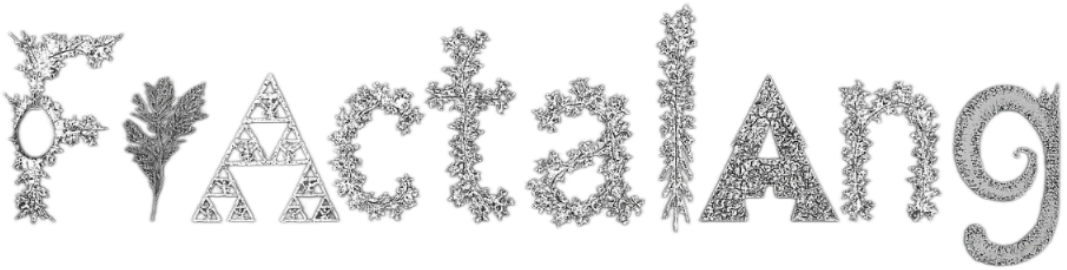


Informe Final - Fractalang

Diseño e Implementación de un Lenguaje

v1.0.4

1. Equipo

Nombre	Apellido	Legajo	E-mail
Joaquín José	Huerta	64252	johuerta@itba.edu.ar

Simón	Marabi	61626	smarabi@itba.edu.ar
Nicolás	Rivas	64292	nrivas@itba.edu.ar
Toribio	Vitón Sconza	64275	tvitonsconza@itba.edu.ar

2. Repositorio

La solución y su documentación serán versionadas en: https://github.com/nrivas-itba/TLA.

3. Introducción

En el presente informe se documenta el diseño e implementación de Fractalang, un *Domain Specific Language* (*DSL*) orientado a la generación de fractales mediante reglas, iteraciones y construcciones geométricas simples.

La idea fue desarrollar un lenguaje expresivo pero accesible, que permita describir de forma declarativa fractales clásicos como *Mandelbrot*, *Sierpinski* o *Barnsley Fern* así como nuevos patrones a partir de modificaciones en sus fórmulas, parámetros y transformaciones. El compilador desarrollado procesa programas escritos en Fractalang y produce como salida una imagen .bmp sin compresión.

El diseño del lenguaje se basa en una sintaxis simple y case sensitive, con bloques definidos por indentación y terminación por salto de línea.

Este informe resume el modelo computacional del lenguaje, las decisiones de diseño adoptadas, la arquitectura del compilador, las dificultades encontradas y las posibles extensiones futuras.

4. Modelo Computacional

4.1. Dominio

Fractalang es un lenguaje de programación con el objetivo de construir y graficar fácilmente fractales de diversos tipos, por medio de vistas y transformaciones. El lenguaje permite a los usuarios realizar los fractales de una manera sumamente simple, aunque

requiere conocimientos previos de los mismos para poder graficarlos. Adicionalmente también se permite la generación de figuras geométricas simples, como por ejemplo cuadrados y triángulos.

Los fractales o figuras geométricas generadas lo hacen en formato BMP. Estos no presentan compresión y también se pueden pasar a otros formatos fácilmente.

4.2. Lenguaje

Fractalang es un lenguaje declarativo, no tipado y case-sensitive, diseñado específicamente para la definición y generación de fractales. Este lenguaje se construye alrededor de reglas de dibujo, transformaciones iterativas y expresiones matemáticas que describen el comportamiento del fractal en el plano.

Entre las características del lenguaje se encuentran las siguientes:

- Todas las magnitudes numéricas se tratan como números de doble precisión (double). Esto permite manejar tanto coordenadas enteras como fracciones complejas necesarias para el cálculo de fractales sin errores de cast.

- En cuanto a los tipos de datos, no se requiere la declaración explícita de tipos. El lenguaje infiere los tipos basándose en los literales.

- Hay soporte nativo para coordenadas, rangos y manejo transparente de números complejos en fractales de tiempo de escape (donde z representa un par de coordenadas).

- Para fractales geométricos (por ejemplo, el de Sierpinski), el lenguaje permite llamadas recursivas. La terminación no está garantizada por el compilador, sino que depende de una condición de parada explícita definida por el usuario.

- Se permite nativamente la creación de fractales que dependen de probabilidades. Esto implica un comportamiento no determinista.

- Las variables definidas en los parámetros de una regla son locales a esa regla y ocultan cualquier identificador externo.

- Existen variables reservadas de solo lectura dependientes del contexto de renderizado. Estas son solo 2, :x: e :y:, las cuales representan la coordenada del píxel actual en fractales de escape.

- El usuario no gestiona punteros ni asignación de memoria. El compilador se encarga de eso automáticamente.

- Las figuras se definen en un sistema de coordenadas cartesianas definido en view, independiente de la resolución de salida en píxeles.

Por otro lado, en el lenguaje los valores por defecto de los colores son blanco y negro, a su vez la resolucion por defecto es 1920 x 1080.

5. Implementación

5.1. Frontend

5.1.1. Análisis Léxico (Flex)

El analizador léxico se implementó utilizando Flex, responsable de convertir el flujo de caracteres del programa fuente en tokens bien definidos. Los tokens reconocidos están directamente alineados con las construcciones del lenguaje detalladas en la especificación.

Se distinguen explícitamente tokens como: size, view, color, rule, start, call, stop , escape, until, max,draw, polygon, point, transform, scale, rotate, translate, shear, points y operadores relacionales y aritméticos e indentación.

Estas palabras clave fueron tratadas como tokens reservados, insensibles a ambigüedad. El lexer reconoce:

- Identificadores de reglas y variables (combinación de letras, números y guiones bajos)
- Números enteros y flotantes
- Números hexadecimales para colores en la instrucción color
- Paréntesis, corchetes, comas, dos puntos, usados en la sintaxis estructural del lenguaje

Como Fractalang define bloques mediante indentación, el analizador léxico detecta:

- Nivel de indentación de cada línea
- INDENT y DEDENT

- Salto de línea como delimitador de instrucciones.

Esto permite que el parser reconozca correctamente el alcance de cada bloque, como el contenido de una regla.

5.1.2. Análisis Sintactico (Bison)

El analizador sintáctico se implementó con Bison, construyendo el AST y validando que el programa respete la gramática del lenguaje.

El parser:

1. Valida la estructura global del programa: size (opcional), view (obligatoria), color (opcional), una o más rule y un único start.

2. Construye nodos del AST para cada instrucción.

3. Maneja listas de instrucciones, listas de transformaciones y listas de puntos de forma recursiva.

5.2. Backend

El backend es responsable de tomar el AST validado y lo transforma en una imagen en formato .bmp que representa la figura o fractal especificado. Esta etapa se divide en tres componentes principales, el generator, el interpreter y el bitmap.

El generator sería como el punto de entrada al backend. Su función principal es recibir el CompilerState, verificar la existencia de un AST válido y delegar la ejecución al módulo de interpretación.

Por otro lado, el núcleo del backend reside en el interpreter, el cual recorre el AST para ejecutar la lógica del fractal. Este puede hacer muchas tareas, como la configuración del entorno (por ejemplo, el tamaño de imagen, colores, etc.), evalúa expresiones, ejecuta reglas y se encarga de la recursividad, entre otras.

Por último, el bitmap separa la lógica matemática de la generación de archivos. Este componente abstrae la manipulación de píxeles y la codificación del formato de archivo. Crea el archivo .bmp, y dibuja los píxeles individuales.

5.3. Adicionales

No hay adicionales significativos.

5.4. Dificultades Encontradas

Si bien durante la primera entrega no hubo grandes sobresaltos, al principio se tenía solamente una idea sin concretar la cual generaba muchas inconsistencias como por ejemplo sobre si usar indentación o no y otros aspectos que "quedaron en el aire" hasta que se realizó el stage II del proyecto.

Para el desarrollo de la segunda entrega, se encontró un caso muy específico de memory leak al intentar hacer un fractal de Mandelbrot. Sin embargo, luego de consultarlo con la cátedra, se decidió ignorar el problema ya que este se consideró muy específico y rebuscado, de forma tal que no afectaría normalmente la experiencia del usuario.

Para el desarrollo de la tercera entrega, se encontró el problema de que había unos ajustes que hacer respecto de la segunda entrega. Entre los caminos que podíamos tomar se encontraba o ignorarlos pero hacer más difícil el análisis semántico o encargarnos de redefinir todo, pero hacer el análisis posterior más sencillo. Se optó por esta última opción, ya que se consideraba mejor estilo.

Por otro lado, también en la tercera entrega se encontró el desafío de finalmente, una vez validado el código de entrada tener que finalmente dibujar los fractales y figuras geométricas. Para esto si bien se estudiaron diversas librerías que lo permitían, como por ejemplo, stb_image_write o Cairo, al final lo que se terminó decidiendo hacer fue una implementación "casera".

6. Futuras Extensiones

Para una futura versión de Fractalang se tienen varias mejoras en mente.

En primer lugar, una de las limitaciones más importantes es que se generan los fractales en solamente el formato bmp. Si bien este presenta sus ventajas claras, se espera que para una futura versión del lenguaje se aprovechen las fortalezas del formato para que haya un soporte nativo en otros formatos. Entre estos se encontrarían los formatos SVG, PNG, JPEG, JPG e incluso TIFF.

En segundo lugar, para la creación de los fractales se utilizan operaciones básicas de transformación. Si bien se pueden realizar todas las operaciones posibles (en 2 dimensiones) utilizando solamente 3 de las operaciones, hay 4 disponibles para mejorar la experiencia del usuario. Estas son TRANSLATE, SCALE, ROTATE y SHEAR. En una futura expansión del lenguaje se podría ampliar este conjunto de operaciones. Por ejemplo, futuras adiciones permitirían operaciones REFLECTION o más sencillamente un FLIP HORIZONTALLY o FLIP VERTICALLY (un caso específico de reflexión). Si bien estas operaciones se pueden realizar

en la versión actual de Fractalang, agregarlas explícitamente mejoraría la experiencia del usuario.

Por otro lado, se mencionaron previamente operaciones que se pueden hacer ya en la corriente versión del lenguaje, por medio de las operaciones disponibles. Todas estas se denominan transformaciones lineales (1). Para una futura versión del lenguaje, se podrían agregar transformaciones no lineales, las cuales actualmente no hay forma de hacerlas como combinación de las lineales permitiendo torcer, curvar y deformar las figuras como nunca antes vistas. Esto podría ser cubierto por una operación WARP .

7. Conclusiones

A manera de conclusión se puede mencionar que durante la cursada de la materia se aprendió acerca de los compiladores y el proceso de reconocimiento de lenguajes. No solo eso sino su implementación fundamental, concretando una idea la cual había sido originada en otras materias, más particularmente en Arquitectura de Computadoras (72.08).

Si bien el desarrollo no fue un camino fácil y queda mucho espacio para mejoras como se especificó anteriormente, se considera el resultado obtenido al final como exitoso.

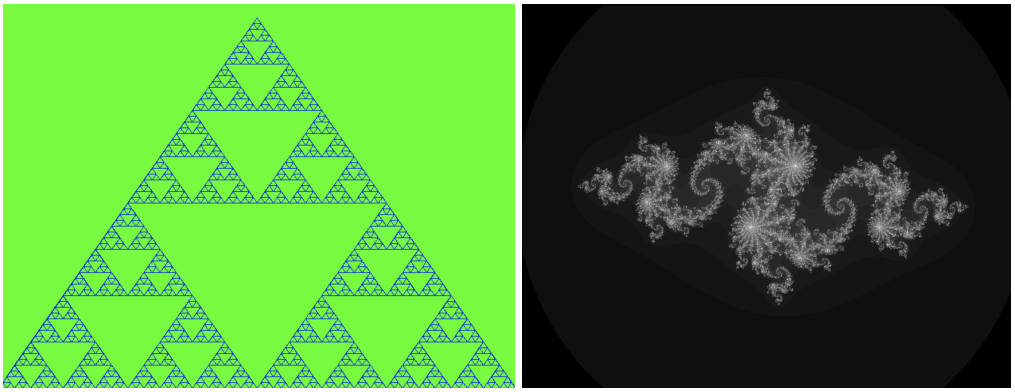
8. Referencias

No las hay.

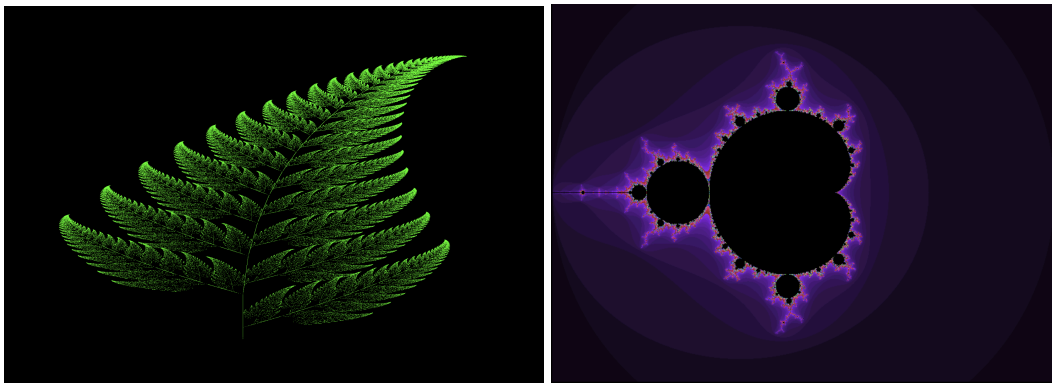
9. Bibliografia

(1) Transformation matrix. (2022, May 15). Wikipedia. https://en.wikipedia.org/wiki/Transformation_matrix. Consultado el 28 de noviembre de 2025.

Anexo con imágenes generadas utilizando Fractalang



Figuras 1 y 2. (De izquierda a derecha) Fractales de Sierpinski y de Julia.



Figuras 3 y 4. (De izquierda a derecha) Fractales de Barnsley Fern y de Mandelbrot.