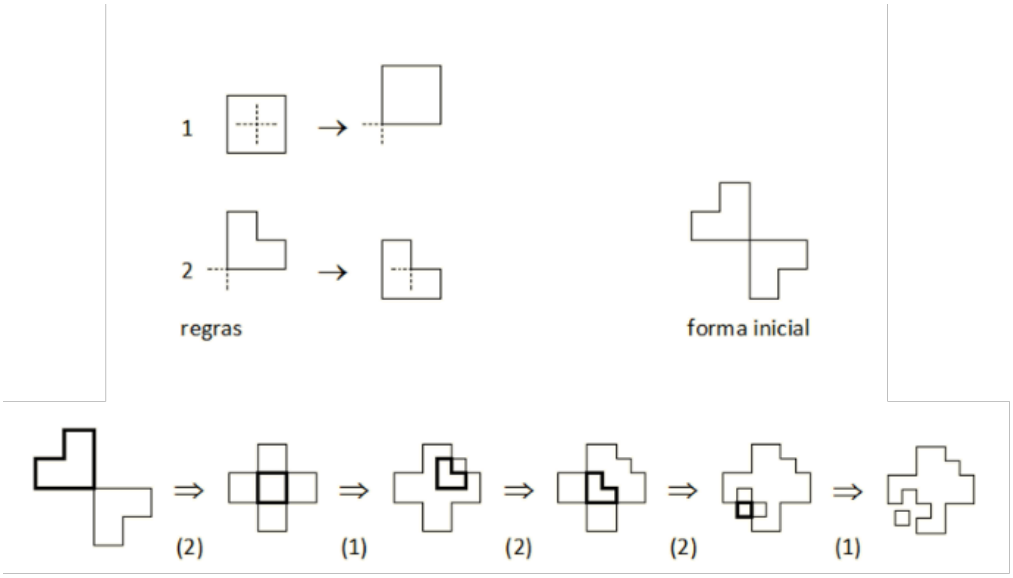

Figura 3 – Exemplo de aplicação das regras de uma gramática de formas. Retirada de (PINTO, 2010)

O processo de derivação, que substitui as formas de acordo com as regras aplicadas, pode resultar na exclusão ou inclusão de elementos novos, conforme a diferença do lado esquerdo e direito das regras. A Figura 4 mostra definições e exemplos dessas operações.

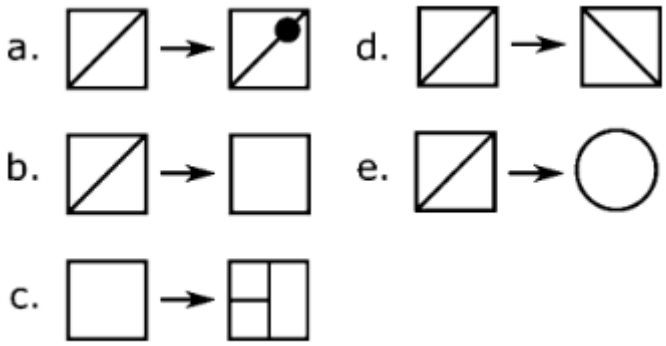

Figura 4 – Diferentes tipos de transformação das regras. (a) Adição, (b) Subtração, (c) Divisão, (d) Modificação e (e) Substituição. Fonte: (GU; BEHBAHANI, 2018)

2.3 TEORIA DA COMPUTAÇÃO E GRAMÁTICAS DE FORMAS

Na hierarquia de Chomsky, gramáticas generativas e máquinas de Turing são equivalentes, com mesmo poder computacional, sendo possível que o que seja computado em um seja computado em outro. Dito isso, é importante notar que o custo de recursos computacionais exigidos para cada modelo não é o mesmo, sendo necessária adaptações e entendimento das limitações de cada modelo. Segundo (PINTO, 2010) "O processo computacional para gramáticas de formas apresenta um desafio, o entendimento sobre

derivações de natureza não determinística e o possivelmente espaço de soluções infinito". O formalismo sobre as gramáticas de formas produziram extensos estudos sobre o assunto, mas a automatização desses processos é ainda explorada. Duas características centrais que distinguem as gramáticas de formas de outros modelos generativos são:

- Recursão: a recursão permite que um design seja descrito em termos de estados consecutivos, em que cada um deve ser construído para permitir a construção do próximo, um processo que requer um estado inicial e um conjunto de regras que geram os termos da sequência. Nesse sentido, um dos desafios computacionais se encontra na construção de um programa que suporte essa recursão de construções visuais.

- Incorporação: Numa computação simbólica, a combinação de símbolos discretos como letras ou números será sempre inequivocamente computada de uma forma não ambígua. Já em uma computação visual, a combinação de formas produzem um adicional muito complexo de formas diversas e emergentes, de diferentes topologias daquelas que a originaram. Com isso, um desafio também é encontrado na implementação de um programa que suporte computação espacial e a captura de continuidade visual da emergência de formas.

3 TRABALHOS RELACIONADOS

Gramáticas de formas no contexto arquitetural foram introduzidas como formas de analisar estilos arquitetônicos, complexos ou não, e identificar suas formas e composições espaciais, desde então são frequentemente propostas como mecanismos e ferramentas para o design na arquitetura. A implementação de um algoritmo de geração das produções de gramáticas de formas permite a produção de formas alternativas seguindo um padrão, definido pelas regras da gramática.

3.1 GRAMÁTICAS DE CASAS DE PRAIRIE

Prairie School foi um estilo arquitetônico do final do século XIX e começo do século XX, com projetos que são usualmente marcados por suas linhas horizontais, apresentando lajes terminando em beirais parcialmente sobrepostos. Uma gramática de forma baseada em sua composição (KONING; EIZENBERG, 1981) consegue gerar uma composição básica de uma típica casa desse estilo. Para esse desenvolvimento, o layout principal das casas é criado a partir de um ponto focal principal, a lareira, fazendo com que as regras sejam todas construídas como adições em volta deste ponto, com blocos adicionados construindo a composição base da casa.

Na figura 5 está representada a árvore de parte das possibilidades de geração dessa primeira etapa de construção, em cada ponto um design é escolhido para dar sequência a geração de novas formas.

Na primeira derivação, as regras 1 e 2 geram uma lareira de uma face ou duas. Na segunda derivação, as regras 3-6 geram o cômodo que envolve a lareira, definindo um ponto do layout. Na terceira derivação, a regra 7 é a única a ser aplicada e a que define a estrutura e forma da casa. Nas derivações seguintes são aplicados o que se foi chamado de extensões obrigatórias, ponto de definição do estilo arquitetônico.

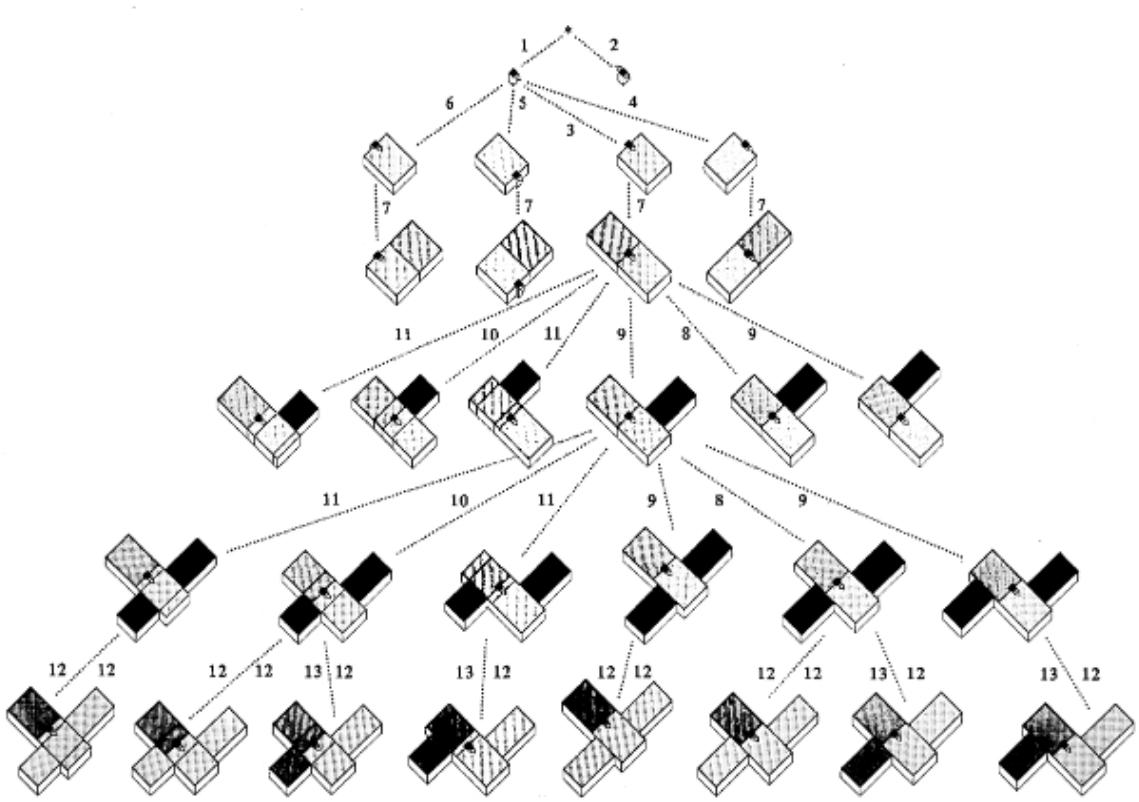

Figura 5 – Aplicação das regras da gramática de casas de Prairie. Retirado de (KONING; EIZENBERG, 1981)

Nas próximas etapas da construção, são adicionados tetos, varandas, e outros elementos mais complexos que compõem o design especificado.

3.2 GRAMÁTICAS DE CASAS QUEEN ANNE

O estilo arquitetônico Queen Anne foi um popular estilo vitoriano que emergiu nos Estados Unidos no período de 1880. Seus traços principais são as fachadas assimétricas, colunas clássicas, varandas e torres em formas redondas, quadradas ou poligonais no geral. Com a gramática elaborada por (FLEMMING, 1987), é possível exemplificar e gerar um exemplo básico de uma típica casa Queen Anne. Porém, a gramática não consegue gerar uma casa completa, em detalhes, devido a seus aspectos muito complexos que não conseguem ser aplicados a um par de regras, mas que podem ser manualmente inseridos por um designer ou arquiteto envolvido em um projeto. Um diferencial é que a primeira parte dessa gramática é bi-dimensional, enquanto a segunda parte é tri-dimensional.

Esse trabalho é um bom exemplo de uma ferramenta auxiliar no processo de criação e desenvolvimento, sem criar completamente todos os passos necessários, que envolve uma percepção humana e qualificada para terminar a projeção.

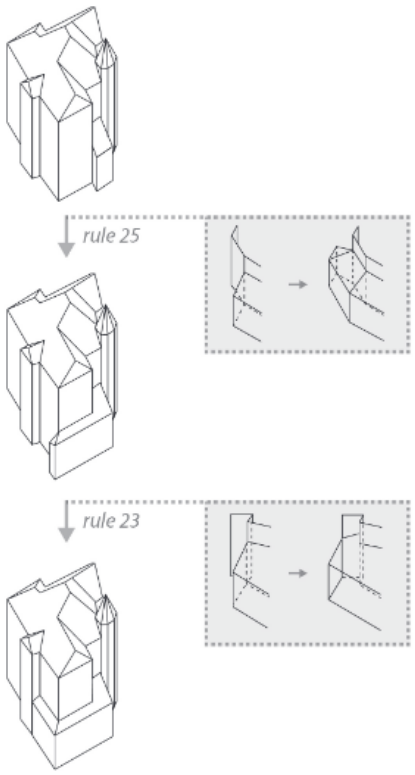

Figura 6 – Aplicação de regras da gramática que permitem a ampliação de paredes no exterior da casa. Retirado de (M. VERKERK, 2014)

3.3 GRAMÁTICAS DE CASAS MALAGUEIRA

A gramática de casas Malagueira têm seu design baseado nas construções de trinta e cinco casas criadas por Álvaro Siza em Malagueira, Portugal, entre 1977 e 1996. Essa gramática foi co-desenvolvida pelo próprio arquiteto português, sendo considerada uma extensão de seu trabalho, e em certo momento o mesmo já não conseguia propriamente diferenciar entre casas feitas pelo arquiteto ou geradas pela gramática. A gramática em si possui três estágios e é bi-dimensional, e possui muitos detalhes funcionais em seu estágio mais avançado, como a definição de cozinhas, lavanderias e outros espaços. O trabalho de Jose Duarte (DUARTE, 2005) foi muito significante, já que propôs um modelo capaz de gerar diversos designs de habitação em massa, sem repetições. Em suas conclusões, Duarte afirma a importância de um programa de computador que, dado uma gramática que pode ser sistematizada em forma de linguagem, faria a codificação da mesma e permitiria um uso ainda mais eficiente do design de Siza, principalmente na aceleração do processo de geração.

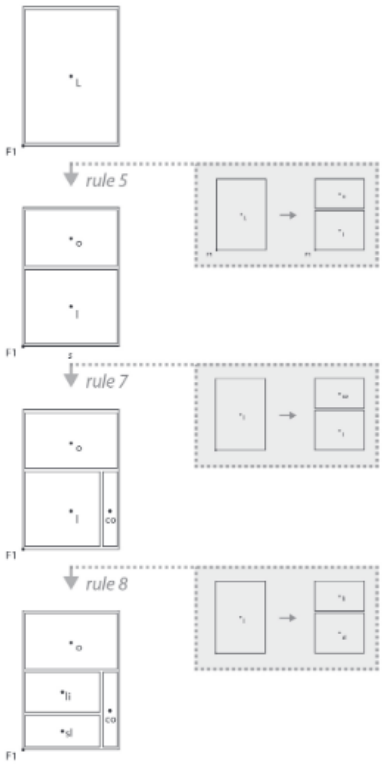

Figura 7 – Aplicação das regras sub-divisionais da gramática. Retirado de (M. VERKERK, 2014)

3.4 COMPARAÇÃO COM TRABALHOS RELACIONADOS

A Tabela 1 abaixo compara os trabalhos relacionados com o trabalho atual, levando em consideração os modelos de representação, se a maioria das regras referente as gramáticas são subdivisionais e se esses trabalhos implementam algum tipo de algoritmo gerador para a gramática discutida.

	Representação 3D	**Regras subdivisionais**	**Algoritmo gerador**
Casas Malagueira	Não	Sim	Não
Casas Queen Anne	Sim	Não	Não
Casas de Prairie	Sim	Não	Não
TCC	Não	Sim	Sim

Tabela 1 – Comparação dos trabalhos relacionados

4 DESENVOLVIMENTO

Esse capítulo irá abordar as etapas de desenvolvimento e construção do algoritmo para a geração das produções de uma gramática de formas, visando entender suas estruturas e componentes necessários.

4.1 REPRESENTAÇÃO

Essa seção descreve as estruturas utilizadas que representam as gramáticas de formas, de forma que ela possa ser representada em código como entrada do algoritmo. Para todos os desenvolvimentos práticos foi definido o uso da linguagem Python, por possuir bibliotecas de grande auxílio para este trabalho, juntamente com o modelo orientado a objetos com os seguintes modelos:

1. `Shape`: representação de uma forma

   Cada forma possui:

   `geometry ->` uma geometria representada como um *Polygon* da biblioteca *shapely*

   `type ->` identificação em *string* do tipo de cômodo que a forma representa.

   `adjacent_shape- >` lista de cômodos adjacentes do tipo *Shape*

2. `Rule`: representação de uma regra

   Essa é a classe pai de representação de uma regra da gramática, ou seja, para cada regra criada uma classe filha é implementada.

   Cada regra possui:

   `left_side ->` identificação em *string* do lado esquerda da regra, representando um cômodo gerador

   `right_side_function ->` função do lado direito da regra que define a produção e transformação da forma, retorna a lista de novas formas criadas.

   `condition - >` função *boolean* que representa se a condição para que a regra seja aplicada é verdadeira

   `detect_shape - >` função *boolean* que detecta a forma atual no lado esquerdo da regra

   `apply_shape - >` função comum que aplica o lado direito da regra

3. `Grammar`: representação de uma gramática

   Classe com todos os elementos de uma gramática: um conjunto de regras, e uma forma inicial.

   `rules ->` conjunto de regras

   `initial_shape ->` Objeto do tipo Shape representando a forma inicial

4.2 ELEMENTOS DA EXECUÇÃO

Para a execução do algoritmo, precisaremos de elementos e métodos auxiliares para a detecção e transformação da formas presentes na gramática. Esses métodos se encontram dentro da classe Rule e são comuns para todas as regras que são implementadas.

4.2.1 Detecção das formas produtoras

Processo que identifica o lado esquerdo de alguma regra na forma atual.
Entradas:

1. `shape`: Objeto da classe Shape que representa o cômodo que desejamos encontrar no lado esquerdo de alguma regra.

2. `current_shapes`: Conjunto de todos os objetos do tipo Shape que formam a forma resultante.

Algorithm 1 Código para a detecção do lado esquerdo da regra

if $shape.type == rule.left\_side$ and $rule.condition(shape, current\_shape)$ is true **then**
    return true
else
    return false
end if

Para a detecção de uma forma do lado esquerdo de uma regra, foi-se usada uma identificação direta do tipo do cômodo que o polígono representa, uma vez que a gramática se baseia em uma ideia de que as funcionalidades de cada cômodo podem gerar outros cômodos que tem uma ligação direta com a funcionalidade detectada, ou outros modos daquela funcionalidade ser representada e/ou dividida. Isso faz com que a geometria do lado esquerdo da regra seja dinâmica, ou seja, uma vez que um tipo de cômodo é identificado do lado esquerdo da regra, a regra assume aquela forma para que a regra aplicada seja em cima da forma sendo detectada.

Além disso, pensando na adaptação para os projetos arquitetônicos, para cada regra implementada é possível se definir uma condição para que a regra possa ser efetivamente detectada, já que o elemento de recursão nem sempre é desejado na adaptação destes modelos reais.

4.2.2 Aplicação da regra

Para a execução da aplicação do lado direito da regra, cada regra implementada possui sua própria função de transformação que é atribuída à *right_side_function*

O nome do método é *apply_shape* e as entradas são as mesmas do método *detect_shape*. A chamada do método apenas executa a função *right_side_function* estabelecida pela regra

4.2.3 Transformação da forma

A transformação da forma é o processo linear de subdivisão que ocorre em cima da geometria do polígono (pontos) que representa a forma. Para isso, foram criados métodos utilitários com todas as possibilidades de transformação de um polígono para as gramática criadas na seção 4.3. Os métodos construídos transformam o polígono de acordo com que o lado direito das regras das gramáticas requerem, o que envolve a divisão do polígono verticalmente ou horizontalmente em diferentes pontos, levando em conta que todos os polígonos que serão trabalhados são retangulares.

As bibliotecas auxiliares utilizadas para representação e manipulação dos polígonos foram:

- *shapely.geometry*: Criação de linhas e polígono com as classes *LineString* e *Polygon*, respectivamente
- *shapely.ops*: Operações e manipulações em cima de uma geometria, como a divisão com o método *split*.

Entrada: a geometria da forma a ser transformada, representada por *geometry*.
Saída: Lista com os novos polígonos gerados a partir da divisão.

1. Dividindo metade do polígono verticalmente: Primeiramente, são listados os pontos máximos e mínimos de x e y com o atributo da classe Polygon chamado *bounds*. Para o cálculo da linha de divisão, é preciso encontrar o ponto médio ao longo do eixo x, que é feito dividindo por 2 a soma do ponto máximo e mínimo deste. Com o cálculo do ponto médio, é possível criar a linha de divisão que irá cortar o polígono.

Algorithm 2 Código para a divisão vertical do polígono

$min\_x, min\_y, max\_x, max\_x \leftarrow geometry.bounds$
$mid\_x \leftarrow (min\_x + max\_x)/2$
$split\_line \leftarrow LineString([(mid\_x, min\_y), (mid\_x, max\_y)])$
return $split(geometry, splitline)$

2. Dividindo metade do polígono horizontalmente: A lógica é a mesma descrita para a divisão vertical, mas utilizando o eixo y para a criação da linha.

Algorithm 3 Código para a divisão horizontal do polígono

$min\_x, min\_y, max\_x, max\_x \leftarrow geometry.bounds$
$mid\_y \leftarrow (min\_y + max\_y)/2$
$split\_line \leftarrow LineString([(min\_x, mid\_y), (max\_x, mid\_y)])$
return $split(geometry, splitline)$

Para os métodos em que o polígono é dividido a 3/4, 1/4 ou 1/8 da forma, a largura ou altura é calculada e então dividida pelo desejado para que a linha de divisão seja criada neste ponto proporcional.

3. Dividindo um quarto do polígono verticalmente:

   Nesse ponto, a lógica de divisão do polígono é aplicada igualmente, e o cálculo da linha é criado com base em 1/4 da largura. Isso cria uma divisão de 1/4 do polígono, podendo ser mais a *esquerda* ou mais a *direita*.

Algorithm 4 Código para a divisão vertical do polígono mais a esquerda

$min\_x, min\_y, max\_x, max\_x \leftarrow geometry.bounds$
$quarter\_x \leftarrow (max\_x - min\_x)/4$
$split\_line \leftarrow LineString([(min\_x + quarter\_x, min\_y), (min\_x + quarter\_x, max\_y)])$
return $split(geometry, splitline)$

Algorithm 5 Código para a divisão vertical do polígono mais a direita

$min\_x, min\_y, max\_x, max\_x \leftarrow geometry.bounds$
$quarter\_x \leftarrow (max\_x - min\_x)/4$
$split\_line \leftarrow LineString([(max\_x - quarter\_x, min\_y), (max\_x - quarter\_x, max\_y)])$
return $split(geometry, splitline)$

4. Dividindo um quarto do polígono horizontalmente

   A mesma lógica para a divisão vertical é aplicada, mas calculando agora a altura e criando a possibilidade de um polígono ser dividido mais *abaixo* ou mais *acima*

5. Adicionando transformações de "canto"

   Para aquelas regras que fazem uma divisão de "canto", (regras 17-19 da figura 9), um quadrado proporcional a 1/4 ou 1/8 da área é criado em um canto desejado e depois este espaço criado é subtraído da forma original. Ou seja, tanto o polígono criado quando a forma modificada são adicionados como os novos polígonos resultantes dessa transformação.

Algorithm 6 Código para a criação de uma divisão de canto

$polygon\_area \leftarrow geometry.area$
$small\_square\_area \leftarrow polygon\_area/4$
$bottom\_left\_point \leftarrow get\_bottom\_left\_point(geometry)$
$small\_square \leftarrow create\_square\_with\_area(leftmost\_point, area)$
$transformed\_shape \leftarrow geometry.difference(small\_square)$
return $[transformed\_shape, small\_square]$

Os métodos *get_bottom_left_point* e *create_square_with_area* descobrem o ponto mais embaixo a esquerda e criam o quadrado com a área desejada, respectivamente.

4.2.4 Exibição da forma resultante

Para exibir a forma resultante, a biblioteca usada é a *matplotlib*, capaz de demonstrar os polígonos resultantes depois de todas as transformações. Os métodos usados da biblioteca foram *plot* para mostrar o polígono, *fill* para prrenchimento das cores e *text* para escrever o cômodo que ele representa.

4.2.5 Geração de formas

Com a definição dos métodos e elementos auxiliares definidos, podemos elaborar e criar o algoritmo que executa as derivações de uma gramática de formas:

1. Entrada: `shape_grammar -> Objeto do tipo Grammar`
2. Variável: `current_shape -> Lista que representa a forma atual, contém todos os polígonos que a formam. Inicialmente vazia`
3. Variável: `possible_rules -> Lista auxiliar para adicionar as possíveis regras detectadas`
4. Método: `pick_rule(possible_rules) -> Método auxiliar que escolhe aleatoriamente uma regra da lista`

Algorithm 7 Geração de formas

```
Entrada: Gramática de formas representada por shape_grammar
current_shape.append(shape_grammar.initial_shape)
has_possible_rules ← true
while (has_possible_rules is true:) do
    for shapes in current_shape do
        for rule in shape_grammar.rules do
            if rule.detect_shape(shape, current_shape) is true: then
                possible_rules.append(rule)
            end if
        end for
        if possible_rules.size() > 0 then
            picked_rule = pick_rule(possible_rules)
            new_shapes ← picked_rule.apply_rule(current_shape, shape)
            current_shape.append(new_shapes)
        else
            has_possible_rules ← false
        end if
    end for
end while
plot_shape(current_shape)
```

Dada a gramática de formas de entrada, a variável *current_shape* é atualizada com a forma inicial e a *has_possible_rules* recebe true para que o algoritmo comece. A forma atual é representada por polígonos menores com suas respectivas geometrias e a primeira iteração percorre por todos eles, adicionando regras possíveis a uma lista. Para garantir mais diversidade de produções, a regra é escolhida aleatoriamente e então seu lado direito é executado, garantindo a transformação dos polígonos e a geração de novas formas. A forma atual recebe suas novas formas transformadas, montando a nova forma atual para o próximo ciclo. Caso não hajam mais regras a serem aplicadas, o programa termina e a forma resultante é exibida.

4.3 PREPARAÇÃO

A preparação para a execução requer a criação de gramáticas de formas que serão a entrada do algoritmo desenvolvido, representada nas estruturas necessárias. Essa preparação pode ser dividida em:

1. Definição da forma inicial
2. Definição das regras da gramática

4.3.1 Gramática de projeto arquitetônico: Casa

Como uma das entradas do algoritmo, foi criada uma gramática de formas que gera projetos arquitetônicos de uma casa simples de até 2 quartos. Para isso, foram buscados

exemplos de plantas e projetos de casas reais, assim como exemplos de alguns dos trabalhos relacionados, como a gramática de casas Malagueira. Essa gramática é baseada em regras sub-divisionais e tem como símbolo inicial um espaço retangular que é a representação do espaço em que a casa será planejada com a criação de um projeto arquitetônico. As regras foram criadas levando em consideração as necessidades para uma casa funcional que faça sentido no nosso mundo real.

Figura 8 – Forma inicial da gramática de formas de uma casa

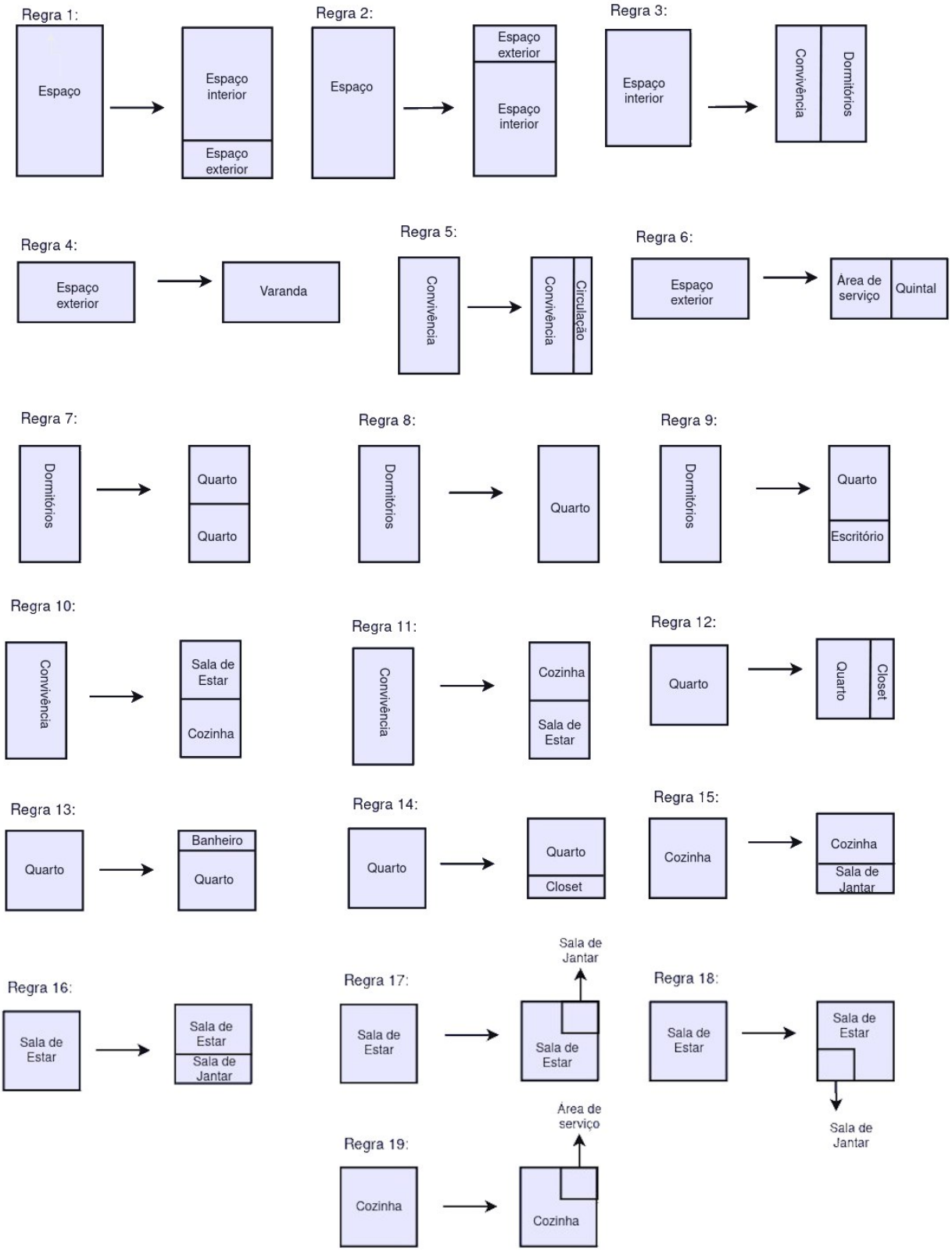

Figura 9 – Regras da gramática de formas para o projeto arquitetônico de uma casa

As regras 1 e 2 transformam a forma inicial com uma divisão de espaço interior e

exterior, criando 2 novos cômodos.

A regra 3 transforma o espaço interior com uma divisão de convivência e dormitórios. O cômodo de convivência gera os espaços comuns em uma casa, como a cozinha, salas de estar e jantar (regras 10 e 11), enquanto o espaço de dormitórios gera 1 ou 2 quartos, assim como a possibilidade de gerar um quarto e um escritório (regras 7, 8 e 9).

A regra 5 adiciona um espaço de circulação adjacente à área de convivência.

As regras 4 e 6 transformam o `"Espaço exterior"`, sendo possível a criação de 2 novos cômodos, como a área de serviço e o `"Quintal"`, ou a transformação do `"Espaço exterior"` inteiramente em uma varanda

As regras 12, 13 e 14 adicionam elementos que podem ser criados a partir de um quarto, como um banheiro ou um *closet*

Uma sala de jantar pode ser adicionada a partir de uma cozinha (regra 15) ou em diferentes posições a partir de uma sala de estar (regras 16, 17 e 18)

Adicionalmente, uma área de serviço pode ser adicionada a partir de uma cozinha (regra 19).

4.3.2 Classificação da gramática criada

Para verificar o nível de computabilidade da gramática, é possível classificá-la como um dos 4 tipos definidos na hierarquia de Chomsky. Observando a figura 9 é possível perceber que as regras são da forma N $\rightarrow (N \cup T)^+$, ou seja, o lado esquerdo da gramática sempre é representado por um único cômodo não-terminal que gera novos cômodos terminais ou não-terminais. Por exemplo, o cômodo não-terminal "Quarto"pode gerar um banheiro (terminal) ou um closet (terminal), ambos acompanhados pelo símbolo Quarto (não terminal). Dado as condições da gramática, um quarto também pode virar um símbolo terminal assim que um banheiro ou closet não possa mais ser gerado. Essa regra específica pode ser representada como:

$$\text{Quarto} \rightarrow \text{Quarto(banheiro)} \mid \text{Quarto(closet)} \mid \text{(quarto)}$$

Portanto, a gramática pode ser considerada como livre de contexto e pode ser reconhecida por um Autômato de Pilha.

4.3.3 Representação em código

Essa seção desenvolve a construção da gramática de formas exibida na seção 4.3.1 com os modelos de representação mostrados na seção 4.1

- Representação da forma inicial

  ```
  initial_shape -> Shape(Polygon([(0, 0), (0, 15), (10, 15), (10, 0)]),
  "Espaço", [])
  ```

O polígono inicial é inicializado com dimensões de 10m x 15m, gerando 150m² de área.

O tipo do cômodo inicial é "Espaço", indicando a forma inicial.

A lista formas adjacentes é inicializada vazia.

- Representação das regras

  Cada regra irá herdar os métodos da classe pai *Rule*, e deverá criar seus atributos específicos de lado esquerdo, direito e condição. Quando uma regra tem a condição como *true*, quer dizer que ela sempre pode ser aplicada. As funções de transformação e divisão estão de acordo com a seção 4.2.3.

  – **Regra 1**:

    `left_side ->` "Espaço"

    `condition -> true`

    `right_side_function ->` Função que divide o polígono em 1/8 no ponto mais acima e retorna os cômodos gerados: `"Espaço interior"` e `""Espaço exterior""`

  – **Regra 2**:

    `left_side ->"Espaço"`

    `condition -> true`

    `right_side ->` Função que divide o polígono em 1/8 no ponto mais abaixo e retorna os cômodos gerados: `"Espaço interior"` e `""Espaço exterior""`

  – **Regra 3**:

    `left_side -> "Espaço interior"`

    `condition -> true`

    `right_side ->` Função que divide o polígono verticalmente, gerando os cômodos: `"Dormitórios"` e `"Convivência"`.

  – **Regra 4**:

    `left_side -> "Espaço exterior"`

    `condition -> true`

    `right_side ->` Função que transforma um `"Espaço exterior"` em um cômodo `"Varanda"`, mantendo sua geometria original.

– **Regra 5**:

`left_side ->` `"Convivência"`

`condition ->` Se já existe um cômodo `"Circulação"` na lista de formas adjacentes, retorna *false*, se não, *true*

`right_side ->` Função que divide o polígono 1/4 mais a direita, gerando os cômodos `"Convivência"` e `"Circulação"`.

– **Regra 6**:

`left_side ->` `""Espaço exterior""`

`condition ->` `true`

`right_side ->` Função que divide o polígono verticalmente, retornando dois novos cômodos: `"Área de serviço"` e `"Quintal"`.

– **Regra 7**:

`left_side ->` `"Dormitórios"`

`condition ->` `true`

`right_side ->` Função que divide o polígono horizontalmente, retornando dois novos cômodos: `"Quarto"` e `"Quarto"`.

– **Regra 8**:

`left_side ->` `"Dormitórios"`

`condition ->` `true`

`right_side ->` Função que transforma um `"Dormitórios"` em um cômodo `"Quarto"`, mantendo sua geometria original.

– **Regra 9**:

`left_side ->` `""Dormitórios""`

`condition ->` `true`

`right_side ->` Função que divide o polígono em 3/4 mais abaixo, retornando dois novos cômodos: `"Quarto"` e "Escritório".

– **Regra 10**:

`left_side ->` `"Convivência"`

`condition ->` `true`

`right_side ->` Função que divide o polígono horizontalmente, gerando os cômodos `"Sala de Estar"` e `"Cozinha"`.

– **Regra 11**: Igual a regra 10, com a ordem de `"Sala de Estar"` e `"Cozinha"` invertidas

– **Regra 12**:

`left_side ->` `"Quarto"`

`condition ->` Se já existe um cômodo `"Closet"` na lista de formas adjacentes, retorna *false*, se não, *true*

`right_side ->` Função que divide o polígono 1/4 mais a direita, gerando os cômodos `"Quarto"` e `"Closet"`.

– **Regra 13**:

`left_side ->` `"Quarto"`

`condition ->` Se já existe um cômodo `"Banheiro"` na lista de formas adjacentes, retorna *false*, se não, *true*

`right_side ->` Função que divide o polígono 1/4 mais acima, gerando os cômodos `"Quarto"` e `"Banheiro"`.

– **Regra 14**:

`left_side ->` `"Quarto"`

`condition ->` Se já existe um cômodo `"Closet"` na lista de formas adjacentes, retorna *false*, se não, *true*

`right_side ->` Função que divide o polígono 1/4 mais abaixo, gerando os cômodos `"Quarto"` e `"Closet"`.

– **Regra 15**:

`left_side ->` `"Cozinha"`

`condition ->` Se já existe um cômodo "Sala de Jantar"na lista de formas atuais, retorna *false*, se não, *true*

`right_side ->` Função que divide o polígono 1/4 mais abaixo, gerando os cômodos `"Cozinha"` e `"Sala de Jantar"`

– **Regra 16**:

`left_side ->` `"Sala de Estar"`

`condition ->` Se já existe um cômodo "Sala de Jantar"na lista de formas atuais, retorna *false*, se não, *true*

`right_side ->` Função que divide o polígono 1/4 mais abaixo, gerando os cômodos `"Sala de Estar"` e `"Sala de Jantar"`

– **Regra 17**:

`left_side -> "Sala de Estar"`

`condition ->` Se já existe um cômodo "Sala de Jantar"na lista de formas atuais, retorna *false*, se não, *true*

`right_side ->` Função que divide o polígono com um canto mais a direita, gerando os cômodos `"Sala de Estar"` e `"Sala de Jantar"`

– **Regra 18**:

`left_side -> "Sala de Estar"`

`condition ->` Se já existe um cômodo "Sala de Jantar"na lista de formas atuais, retorna *false*, se não, *true*

`right_side ->` Função que divide o polígono com um canto mais a esquerda, gerando os cômodos `"Sala de Estar"` e `"Sala de Jantar"`

– **Regra 19**:

`left_side -> "Cozinha"`

`condition ->` Se já existe um cômodo `"Área de serviço"` na lista de formas atuais, retorna *false*, se não, *true*

`right_side ->` Função que divide o polígono com um canto mais a direita, gerando os cômodos `"Cozinha"` e `"Área de serviço"`.

4.3.4 Gramática de projeto arquitetônico: Escritório

Assim como a gramática anterior, essa gramática com regras sub-divisionais foi criada buscando exemplos reais de escritórios pequenos de até duas salas.

Figura 10 – Forma inicial da gramática de formas de um escritório

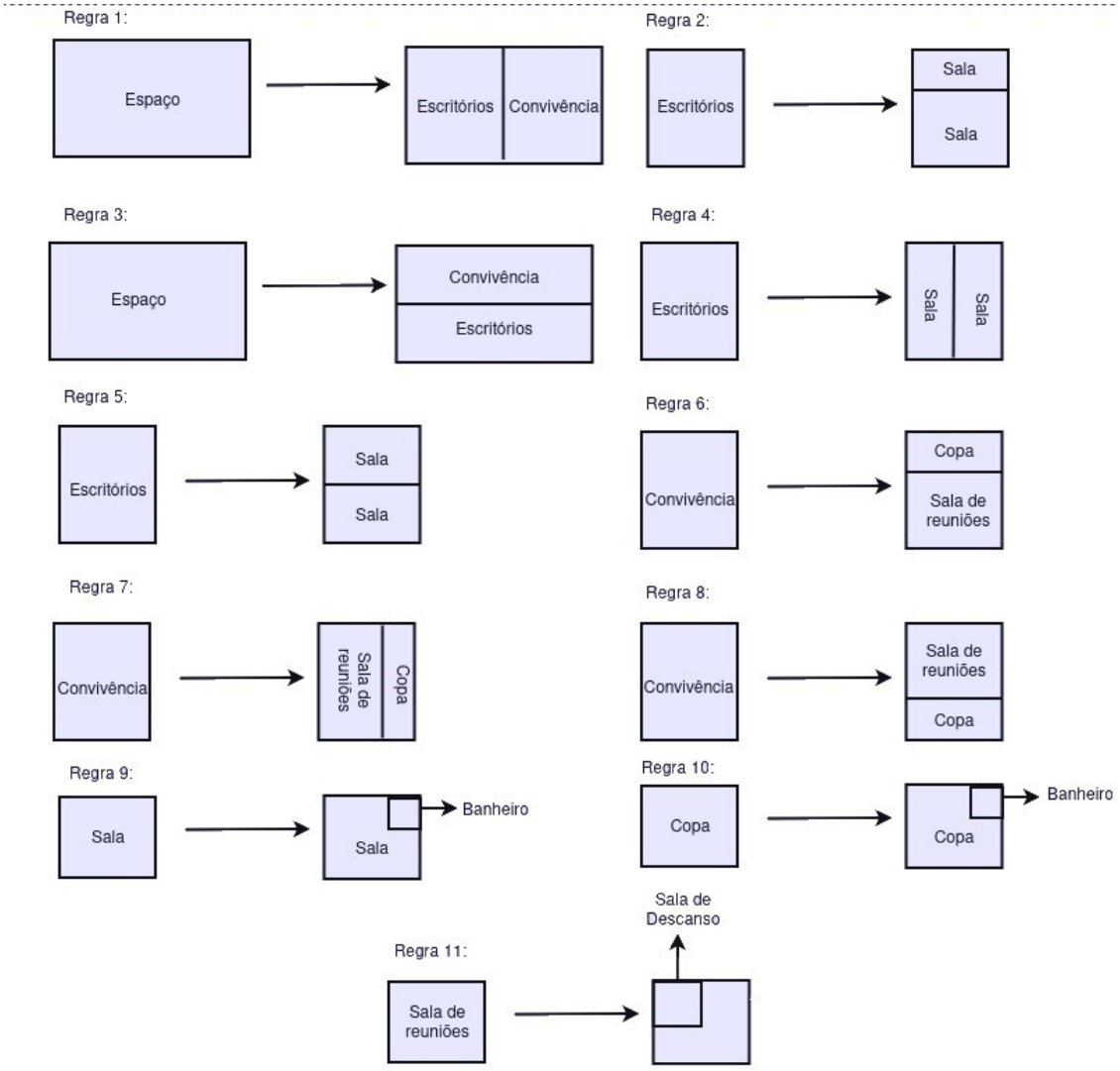

Figura 11 – Regras da gramática de formas para o projeto arquitetônico de um escritorio

4.3.5 Representação em código

Essa seção desenvolve a construção da gramática de formas exibida na seção 4.3.4 com os modelos de representação mostrados na seção 4.1

- Representação da forma inicial

  `initial_shape -> Shape(Polygon([(0, 0), (0, 15), (10, 15), (10, 0)]), "Espaço", [])`

  O polígono inicial é inicializado com dimensões de 10m x 15m, gerando 150m² de área.

  O tipo do cômodo inicial é "Espaço", indicando a forma inicial.

  A lista formas adjacentes é inicializada vazia.

- Representação das regras

As regras são criadas com as mesmas estruturas da gramática anterior, continuando a se basear em regras sub-divisionais.

- **Regra 1**:

  `left_side ->` "Espaço"

  `condition -> true`

  `right_side_function ->` Função que divide o polígono verticalmente e retorna os cômodos gerados: "Escritórios"e `"Convivência"`

- **Regra 2**:

  `left_side ->`"Escritórios"

  `condition -> true`

  `right_side ->` Função que divide o polígono em 1/4 no ponto mais acima e retorna os cômodos gerados: `"Sala"` e `"Sala"`

- **Regra 3**:

  `left_side ->` "Espaço"

  `condition -> true`

  `right_side ->` Função que divide o polígono horizontalmente, gerando os cômodos: `"Dormitórios"` e `"Convivência"`.

- **Regra 4**:

  `left_side ->` "Escritório"

  `condition -> true`

  `right_side ->` Função que divide o polígono verticalmente e retorna os cômodos gerados: `"Sala"` e `"Sala"`

- **Regra 5**:

  `left_side ->` "Escritório"

  `condition -> true`

  `right_side ->` Função que divide o polígonohorizontalmente e retorna os cômodos gerados: `"Sala"` e `"Sala"`

- **Regra 6**:

  `left_side -> "Convivência"`

  `condition -> true`

`right_side ->` Função que divide o polígono 1/4 mais acima, retornando dois novos cômodos: `"Copa"` e `"Sala de Reuniões"`.

- **Regra 7**:

  `left_side -> "Convivência"`

  `condition -> true`

  `right_side ->` Função que divide o polígono 1/4 mais a direita, retornando dois novos cômodos: `"Copa"` e `"Sala de reuniões"`.

- **Regra 8**:

  `left_side -> "Convivência"`

  `condition -> true`

  `right_side ->` Função que divide o polígono 1/4 mais abaixo, retornando dois novos cômodos: `"Copa"` e `"Sala de reuniões"`.

- **Regra 9**:

  `left_side -> "Sala"`

  `condition ->` Se já existe um cômodo `"Banheiro"` na lista de formas adjacentes ou se a quantidade de banheiros nas formas atuais é igual a 2, retorna *false*, se não, *true*

  `right_side ->` Função que divide o polígono com um canto mais a direita, gerando os cômodos `"Sala"` e `"Banheiro"`.

- **Regra 10**:

  `left_side -> "Copa"`

  `condition ->` Se já existe um cômodo `"Banheiro"` na lista de formas adjacentes, retorna *false*, se não, *true*

  `right_side ->` Função que divide o polígono com um canto mais a esquerda, gerando os cômodos `"Copa"` e `"Banheiro"`.

- **Regra 11**: `left_side -> "Sala de Reuniões"`

  `condition ->` Se já existe um cômodo `"Sala de descanso"` na lista de formas adjacentes, retorna *false*, se não, *true*

  `right_side ->` Função que divide o polígono com um canto mais a esquerda, gerando os cômodos `"Sala de Reuniões"` e `"Sala de descanso"`.

4.4 RESULTADOS

Esse seção é dedicada para os resultados obtidos do algoritmo de geração de formas. Será demonstrada cada etapa das produções da gramática e quais foram as regras aplicadas, assim como a forma resultante.

4.4.1 Experimento 1: Primeira derivação do projeto arquitetônico de uma casa

Ao inicializar a gramática com as regras e a forma inicial de acordo com a seção 4.3.5, o algoritmo irá começar a detectar as regras e aplicar suas transformações.

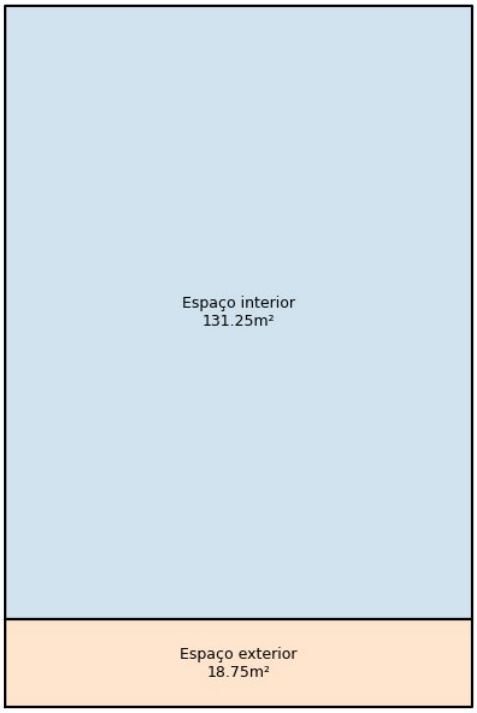

Figura 12 – Experimento 1: Detecção e aplicação da regra 1

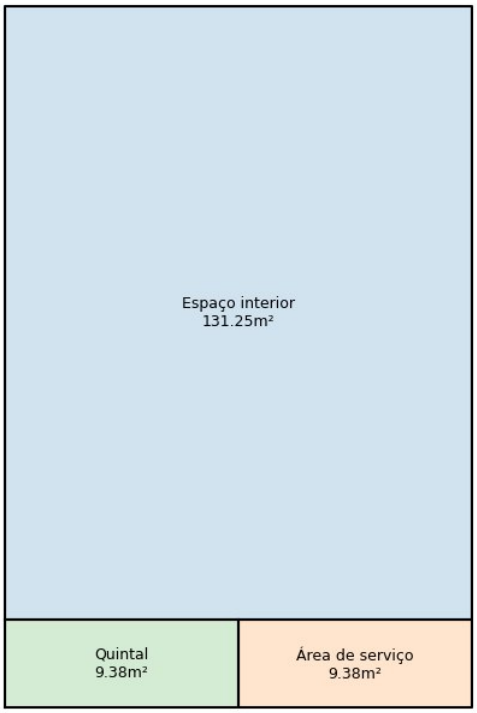

Figura 13 – Experimento 1: Detecção e aplicação da regra 6

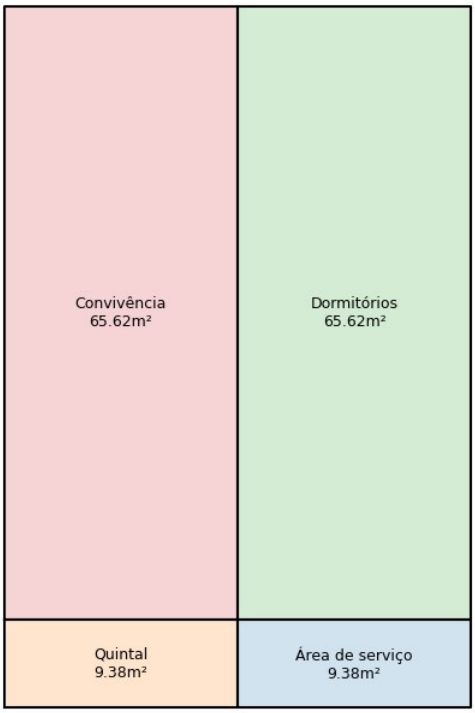

Figura 14 – Experimento 1: Detecção e aplicação da regra 3

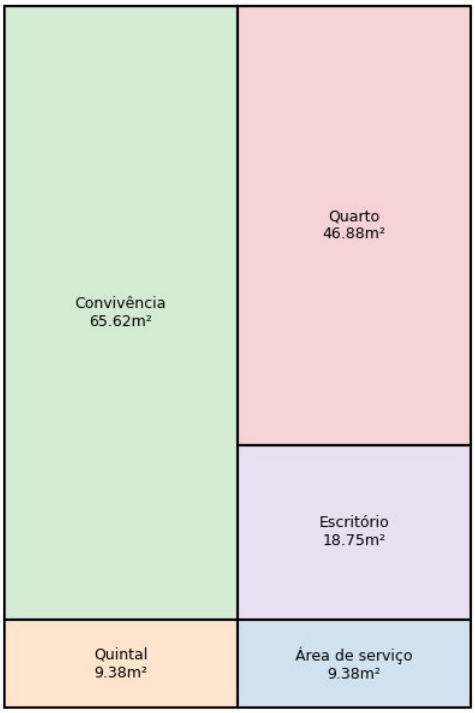

Figura 15 – Experimento 1: Detecção e aplicação da regra 9

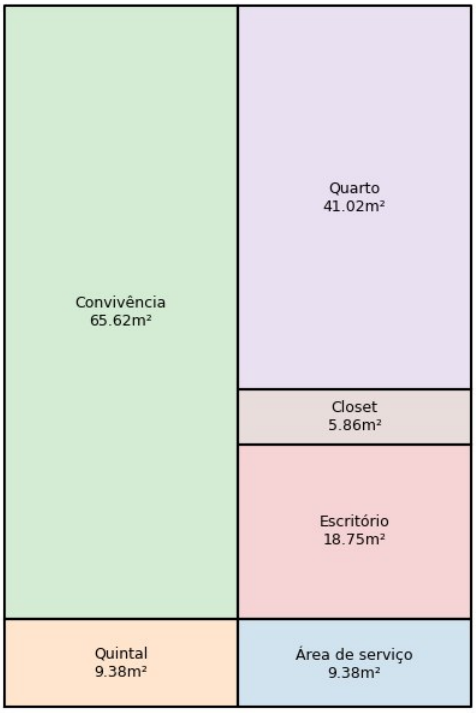

Figura 16 – Experimento 1: Detecção e aplicação da regra 14

Figura 17 – Experimento 1: Detecção e aplicação da regra 11

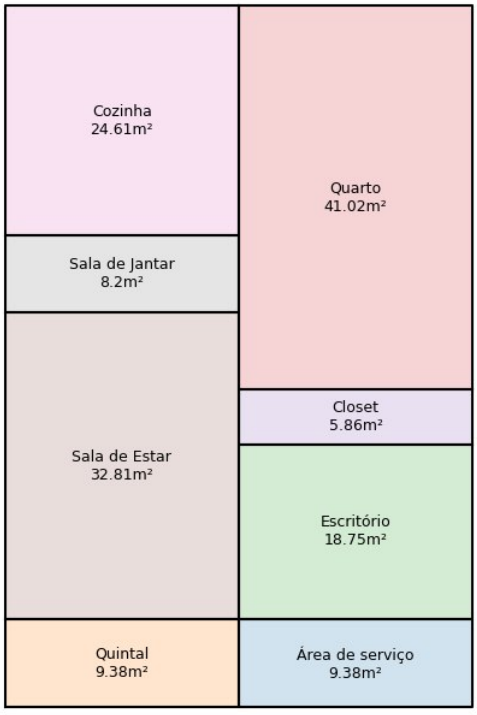

Figura 18 – Experimento 1: Detecção e aplicação da regra 15

Figura 19 – Experimento 1: Detecção e aplicação da regra 13

A figura 19 representa a forma resultante final da execução do algoritmo, já que mais nenhuma regra foi detectada. A execução detectou e aplicou 8 regras dentre as 19, e gerou 9 cômodos diferentes. As regras aplicadas geraram uma casa de 1 quarto e 1 escritório, com uma sala de jantar entre a cozinha e a sala de estar e uma área de serviço externa.

4.4.2 Experimento 2: Segunda derivação do projeto arquitetônico de uma casa

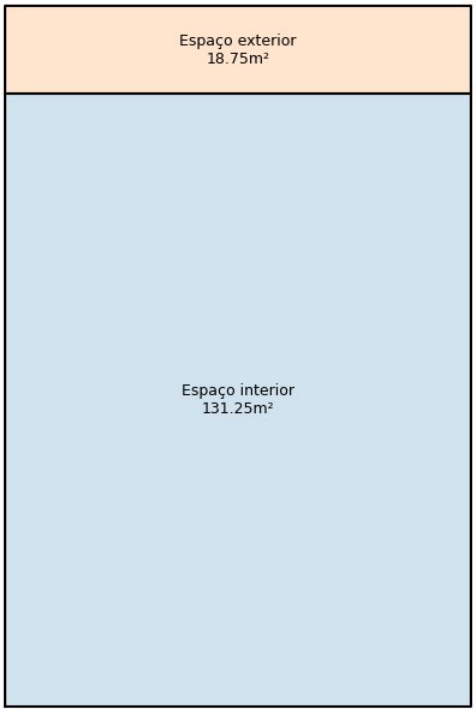

Figura 20 – Experimento 2: Detecção e aplicação da regra 2

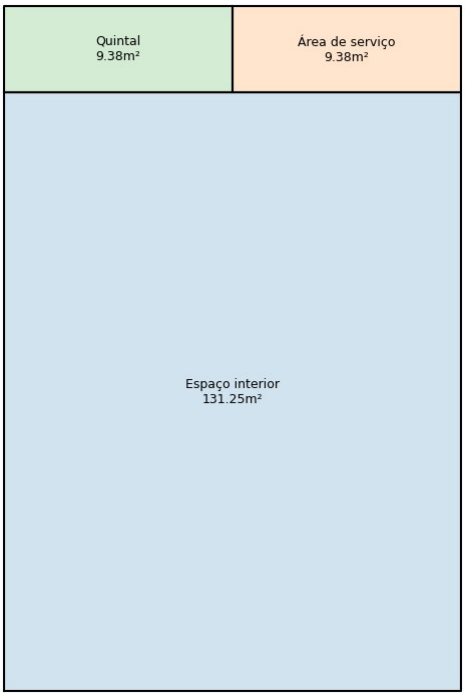

Figura 21 – Experimento 2: Detecção e aplicação da regra 6

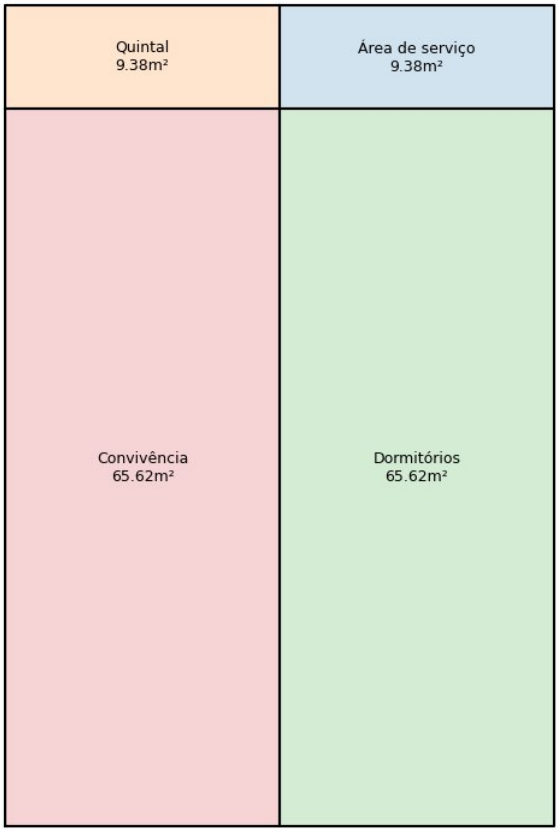

Figura 22 – Experimento 2: Detecção e aplicação da regra 3

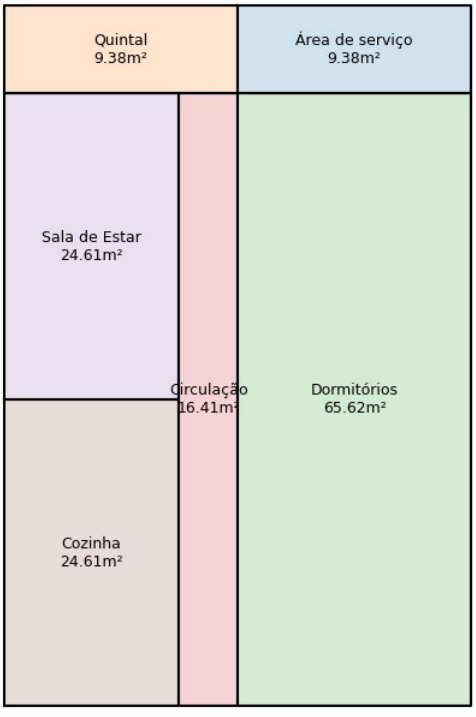

Figura 23 – Experimento 2: Detecção e aplicação da regra 10

Figura 24 – Experimento 2: Detecção e aplicação da regra 7

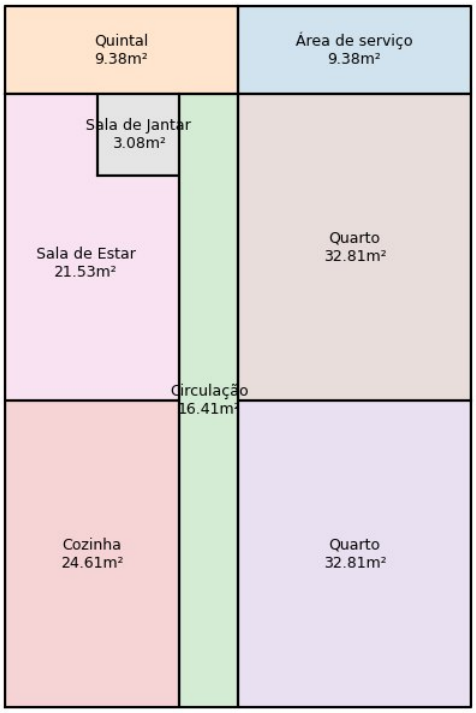

Figura 25 – Experimento 2: Detecção e aplicação da regra 17

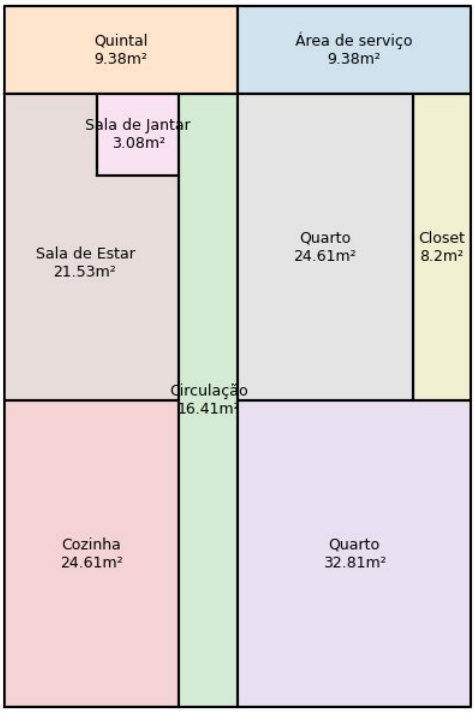

Figura 26 – Experimento 2: Detecção e aplicação da regra 12

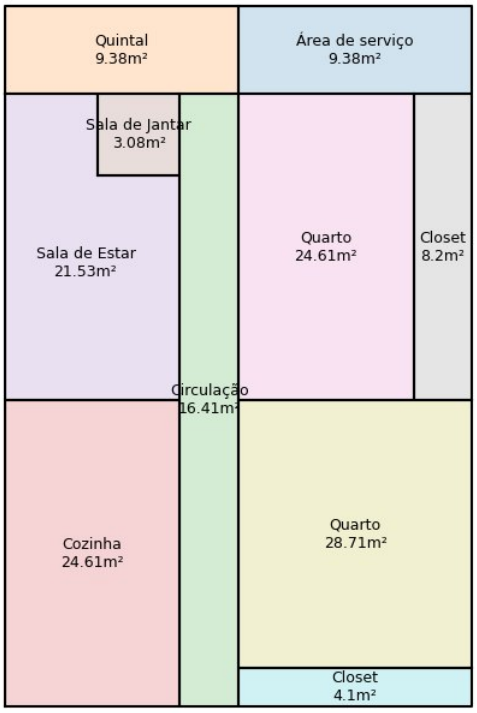

Figura 27 – Experimento 2: Detecção e aplicação da regra 14

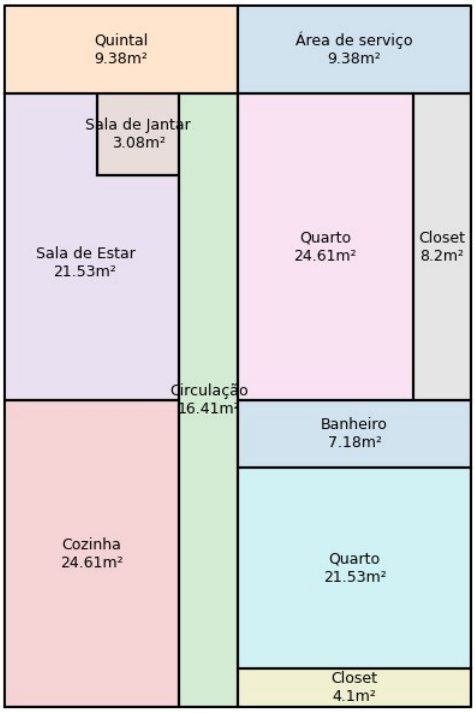

Figura 28 – Experimento 2: Detecção e aplicação da regra 13

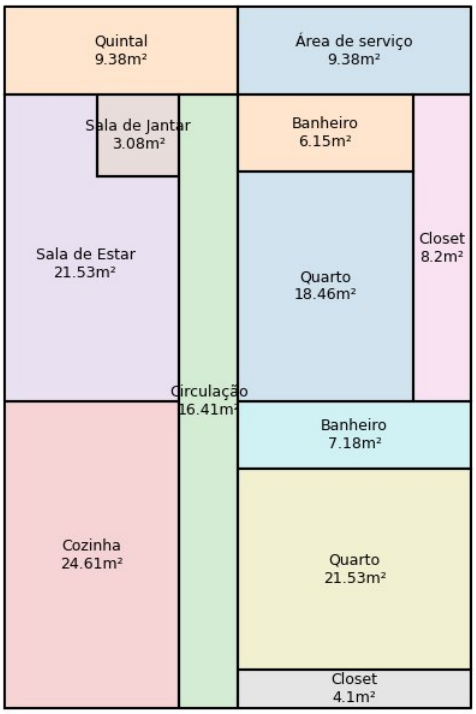

Figura 29 – Experimento 2: Detecção e aplicação da regra 13

A figura 29 é a forma resultante da execução. Diferentemente do experimento 1, essa execução detectou e aplicou 10 regras e gerou 12 cômodos. As regras aplicadas geraram

2 quartos, um espaço central de circulação e uma sala de jantar adicionada no canto da sala de estar, ao invés de entre um cômodo e outro.

4.4.3 Experimento 3: Derivação do projeto arquitetônico de um escritório

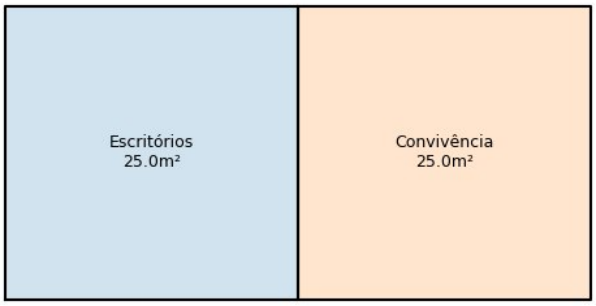

Figura 30 – Experimento 3: Detecção e aplicação da regra 1

Figura 31 – Experimento 3: Detecção e aplicação da regra 8

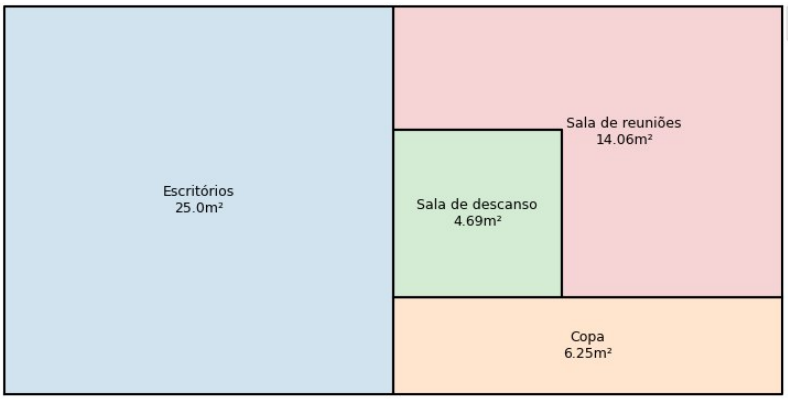

Figura 32 – Experimento 3: Detecção e aplicação da regra 11

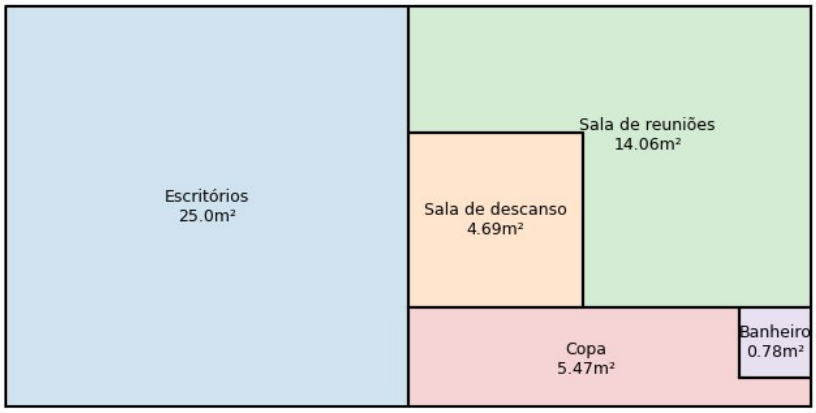

Figura 33 – Experimento 3: Detecção e aplicação da regra 10

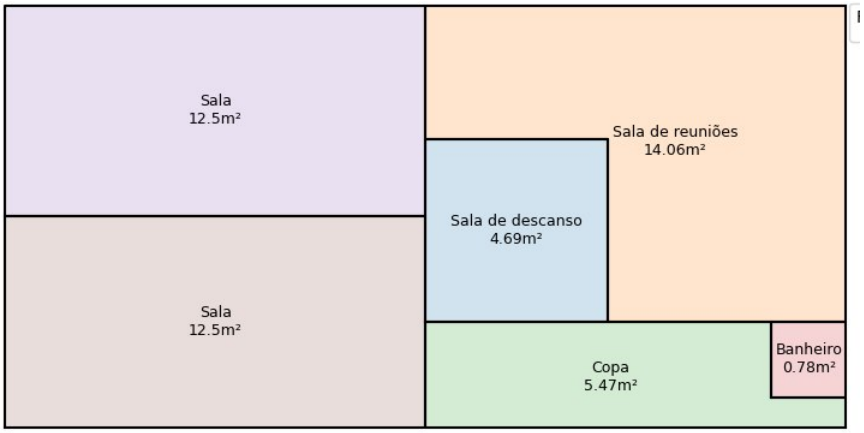

Figura 34 – Experimento 3: Detecção e aplicação da regra 5

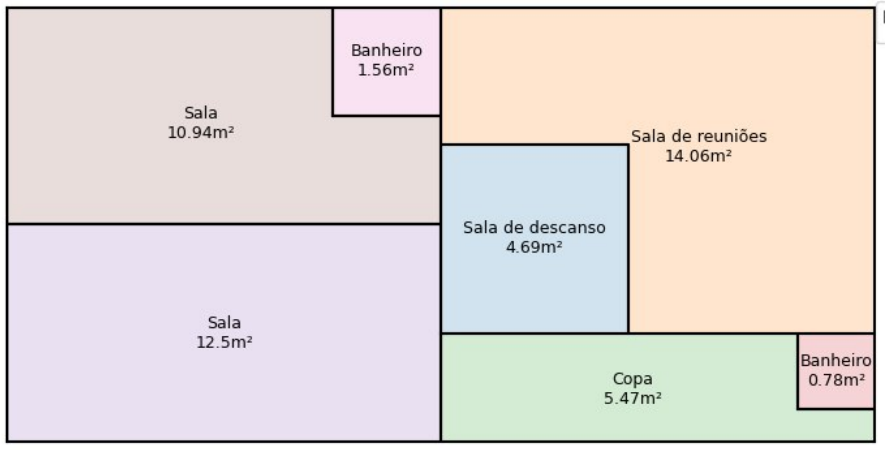

Figura 35 – Experimento 3: Detecção e aplicação da regra 9

A figura 35 representa a forma resultante da produção de um escritório.

4.5 ANÁLISE DE DESEMPENHO E COMPARATIVA

Após a exibição da forma resultante, é possível comparar o projeto gerado com plantas de construções reais, a fim de analisar se o resultado se assemelha a realidade e retrata um projeto arquitetônico funcional, com cômodos bem estabelecidos.

Figura 36 – Planta real de uma casa simples de 3 quartos. Retirado de: (TOSI, 2019)

Figura 37 – Planta real de uma casa simples de 2 quartos. Retirado de: (TOSI, 2019)

Figura 38 – Planta real de um escritório de advocacia. Retirado de: (EXPLORE. . . , s.d.)

Ao comparar a planta com a figura 29 resultante, é possível perceber que a produção da gramática conseguiu reproduzir requisitos posicionais de projetos arquitetônicos, como a divisão dos espaços de convivência que são geralmente adjacentes (Sala de Estar, Cozinha,

Sala de Jantar) e os quartos que são adjacentes a banheiros. Além disso, os espaços externos e de serviço também foram contemplados e posicionados de forma que façam sentido com o mundo real.

Já para a planta de escritório, a gramática contempla os espaços principais que compõem a planta, como um espaço de reuniões, espaços de trabalho, banheiros e convivência (copa).

Em relação ao desempenho do algoritmo, o tempo de execução para cada produção da gramática foi medido usando o comando *time* do Linux ao executar o código *Python*. A gramática executada foi o projeto arquitetônico de uma casa, que possui 19 regras. Em média, o tempo de execução para uma produção é de 0,370 segundos. Ou seja, para gerar 20 designs, o tempo total seria de 7,4 segundos. O maior gargalo do algoritmo é a detecção de cada regra sobre cada pedaço menor da forma, ou seja, quanto mais regras e quanto mais cômodos são gerados, maior o tempo de execução.

Considerando isso, o algoritmo representa uma aplicação viável para a geração automática de produções de uma gramática de formas aplicada para o contexto arquitetônico.

5 CONCLUSÃO

Para esse trabalho, foram definidos conceitos e fundamentação teórica necessária para alcançar os objetivos estipulados. No capítulo 4, foram-se criadas gramáticas de formas que representassem projetos arquitetônicos, e a tradução de suas estruturas em código permitiu o desenvolvimento de um algoritmo capaz de gerar produções com base em uma gramática de forma de entrada, automatizando a criação dessas produções e permitindo diversos *designs* diferentes e possibilidades diversas. Também foi possível comparar as gramáticas com projetos reais e garantir a viabilidade do algoritmo. O trabalho também foi avaliado como um grande auxiliador no meio educativo, já que as imagens são de grande ajuda visual para o etendimento do conceito de gramáticas e linguagens, podendo este ser utilizado nas disciplinas de Teoria da Computação e Linguagens Formais.

Para os trabalhos futuros, podem ser explorados os pontos em relação ao tamanho de um cômodo e a área máxima e mínima que ele pode ocupar, assim como verificar se uma forma pode ser sub-dividida se esta já esta em sua área mínima especificada, adicionando condições nas regras da gramática de forma a refletir ainda mais nosso mundo real.

Já no quesito educacional, esse algoritmo pode ser usado como base para a criação de um aplicativo voltado para o intuito educativo, com botões interativos para a geração dessas gramáticas.

Ainda na representação 2D, outras formas geométricas podem ser exploradas além de espaços retangulares, assim como a ampliação para outros tipos de estruturas como prédios, construções, meios de transporte (trens, barcos).

Outras formas de representação e exibição da forma resultante podem ser exploradas com a ajuda de alguma ferramenta 3D de criação de projetos arquitetônicos, criando gramáticas mais complexas.

REFERÊNCIAS

BONACIC, Vladimir; GIPS, James; STINY, George. Shape Grammars and Their Uses: Artificial Perception, Shape Generation and Computer Aesthetics. **Leonardo**, v. 10, nov. 1978. DOI: 10.2307/1573793.

CHOMSKY, N. Three models for the description of language. **IRE Transactions on Information Theory**, v. 2, n. 3, p. 113–124, 1956. DOI: 10.1109/TIT.1956.1056813.

DUARTE, Jose. Towards the mass customization of housing: The grammar Siza's houses at Malagueira. **Environment and Planning B: Planning and Design**, v. 32, p. 347–380, mai. 2005. DOI: 10.1068/b31124.

EXPLORE as plantas do Duo Concept Office. [*S.l.*: *s.n.*]. Disponível em: https://www.cyrela.com.br/empreendimentos/duo-concept-office#Plantas.

FLEMMING, Ulrich. More than the sum of parts: The grammar of Queen Anne houses. **Environment and Planning B: Planning and Design**, v. 14, p. 323–350, jul. 1987. DOI: 10.1068/b140323.

GU, Ning; BEHBAHANI, Peiman Amini. Shape Grammars: A Key Generative Design Algorithm. *In:* **Handbook of the Mathematics of the Arts and Sciences**. Edição: Bharath Sriraman. Cham: Springer International Publishing, 2018. P. 1–21. ISBN 978-3-319-70658-0. DOI: 10.1007/978-3-319-70658-0_7-1. Disponível em: https://doi.org/10.1007/978-3-319-70658-0_7-1.

KONING, H; EIZENBERG, J. The Language of the Prairie: Frank Lloyd Wright's Prairie Houses. **Environment and Planning B: Planning and Design**, v. 8, p. 295–323, set. 1981. DOI: 10.1068/b080295.

M. VERKERK, Nina. A general understanding of shape grammar for the application in architectural design, mar. 2014.

PAUWELS, Pieter *et al.* Shape Grammars for Architectural Design: *In:* DOI: 10.1007/978-3-662-47386-3_28.

PINTO, FABIANO DA SILVEIRA. Gramáticas de formas como modelo computacional teóricos, p. 83, 2010.

SIPSER, Michael. **Introduction to the Theory of Computation**. Third. Boston, MA: Course Technology, 2013. ISBN 113318779X.

TOSI, por Bruna. **Plantas de Casas Simples: +63 Projetos Funcionais para Casas Pequenas**. [*S.l.*: *s.n.*], dez. 2019. Disponível em: https://www.vivadecora.com.br/revista/plantas-de-casas-simples/. Acesso em: 11 jun. 2024.

ANEXO A – CÓDIGO FONTE

O código fonte desenvolvido para este projeto encontra-se integralmente disponível no repositório da autora no GitHub, acessível através do seguinte link: `https://github.com/palomacione/tcc`.

ANEXO B – ARTIGO

Neste apêndice será apresentado o artigo no formato SBC, referente ao presente projeto.

Um algoritmo para geração de projetos arquitetônicos utilizando gramática de formas

Paloma Zankely Alves de Oliveira Cione[1], **Rafael de Santiago**[1]

[1]Universidade Federal de Santa Catarina

Abstract. *The constant digitization and automation of various methods impact society across multiple sectors, also influencing art and other creative processes. Shape grammars, in the theory of computation, are a class of grammars that have a set of rules generating geometric shapes in their productions. Over the past decades, shape grammars have been considered a powerful tool in the creative process of architecture, becoming part of the parametric design set, where resources are shaped based on an algorithm. In this context, the main objective of this work is to create an algorithm that reflects in architectural creations, such as houses, buildings, and other structures, based on a specific defined style, analyzing its computational complexity and potential technological barriers that may be encountered.*

Resumo. *A constante digitalização e automatização de métodos diversos impacta a sociedade em múltiplos setores, influenciando também a arte e outros processos criativos. Gramáticas de formas, na teoria da computação, são uma classe de gramáticas que possuem um conjunto de regras que geram formas geométricas em suas produções. Com isto, nas últimas décadas, gramáticas de formas são consideradas uma ferramenta poderosa para o processo de criação na arquitetura, se tornando parte do conjunto do design paramétrico, em que recursos são moldados com base em um algoritmo. Nesse contexto, o objetivo principal deste trabalho é criar um algoritmo que reflita em criações arquitetônicas, como casas, prédios, e outros edifícios, se baseando em um estilo específico definido, analisando sua complexidade computacional e possíveis barreiras tecnológicas que possam ser encontradas.*

1. Introdução

Com a sua criação em meados de 1970 [Bonacic et al. 1978], gramáticas de formas foram introduzidas originalmente como uma maneira de analisar e sintetizar a arte, entendendo e identificando seus padrões e formas. Desde então, a utilização de gramáticas de formas foi ampliada e é altamente eficaz na produção de *design* e geração de formas com o intuitivo criativo, estético e visual.

Para o *design* e a arte autogerada, é suficiente e desejável a elaboração de formas que se entrelaçam e se sobrepõem, criando novas formas a partir das anteriores e assim criando infinitos *designs* que podem ter as mais diversas faces. Porém, em um projeto arquitetônico, é necessária a limitação de possibilidades e uma melhor definição destas. Por exemplo, como se é dito em [Pauwels et al. 2015], a demarcação de paredes e cômodos suficientemente grandes e funcionais. Podemos então definir que projetos arquitetônicos como casas e prédios tem sua diferença dos *designs* comuns gerados por gramáticas de

forma baseada na funcionalidade de tais estruturas, por isso, desde sua criação e até o presente momento, diferentes gramáticas que analisam e descrevem estilos arquiteturais foram desenvolvidas, visando também a possibilidade do entendimento genérico sobre a aplicação dessas regras.

1.1. Objetivos

O objetivo principal do projeto é o desenvolvimento de um algoritmo que produza de maneira automatizada as derivações de uma gramática de formas, criando projetos arquitetônicos de acordo com as regras da gramática, de modo que essa ferramenta impacte e auxilie tanto a criatividade quanto a inovação na nossa realidade.

Os objetivos específicos são (1) desenvolver uma implementação de um algoritmo que gere produções de uma gramática de formas; (2) expandir o algoritmo para que este consiga usar as limitações de projetos arquitetônicos; (3) conseguir criar formas que condizem com um estilo arquitetônico previamente definido; (4) avaliar o desempenho do algoritmo de forma a verificar se a aplicação é viável; (5) disponibilizar código-fonte e aplicação de maneira acessível ao público interessado;

2. Procedimento metodológico

Este trabalho almeja atingir os objetivos propostos na Seção 1.1 através das seguintes etapas:

1. **Revisão Bibliográfica**: revisão bibliográfica sistemática referente aos temas-chave, visando entender melhor a implementação e os passos necessários.
2. **Desenvolvimento**: desenvolvimento de um algoritmo que gere as produções de uma gramática de formas, adaptando-o para a geração de projetos arquitetônicos definidos pelas gramáticas de entrada;
3. **Experimentação**: desenvolver experimentos e coletar os resultados
4. **Análises**: comparar e analisar os resultados a fim de melhorar a aplicação;
5. **Disponibilização do código fonte**: Tornar o código fonte e os demais artefatos relevantes da aplicação desenvolvida em licença *open source*.

3. Conceituação

3.1. Gramáticas

Como definido por Chomsky [Chomsky 1956], uma gramática é um mecanismo que define o conjunto de regras de uma linguagem, que dita também sua estrutura, de forma a permitir que apenas determinadas combinações sejam válidas, isto é, sejam consideradas sentenças e parte da linguagem especificada. A produção de elementos definidos nestas regras forma uma gramática generativa, capaz de formar sentenças completas. Gramáticas também são um mecanismo importante na representação de linguagens potencialmente infinitas de forma finita, apenas com um conjunto de regras que podem gerar toda a linguagem.

Definição formal de gramática: G = (N, T, P, S)

- N: conjunto de variáveis não-terminais
- T: conjunto de variáveis terminais ($N \cap T = \emptyset$)
- P: conjunto finito de produções $P \subseteq (N \cup T = \emptyset)^+ \rightarrow (N \cup T = \emptyset)^*$
- S: símbolo inicial $S \in N$

3.2. Gramáticas de formas

No mesmo sentido do mecanismo de geração utilizado nas gramáticas generativas, as gramáticas de forma são representações visuais e geométricas de gramáticas, com regras definidas espacialmente ao invés de textualmente, utilizadas para transformar os símbolos iniciais e atingir diferentes formas.

Tendo sua definição como uma quádrupla : G = (V_t,V_m,R,I) onde:

- V_t é um conjunto finito de formas.
- V_m é um conjunto finito de formas de maneira que $V_t \cap V_m = \emptyset$.
- R é um conjunto finito de regras
- I é uma forma inicial e subconjunto de V_t e V_m

As formas contidas em V_t são geradas a partir do conjunto de produtores V_m de acordo com as regras R que serão aplicadas durante sua produção a partir da forma inicial I.

Uma regra é descrita como A $\rightarrow$ B, dividindo-se em lado esquerdo e lado direito.

Uma regra pode ser aplicada quando a sua parte esquerda (A), que no caso, é uma forma, é detectada em uma outra forma C.

Essa detecção pode ocorrer via translação, rotação, escalonamento, entre outros processos lineares.

Exemplo de gramática de forma:

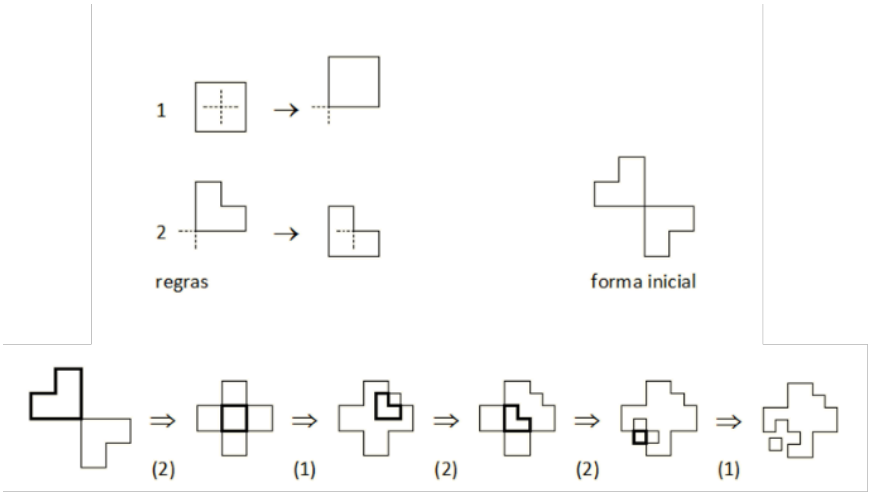

Figura 1. Exemplo de aplicação das regras de uma gramática de formas. Retirada de [PINTO 2010]

4. Trabalhos relacionados

4.1. Gramáticas de casas Queen Anne

O estilo arquitetônico Queen Anne foi um popular estilo vitoriano que emergiu nos Estados Unidos no período de 1880. Seus traços principais são as fachadas assimétricas, colunas clássicas, varandas e torres em formas redondas, quadradas ou poligonais no geral. Com a gramática elaborada por [Flemming 1987], é possível exemplificar e gerar um exemplo básico de uma típica casa Queen Anne. Porém, a gramática não consegue

gerar uma casa completa, em detalhes, devido a seus aspectos muito complexos que não conseguem ser aplicados a um par de regras, mas que podem ser manualmente inseridos por um designer ou arquiteto envolvido em um projeto. Um diferencial é que a primeira parte dessa gramática é bi-dimensional, enquanto a segunda parte é tri-dimensional.

Esse trabalho é um bom exemplo de uma ferramenta auxiliar no processo de criação e desenvolvimento, sem criar completamente todos os passos necessários, que envolve uma percepção humana e qualificada para terminar a projeção.

4.2. Gramáticas de casas Malagueira

A gramática de casas Malagueira têm seu design baseado nas construções de trinta e cinco casas criadas por Álvaro Siza em Malagueira, Portugal, entre 1977 e 1996. Essa gramática foi co-desenvolvida pelo próprio arquiteto português, sendo considerada uma extensão de seu trabalho, e em certo momento o mesmo já não conseguia propriamente diferenciar entre casas feitas pelo arquiteto ou geradas pela gramática. A gramática em si possui três estágios e é bi-dimensional, e possui muitos detalhes funcionais em seu estágio mais avançado, como a definição de cozinhas, lavanderias e outros espaços. O trabalho de Jose Duarte [Duarte 2005] foi muito significante, já que propôs um modelo capaz de gerar diversos designs de habitação em massa, sem repetições. Em suas conclusões, Duarte afirma a importância de um programa de computador que, dado uma gramática que pode ser sistematizada em forma de linguagem, faria a codificação da mesma e permitiria um uso ainda mais eficiente do design de Siza, principalmente na aceleração do processo de geração.

5. Desenvolvimento

Para a execução do algoritmo, precisaremos de elementos e métodos auxiliares para a detecção e transformação da formas presentes na gramática.

- **Representação**:Para todos os desenvolvimentos práticos foi definido o uso da linguagem Python, por possuir bibliotecas de grande auxílio para este trabalho, juntamente com o modelo orientado a objetos com os seguintes modelos:
    1. `Shape`: representação de uma forma
       Cada forma possui:
       `geometry ->` uma geometria representada como um *Polygon* da biblioteca *shapely*
       `type ->` identificação em *string* do tipo de cômodo que a forma representa.
       `adjacent_shape- >` lista de cômodos adjacentes do tipo *Shape*
    2. `Rule`: representação de uma regra
       Essa é a classe pai de representação de uma regra da gramática, ou seja, para cada regra criada uma classe filha é implementada.
       Cada regra possui:
       `left_side ->` identificação em *string* do lado esquerda da regra, representando um cômodo gerador
       `right_side_function ->` função do lado direito da regra que define a produção e transformação da forma, retorna a lista de novas formas criadas.

`condition - >` função *boolean* que representa se a condição para que a regra seja aplicada é verdadeira
`detect_shape - >` função *boolean* que detecta a forma atual no lado esquerdo da regra
`apply_shape - >` função comum que aplica o lado direito da regra

3. `Grammar`: representação de uma gramática
   Classe com todos os elementos de uma gramática: um conjunto de regras, e uma forma inicial.
   `rules ->` conjunto de regras
   `initial_shape ->` Objeto do tipo Shape representando a forma inicial

- **Transformação da forma:** A transformação da forma é o processo linear de subdivisão que ocorre em cima da geometria do polígono (pontos) que representa a forma. Para isso, foram criados métodos utilitários com todas as possibilidades de transformação de um polígono para as gramática criadas na seção **??**. Os métodos construídos transformam o polígono de acordo com que o lado direito das regras das gramáticas requerem, o que envolve a divisão do polígono verticalmente ou horizontalmente, em diferentes pontos.
- **Detecção da forma:** Para a detecção de uma forma do lado esquerdo de uma regra, foi-se usada uma identificação direta do tipo do cômodo que o polígono representa, uma vez que a gramática se baseia em uma ideia de que as funcionalidades de cada cômodo podem gerar outros cômodos que tem uma ligação direta com a funcionalidade detectada, ou outros modos daquela funcionalidade ser representada e/ou dividida. Isso faz com que a geometria do lado esquerdo da regra seja dinâmica, ou seja, uma vez que um tipo de cômodo é identificado do lado esquerdo da regra, a regra assume aquela forma para que a regra aplicada seja em cima da forma sendo detectada.
  Além disso, pensando na adaptação para os projetos arquitetônicos, para cada regra implementada é possível se definir uma condição para que a regra possa ser efetivamente detectada, já que o elemento de recursão nem sempre é desejado na adaptação destes modelos reais.
- **Aplicação da regra:** Para a execução da aplicação do lado direito da regra, cada regra implementada possui sua própria função de transformação que é atribuída à *right_side_function*. A chamada do método executa a função *right_side_function* estabelecida pela regra

Algorithm 1 Geração de formas

Entrada: Gramática de formas representada por $shape\_grammar$
$current\_shape$.append($shape\_grammar.initial\_shape$)
$has\_possible\_rules \leftarrow$ true
while ($has\_possible\_rules$ is true:) **do**
  for $shapes$ in $current\_shape$ **do**
    for $rule$ in $shape\_grammar.rules$ **do**
      if $rule.detect\_shape(shape, current\_shape)$ is true: **then**
        $possible\_rules$.append($rule$)
      end if
    end for
    if $possible\_rules$.size() ¿ 0 **then**
      $picked\_rule$ = $pick\_rule(possible\_rules)$
      $new\_shapes \leftarrow picked\_rule.apply\_rule(current\_shape, shape)$
      $current\_shape$.append(new_shapes)
    else
      $has\_possible\_rules \leftarrow$ false
    end if
  end for
end while
$plot\_shape(current\_shape)$

Dada a gramática de formas de entrada, a variável *current_shape* é atualizada com a forma inicial e a *has_possible_rules* recebe true para que o algoritmo comece. A forma atual é representada por polígonos menores com suas respectivas geometrias e a primeira iteração percorre por todos eles, adicionando regras possíveis a uma lista. Para garantir mais diversidade de produções, a regra é escolhida aleatoriamente e então seu lado direito é executado, garantindo a transformação dos polígonos e a geração de novas formas. A forma atual recebe suas novas formas transformadas, montando a nova forma atual para o próximo ciclo. Caso não hajam mais regras a serem aplicadas, o programa termina e a forma resultante é exibida.

5.1. Criação da gramática de entrada

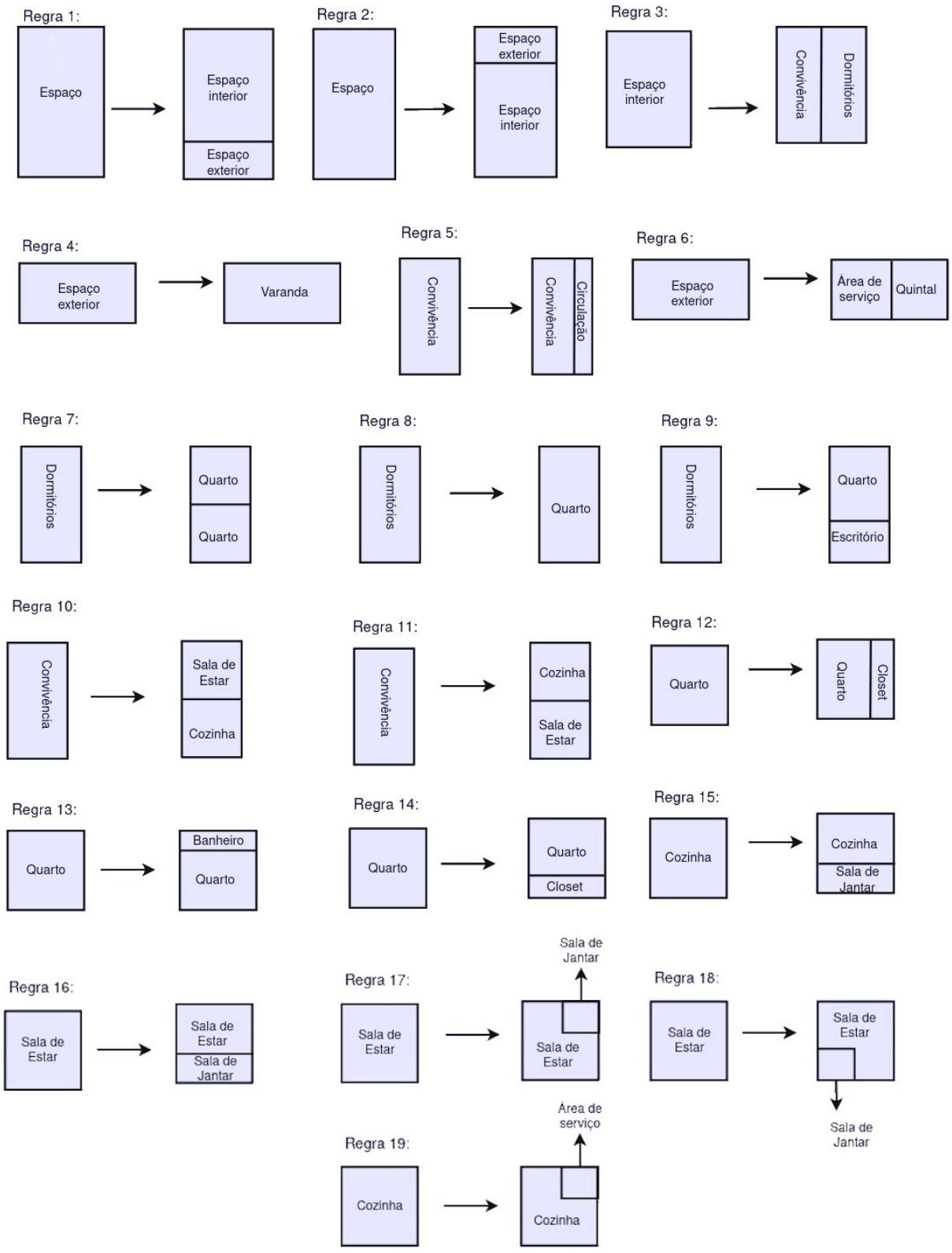

Figura 2. Regras da gramática de formas para o projeto arquitetônico de uma casa

6. Experimentação e Análise

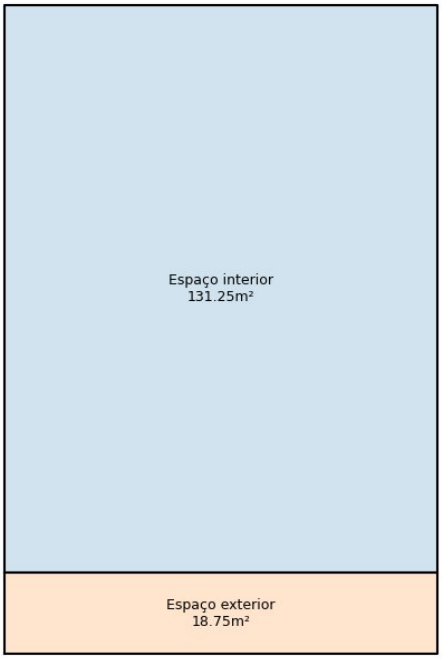

Figura 3. Experimento 1: Detecção e aplicação da regra 1

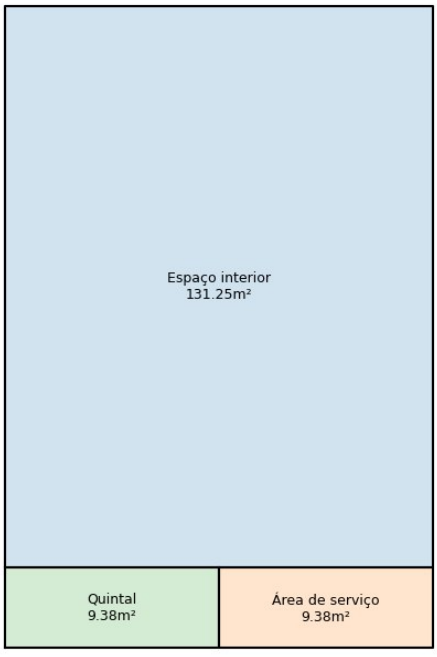

Figura 4. Experimento 1: Detecção e aplicação da regra 6

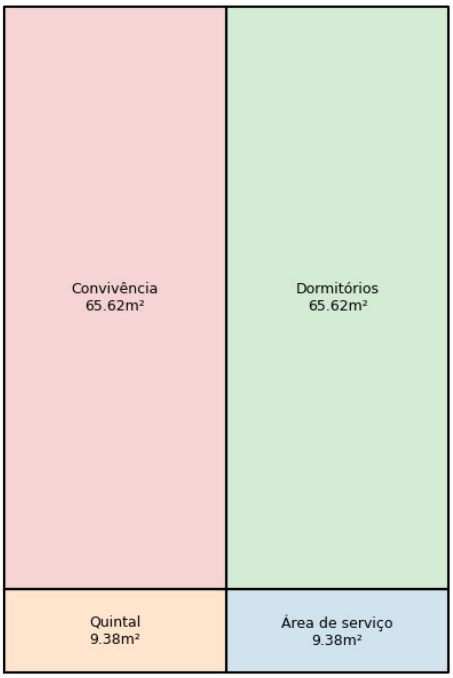

Figura 5. Experimento 1: Detecção e aplicação da regra 3

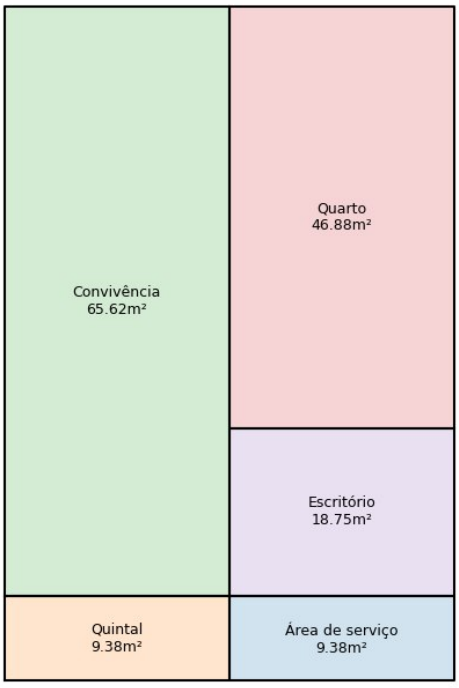

Figura 6. Experimento 1: Detecção e aplicação da regra 9

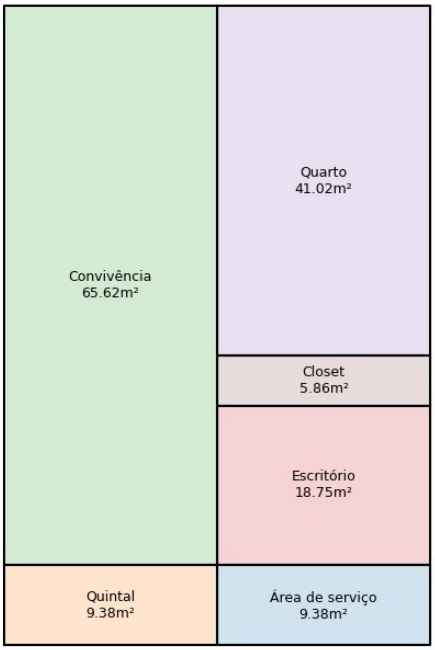

Figura 7. Experimento 1: Detecção e aplicação da regra 14

Figura 8. Experimento 1: Detecção e aplicação da regra 11

Figura 9. Experimento 1: Detecção e aplicação da regra 15

Figura 10. Experimento 1: Detecção e aplicação da regra 13

A figura 10 representa a forma resultante final da execução do algoritmo, já que mais nenhuma regra foi detectada. A execução detectou e aplicou 8 regras dentre as 19, e gerou 9 cômodos diferentes. As regras aplicadas geraram uma casa de 1 quarto e 1 escritório, com uma sala de jantar entre a cozinha e a sala de estar e uma área de serviço externa.

Ao comparar plantas reais com a figura 10 resultante, é possível perceber que a produção da gramática conseguiu reproduzir requisitos posicionais de projetos arquitetônicos, como a divisão dos espaços de convivência que são geralmente adjacentes (Sala de Estar, Cozinha, Sala de Jantar) e os quartos que são adjacentes a banheiros. Além disso, os espaços externos e de serviço também foram contemplados e posicionados de forma que façam sentido com o mundo real.

7. Conclusão e trabalhos futuros

Para esse trabalho, foram definidos conceitos e fundamentação teórica necessária para alcançar os objetivos estipulados. Foram-se criadas gramáticas de formas que representassem projetos arquitetônicos, e a tradução de suas estruturas em código permitiu o desenvolvimento de um algoritmo capaz de gerar produções com base em uma gramática de forma de entrada, automatizando a criação dessas produções e permitindo diversos *designs* diferentes e possibilidades diversas. Também foi possível comparar as gramáticas com projetos reais e garantir a viabilidade do algoritmo. O trabalho também foi avaliado como um grande auxiliador no meio educativo, já que as imagens são de grande ajuda visual para o etendimento do conceito de gramáticas e linguagens, podendo este ser utilizado nas disciplinas de Teoria da Computação e Linguagens Formais.

Para os trabalhos futuros, podem ser explorados os pontos em relação ao tamanho de um cômodo e a área máxima e mínima que ele pode ocupar, assim como verificar se uma forma pode ser sub-dividida se esta já esta em sua área mínima especificada, adicionando condições nas regras da gramática de forma a refletir ainda mais nosso mundo real.

Já no quesito educacional, esse algoritmo pode ser usado como base para a criação de um aplicativo voltado para o intuito educativo, com botões interativos para a geração dessas gramáticas.

Ainda na representação 2D, outras formas geométricas podem ser exploradas além de espaços retangulares, assim como a ampliação para outros tipos de estruturas como prédios, construções, meios de transporte (trens, barcos).

Outras formas de representação e exibição da forma resultante podem ser exploradas com a ajuda de alguma ferramenta 3D de criação de projetos arquitetônicos, criando gramáticas mais complexas.

Referências

Bonacic, V., Gips, J., and Stiny, G. (1978). Shape grammars and their uses: Artificial perception, shape generation and computer aesthetics. *Leonardo*, 10.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124.

Duarte, J. (2005). Towards the mass customization of housing: The grammar siza's houses at malagueira. *Environment and Planning B: Planning and Design*, 32:347–380.

Flemming, U. (1987). More than the sum of parts: The grammar of queen anne houses. *Environment and Planning B: Planning and Design*, 14:323–350.

Pauwels, P., Strobbe, T., Eloy, S., and Meyer, R. (2015). Shape grammars for architectural design:. volume 527.

PINTO, F. D. S. (2010). Gramáticas de formas como modelo computacional teóricos. page 83.