

Contribuciones a GenoMus

Rediseño de un motor de cómputo funcional
sobre estructuras musicales

Autor

Miguel Pedregosa Pérez

Tutores

Miguel Molina Solana

José López Montes



**UNIVERSIDAD
DE GRANADA**

ETSIIT
Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación



Índice

- Introducción
- Análisis
- Planificación
- Desarrollo
- Resultados

Software desarrollado

Ref: p15

- Software libre
- Código abierto
- Licencia MIT
- Publicado en Github



 [mipdr / genomus-core](#) Public


C++ library that implements the core features of the GenoMus project

 MIT license

☆ 0 stars 🍴 0 forks

 [mipdr / genomus-core-js](#) Public

NodeJS bindings for genomus-core

 MIT license

☆ 0 stars 🍴 0 forks

Introducción

Motivación y objetivo

Se propone realizar una contribución al proyecto GenoMus para el fomento de su **calidad y robustez como proyecto de software libre**.

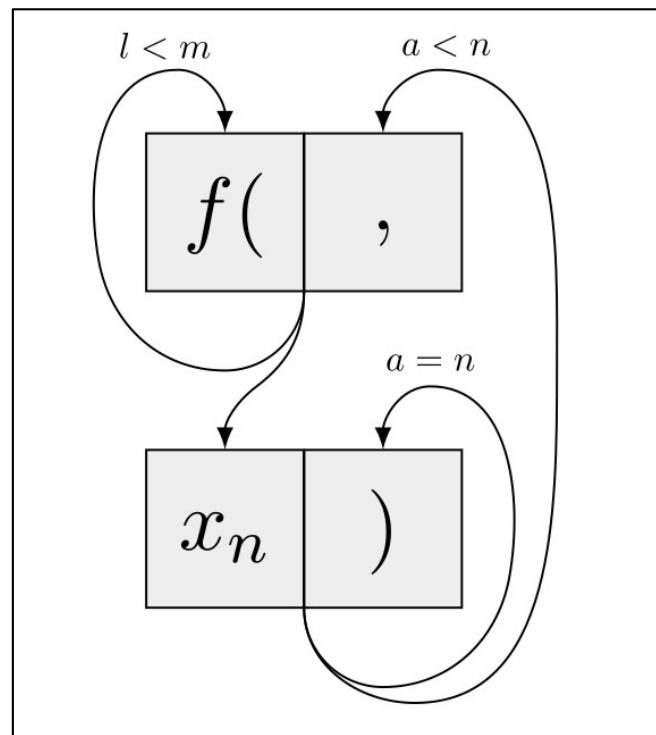
La contribución tendrá como objetivo la **optimización del motor de cómputo de GenoMus**.

GenoMus (/xe'nomus/ o /xeno'mus/)

“GenoMus es una herramienta de creatividad asistida por computadora basada en la metaprogramación autónoma de genotipos musicales. Estos genotipos codifican procedimientos musicales usando un metalenguaje musical basado en programación funcional, y su evaluación da lugar a fragmentos musicales denominados fenotipos.”

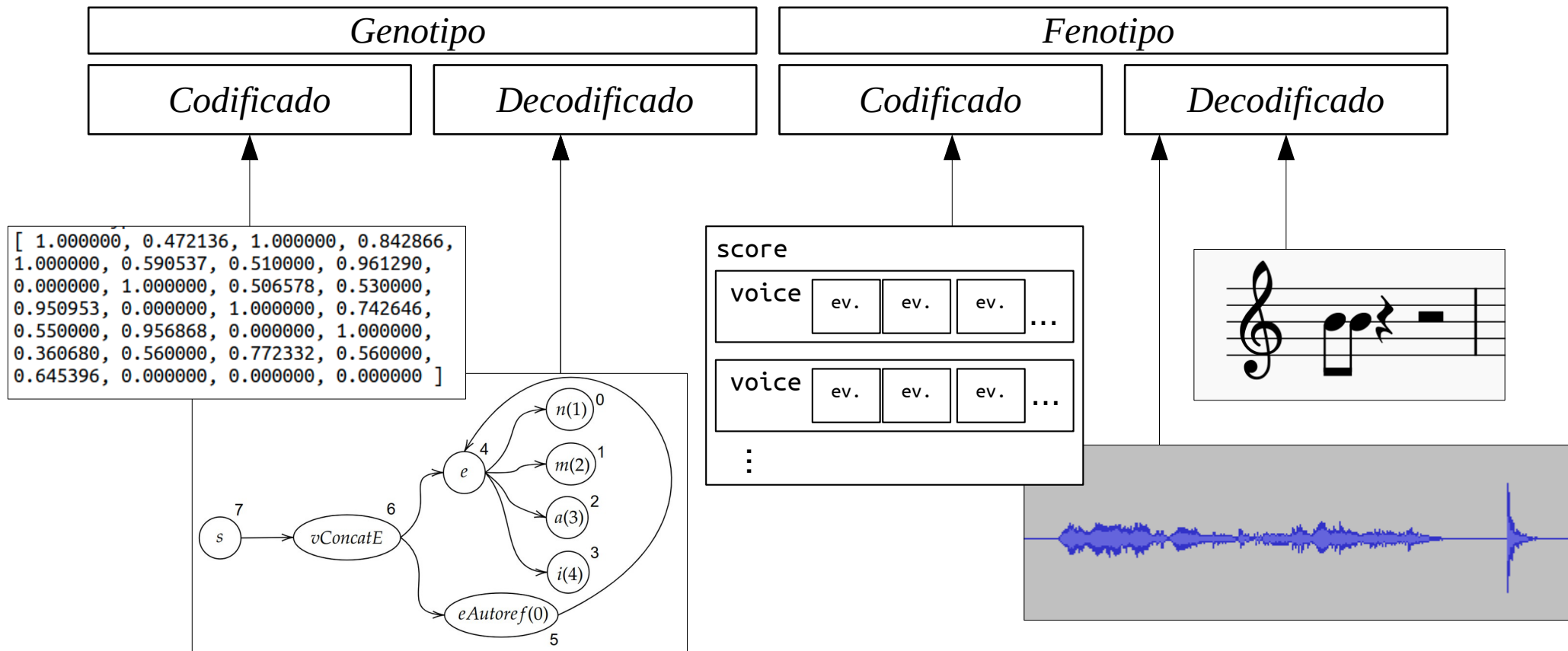


José López-Montes
lopezmontes



Introducción

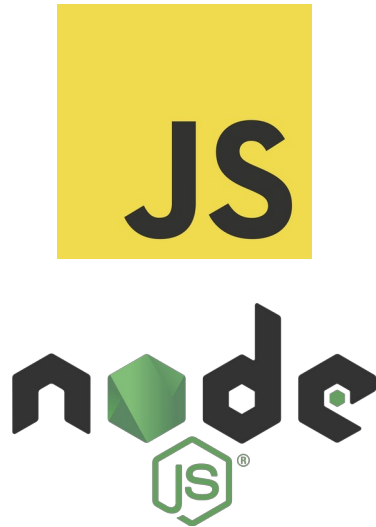
Ref: p21, 22



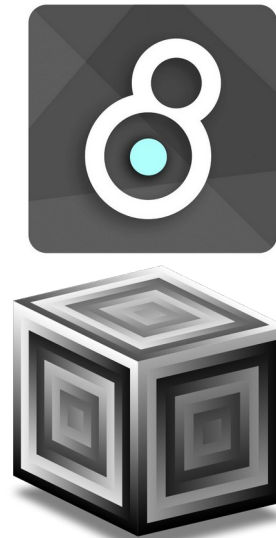
Introducción

Tecnologías de GenoMus

Motor de cómputo



UI & I/O



Análisis

El problema de GenoMus

El rendimiento:

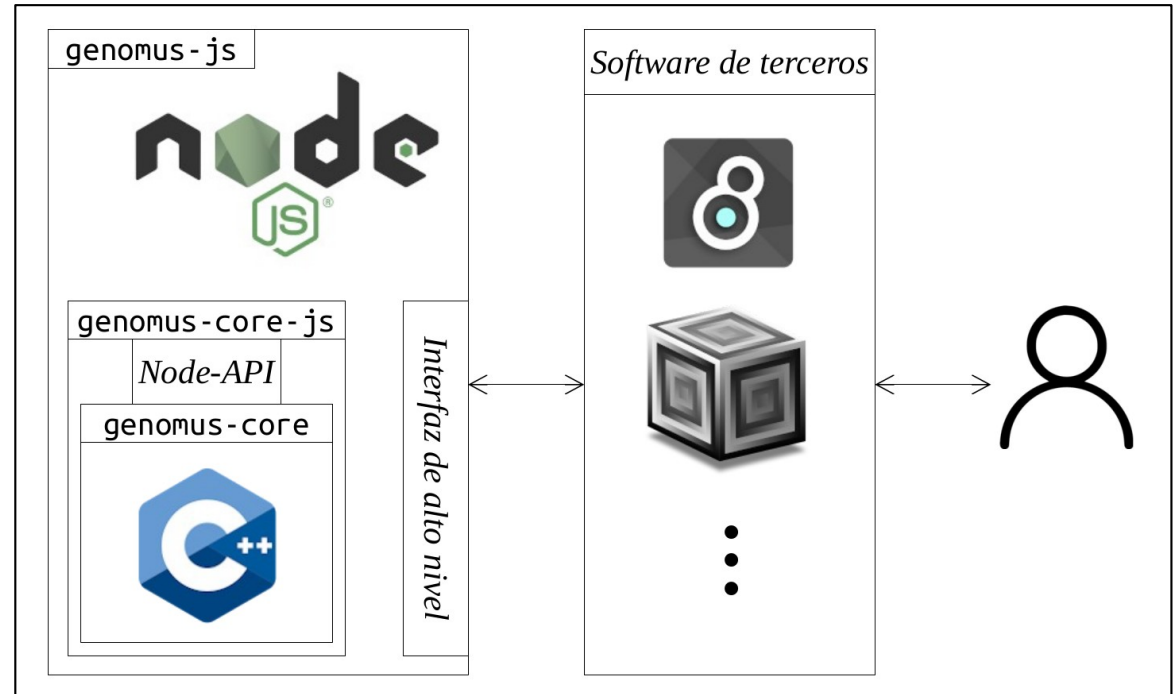
- Afecta la usabilidad
- Obliga a limitar el espacio de expresiones de cómputo
- Limita la exploración del usuario

La base de código:

- Es difícil de abordar
- Carece de modularización
- Convenciones de código minimales

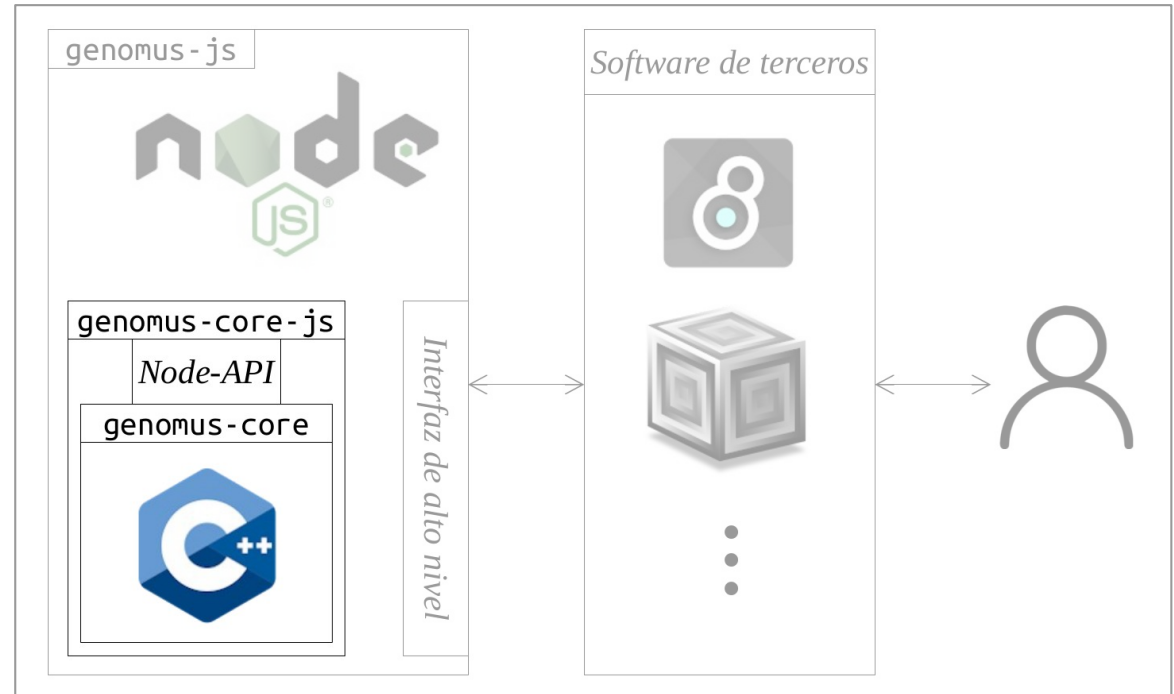
Elección de tecnologías

- Motor de cómputo en C++
- Funcionalidades de alto nivel mantenidas del prototipo



Alcance del proyecto

Este proyecto tiene el objetivo el rediseño del motor de cómputo. Así, el proyecto perseguirá la entrega de un MVP de los módulos de software resaltados en la ilustración.

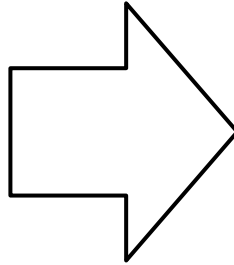


Planificación

Metodología SCRUM

Dificultades :

- Usuario final → Desarrollador
- MVP complejo
- Capacidad de desarrollo inestable



Características de la metodología:

- Sprints flexibles
- QA a través de TDD
- Planificación dinámica de historias e hitos

Tecnologías para el seguimiento del desarrollo



- ♦ Control de versiones
- ♦ Flujo GitFlow



- ♦ Seguimiento del proyecto:
 - ♦ *Historias de usuario*
 - ♦ *Tablero KanBan*
 - ♦ *Backlog*
 - ♦ *Estadísticas de progreso*

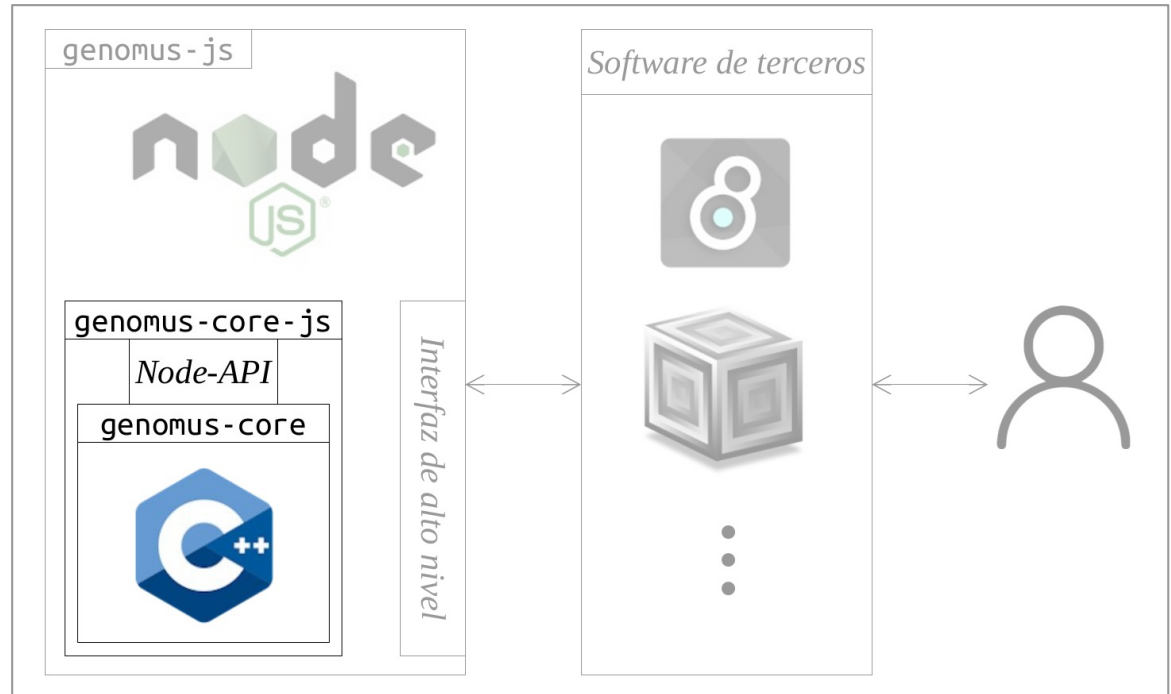


- ♦ Sincronización con Jira
- ♦ Lanzamiento de test automáticos con Github Workflows

Desarrollo

Arquitectura del software

- Paquetes a implementar:
 - genomus-core
 - genomus-core-js



genomus-core

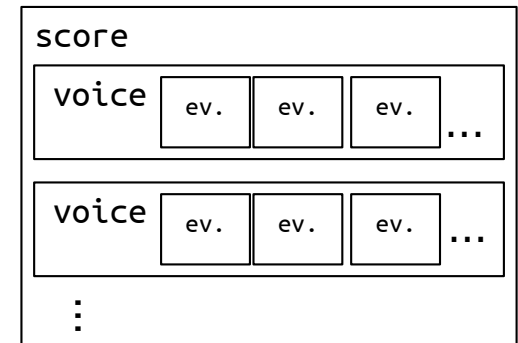
- Biblioteca estática de C++ (*genomus-core.a*)
- ISO C++20 con CMake
- Motor de cómputo de GenoMus:
 - Librería de funciones
 - Framework de cómputo funcional
 - Modelos de datos



CMake
Cross-platform Make

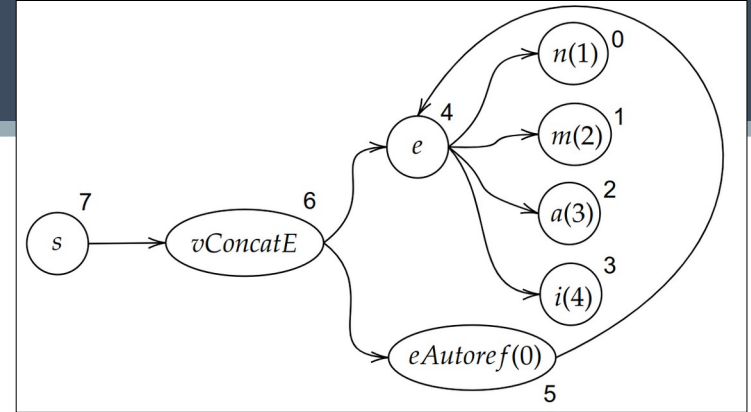
genomus-core: Fenotipos

- Tipos de fenotipo: *score*, *voice*, *event*, *parameter*...
- Árbol homogéneo en tiempo de compilación (TDA EncodedPhenotype)
- Polimorfismo en tiempo de ejecución a través de **Prototipos**
- Polimorfismo con tipado dinámico en tiempo de ejecución



genomus-core: Genotipos

- Funciones GenoMus (TDA GFunction)
 - Funciones declarables en tiempo de ejecución (*a través de clausuras y lambdas*)
 - Polimorfismo funcional a través de **tipado dinámico**
 - A veces funciones puras
- Árboles sintácticos (TDA GTree)
 - Árboles sobre contenedores de nodos
 - **Evaluación del genotipo es función pura durante la ejecución del software**



genomus-core: Modelo de Cómputo

- Evaluación usual de funciones
- Expresiones con notación prefija y paréntesis
- Algoritmo de evaluación funcional por profundidad:
 - Sin composición de funciones
 - Sin evaluación lazy
 - Sin máquina de reducción de grafos

genomus-core: GTest

- Test Framework para C++
- Construido para el proyecto
- Inspirado en MochaJS
- Tests lanzados automáticamente en Github

GTest

```
miguel@portatil:~/TFG/genomus-core$ bash run_tests.sh
+ TEST: Encoded Phenotypes Test
  - Test case #1: Parameter declaration -> OK after 0.028233ms
  - Test case #2: Bad Event declaration -> OK after 0.092416ms
  - Test case #3: Encoded phenotype to normalized vector -> OK after 1.1835ms
  - Test case #4: List types -> OK after 1.26773ms
- 4/4 test cases were successful after 1.27521ms

+ TEST: Decoded GenotypesTest
  - Test case #1: Functional tree build -> OK after 0.314859ms
  - Test case #2: Concat functions -> OK after 0.366906ms
  - Test case #3: Autoreferences -> OK after 0.597672ms
  - Test case #4: Lists -> OK after 0.610968ms
  - Test case #5: Random functions -> OK after 0.78173ms
- 5/5 test cases were successful after 0.788922ms

+ TEST: Parser test
  - Test case #1: Check parser tokens -> OK after 1.27272ms
  - Test case #2: Genotype to string to genotype -> OK after 2.4804ms
- 2/2 test cases were successful after 2.48854ms

+ TEST: Encoded Genotypes Test
  - Test case #1: Decoded genotype to encoded genotype -> OK after 0.083746ms
  - Test case #2: Encoding integers -> OK after 0.12721ms
  - Test case #3: Vector aproximation -> OK after 0.143969ms
  - Test case #4: Germinal vector to genotype -> OK after 0.218584ms
  - Test case #5: Normalize random vector -> OK after 1.45726ms
  - Test case #6: Normalize vector with lists -> OK after 1.52948ms
  - Test case #7: Germinal vector to expression -> OK after 178.774ms
- 7/7 test cases were successful after 178.79ms
```

GTest

```
miguel@portatil:~/TFG/genomus-core$ bash run_tests.sh
+ TEST: Encoded Phenotypes Test
  - Test case #1: Parameter declaration -> OK after 0.028233ms
  - Test case #2: Bad Event declaration -> OK after 0.092416ms
  - Test case #3: Encoded phenotype to normalized vector -> OK after 1.1835ms
  - Test case #4: List types -> OK after 1.26773ms
- 4/4 test cases were successful after 1.27521ms

+ TEST: Decoded GenotypesTest
  - Test case #1: Functional tree build -> OK after 0.314859ms
  - Test case #2: Concat functions -> OK after 0.366906ms
  - Test case #3: Autoreferences -> OK after 0.597672ms
  - Test case #4: Lists -> OK after 0.610968ms
  - Test case #5: Random functions -> OK after 0.78173ms
- 5/5 test cases were successful after 0.788922ms

+ TEST: Parser test
  - Test case #1: Check parser tokens -> OK after 1.27272ms
  - Test case #2: Genotype to string to genotype -> OK after 2.4804ms
- 2/2 test cases were successful after 2.48854ms

+ TEST: Encoded Genotypes Test
  - Test case #1: Decoded genotype to encoded genotype -> OK after 0.083746ms
  - Test case #2: Encoding integers -> OK after 0.12721ms
  - Test case #3: Vector aproximation -> OK after 0.143969ms
  - Test case #4: Germinal vector to genotype -> OK after 0.218584ms
  - Test case #5: Normalize random vector -> OK after 1.45726ms
  - Test case #6: Normalize vector with lists -> OK after 1.52948ms
  - Test case #7: Germinal vector to expression -> OK after 178.774ms
- 7/7 test cases were successful after 178.79ms
```

Test



Test cases



GTest

```
miguel@portatil:~/TFG/genomus-core$ bash run_tests.sh
+ TEST: Encoded Phenotypes Test
  - Test case #1: Parameter declaration -> OK after 0.028233ms
  - Test case #2: Bad Event declaration -> OK after 0.092416ms
  - Test case #3: Encoded phenotype to normalized vector -> OK after 1.1835ms
  - Test case #4: List types -> OK after 1.26773ms
- 4/4 test cases were successful after 1.27521ms

+ TEST: Decoded GenotypesTest
  - Test case #1: Functional tree build -> OK after 0.314859ms
  - Test case #2: Concat functions -> OK after 0.366906ms
  - Test case #3: Autoreferences -> OK after 0.597672ms
  - Test case #4: Lists -> OK after 0.610968ms
  - Test case #5: Random functions -> OK after 0.78173ms
- 5/5 test cases were successful after 0.788922ms

+ TEST: Parser test
  - Test case #1: Check parser tokens -> OK after 1.27272ms
  - Test case #2: Genotype to string to genotype -> OK after 2.4804ms
- 2/2 test cases were successful after 2.48854ms

+ TEST: Encoded Genotypes Test
  - Test case #1: Decoded genotype to encoded genotype -> OK after 0.083746ms
  - Test case #2: Encoding integers -> OK after 0.12721ms
  - Test case #3: Vector aproximation -> OK after 0.143969ms
  - Test case #4: Germinal vector to genotype -> OK after 0.218584ms
  - Test case #5: Normalize random vector -> OK after 1.45726ms
  - Test case #6: Normalize vector with lists -> OK after 1.52948ms
  - Test case #7: Germinal vector to expression -> OK after 178.774ms
- 7/7 test cases were successful after 178.79ms
```

Test

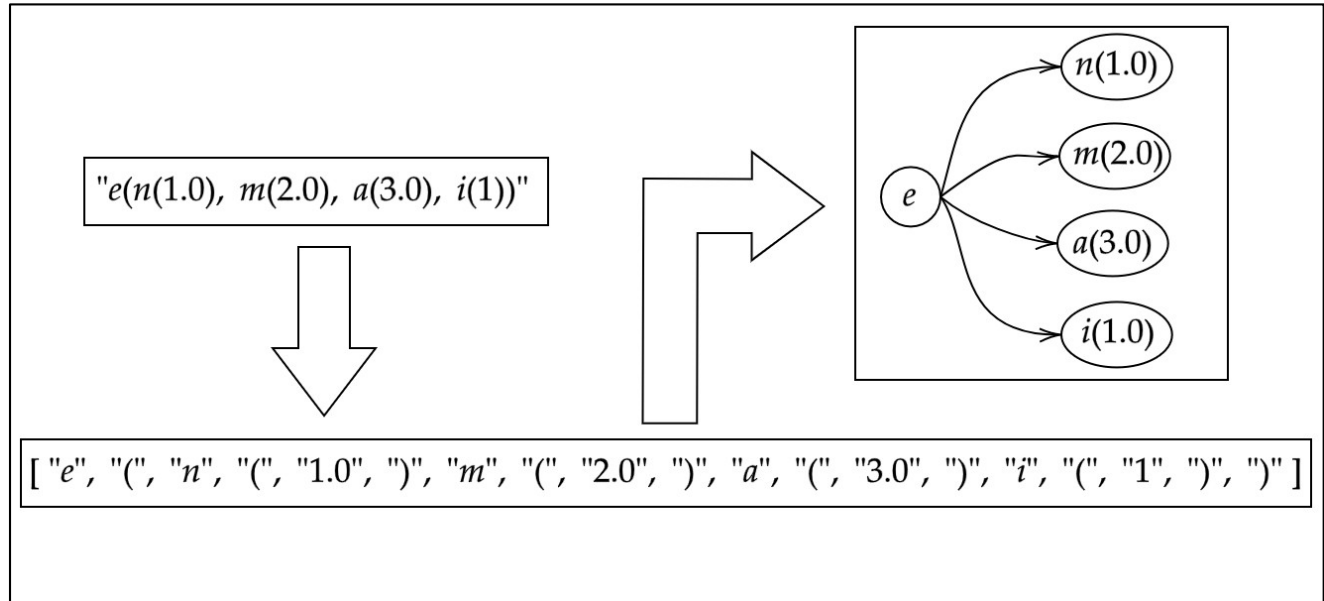
Test cases

genomus-core: Utilidades

- Parser de expresiones GenoMus (texto) a Genotipos decodificados
- Intérprete + CLI

genomus-core: Parser

- Identificación de tokens
- Análisis sintáctico
- Construcción de AST
- Instancia de GTree



genomus-core: Intérprete (CLI)

Intérprete: CLI

```
miguel@portatil:~/TFG/genomus-core$ ./build/interpreter/interpreter
```

```
GENOMUS-CORE INTERPRETER
```

```
Copyright © 2022 Miguel Pedregosa Pérez, José López-Montes
```

```
Type an expression to execute with a semi-colon at the end  
and see what happens!
```

```
> s(vPerpetuumMobile(nRnd(), lm(m(90.393967)), la(a(315.000000)), li(i(58.725177), i(54.277575))));  
{
```

```
    Encoded Genotype:
```

```
      [ 1.000000, 0.472136, 1.000000, 0.842866,  
        1.000000, 0.590537, 0.510000, 0.961290,  
        0.000000, 1.000000, 0.506578, 0.530000,  
        0.950953, 0.000000, 1.000000, 0.742646,  
        0.550000, 0.956868, 0.000000, 1.000000,  
        0.360680, 0.560000, 0.772332, 0.560000,  
        0.645396, 0.000000, 0.000000, 0.000000 ]
```

```
    Decoded Genotype: s(vPerpetuumMobile(nRnd(), lm(m(90.393967)), la(a(315.000000)),  
    li(i(58.725178), i(54.277576))))
```

```
    Encoded Phenotype (human readable): s(v(e(nRnd(0.961290), 0.950953, 0.956868, 0.772332),  
    e(nRnd(0.961290), 0.950953, 0.956868, 0.645396)))
```

```
    Encoded Phenotype (normalized vector): [ 0.618034, 0.236068, 0.961290, 0.950953,  
      0.956868, 0.772332, 0.961290, 0.950953,  
      0.956868, 0.645396 ]
```

```
}
```

```
> s(vPerpetuumMobile(nRnd(), lm(m(90.393967)), la(a(315.000000)), li(i(58.725177), i(54.277575))));
```

```
ERROR: BAD_PARSER_ENTRY_BAD_PARENTHESES
```

```
> s(asdf(nRnd(), lm(m(90.393967)), la(a(315.000000)), li(i(58.725177), i(54.277575))));
```

```
ERROR: BAD_PARSER_ENTRY_BAD_FUNCTION_NAME: asdf
```

```
> \q;
```

```
miguel@portatil:~/TFG/genomus-core$ █
```

Intérprete: CLI

```
miguel@portatil:~/TFG/genomus-core$ ./build/interpreter/interpreter
```

```
GENOMUS-CORE INTERPRETER
```

```
Copyright © 2022 Miguel Pedregosa Pérez, José López-Montes
```

```
Type an expression to execute with a semi-colon at the end  
and see what happens!
```

```
> s(vPerpetuumMobile(nRnd(), lm(m(90.393967)), la(a(315.000000)), li(i(58.725177), i(54.277575))));
```

```
{
```

```
  Encoded Genotype:
```

```
    [ 1.000000, 0.472136, 1.000000, 0.842866,  
      1.000000, 0.590537, 0.510000, 0.961290,  
      0.000000, 1.000000, 0.506578, 0.530000,  
      0.950953, 0.000000, 1.000000, 0.742646,  
      0.550000, 0.956868, 0.000000, 1.000000,  
      0.360680, 0.560000, 0.772332, 0.560000,  
      0.645396, 0.000000, 0.000000, 0.000000 ]
```

```
  Decoded Genotype: s(vPerpetuumMobile(nRnd(), lm(m(90.393967)), la(a(315.000000)),  
li(i(58.725178), i(54.277576))))
```

```
  Encoded Phenotype (human readable): s(v(e(nRnd(0.961290), 0.950953, 0.956868, 0.772332),  
e(nRnd(0.961290), 0.950953, 0.956868, 0.645396)))
```

```
  Encoded Phenotype (normalized vector): [ 0.618034, 0.236068, 0.961290, 0.950953,  
      0.956868, 0.772332, 0.961290, 0.950953,  
      0.956868, 0.645396 ]
```

```
}
```

```
> s(vPerpetuumMobile(nRnd(), lm(m(90.393967)), la(a(315.000000)), li(i(58.725177), i(54.277575)));
```

```
ERROR: BAD_PARSER_ENTRY_BAD_PARENTHESES
```

```
> s(asdf(nRnd(), lm(m(90.393967)), la(a(315.000000)), li(i(58.725177), i(54.277575)));
```

```
);
```

```
ERROR: BAD_PARSER_ENTRY_BAD_FUNCTION_NAME: asdf
```

```
> \q;
```

```
miguel@portatil:~/TFG/genomus-core$
```

Mensaje de bienvenida

Entrada de usuario

Genotipo evaluado

Entrada de usuario

Error de sintaxis

Entrada de usuario

Error de semántica

Terminación de la sesión

genomus-core-js

- Bindings para NodeJS de genomus-core
- Interfaz en TypeScript
- Publicado en el registro público de NPM
- Construido con Node-GYP
- Node-API para enlazar con V8

genomus-core-js

0.1.0 • Public • Published 6 days ago



Readme



Explore

BETA



1 Dependency



0 Dependents



1 Versions



Settings

Resultados

Benchmark

- *Instancia de cómputo a realizar secuencialmente durante 10 segundos:*
 - *Generación de un vector germinal aleatorio*
 - *Normalización del vector*
 - *Transformación vector \rightarrow expresión*
 - *Parseo de la expresión en Genotipo*
 - *Evaluación del Genotipo*

Resultados

Ref: p60, 62, 63

Benchmark

t (ms)	i	t_avg (ms)	t (ms)	i	t_avg (ms)
10633	48	221,52	10014	176	56,90
10194	62	164,42	10056	210	47,89
10554	79	133,59	10436	284	36,75
13221	46	287,41	11393	160	71,21
14195	57	249,04	11452	110	104,11
20244	12	1687	10080	216	46,67
10011	39	256,69	10081	288	35,00
10937	56	195,30	12145	259	46,89
99989	399	250,60	85657	1703	50,30

(a) GenoMus

(b) genomus-core

t: tiempo de ejecución

i: iteraciones completadas

t_avg: tiempo medio de iteración

Tabla 7.3: Resultados del benchmark.

Resultados

Benchmark

*Genomus-core se muestra
unas cinco veces más
rápido que el prototipo*

t: tiempo de ejecución

i: iteraciones completadas

t_avg: tiempo medio de iteración

t (ms)	i	t_avg (ms)
10633	48	221,52
10194	62	164,42
10554	79	133,59
13221	46	287,41
14195	57	249,04
20244	12	1687
10011	39	256,69
10937	56	195,30
99989	399	250,60

(a) GenoMus

t (ms)	i	t_avg (ms)
10014	176	56,90
10056	210	47,89
10436	284	36,75
11393	160	71,21
11452	110	104,11
10080	216	46,67
10081	288	35,00
12145	259	46,89
85657	1703	50,30

(b) genomus-core

Tabla 7.3: Resultados del benchmark.

Preguntas?