Проектирование визуального функционального языка с возможностями оптимизации рекурсии

Завьялов Антон Алексеевич

Научный руководитель:
доцент, к. ф.-м. н. Старолетов Сергей Михайлович

russian seminar on Software Engineering, Theory and Experimental Programming

September 9, 2021

О докладчике

- Образование:
  - Бакалавр, направление подготовки «Программная инженерия», 2017 - 2021, АлтГТУ им. И.И. Ползунова
  - Магистрант направления подготовки «Математика и компьютерные науки», 2021 - н.в., НГУ

- Интересы: теоретическое и прикладное программирование, языки программирования, методы (оптимизирующей) трансляции

План доклада

1. Обзор основных понятий и концепций
2. Демонстрация работы программной системы
3. Рассмотрение деталей проектирования и реализации

Цели и задачи

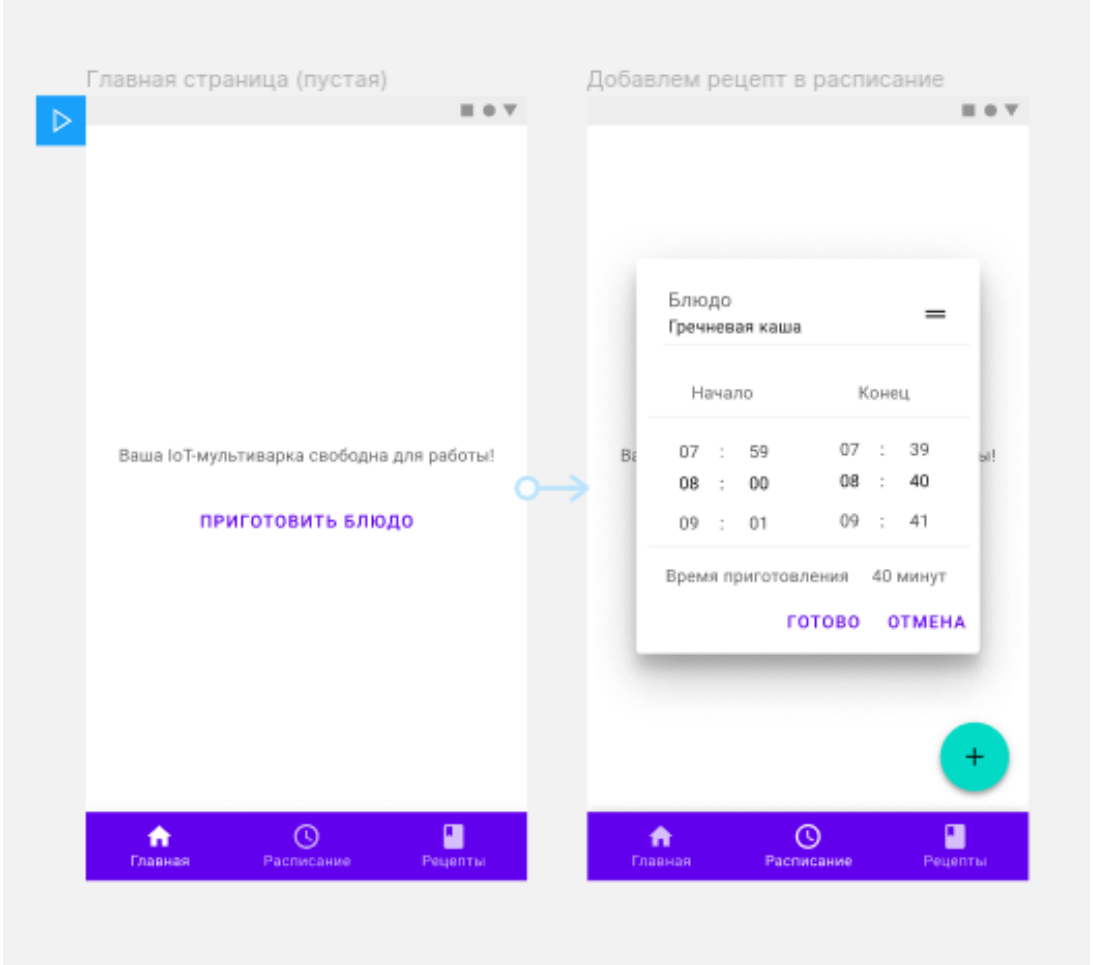
Мотивация

Ограниченность существующих инструментов прототипирования

Задачи, решаемые в рамках работы

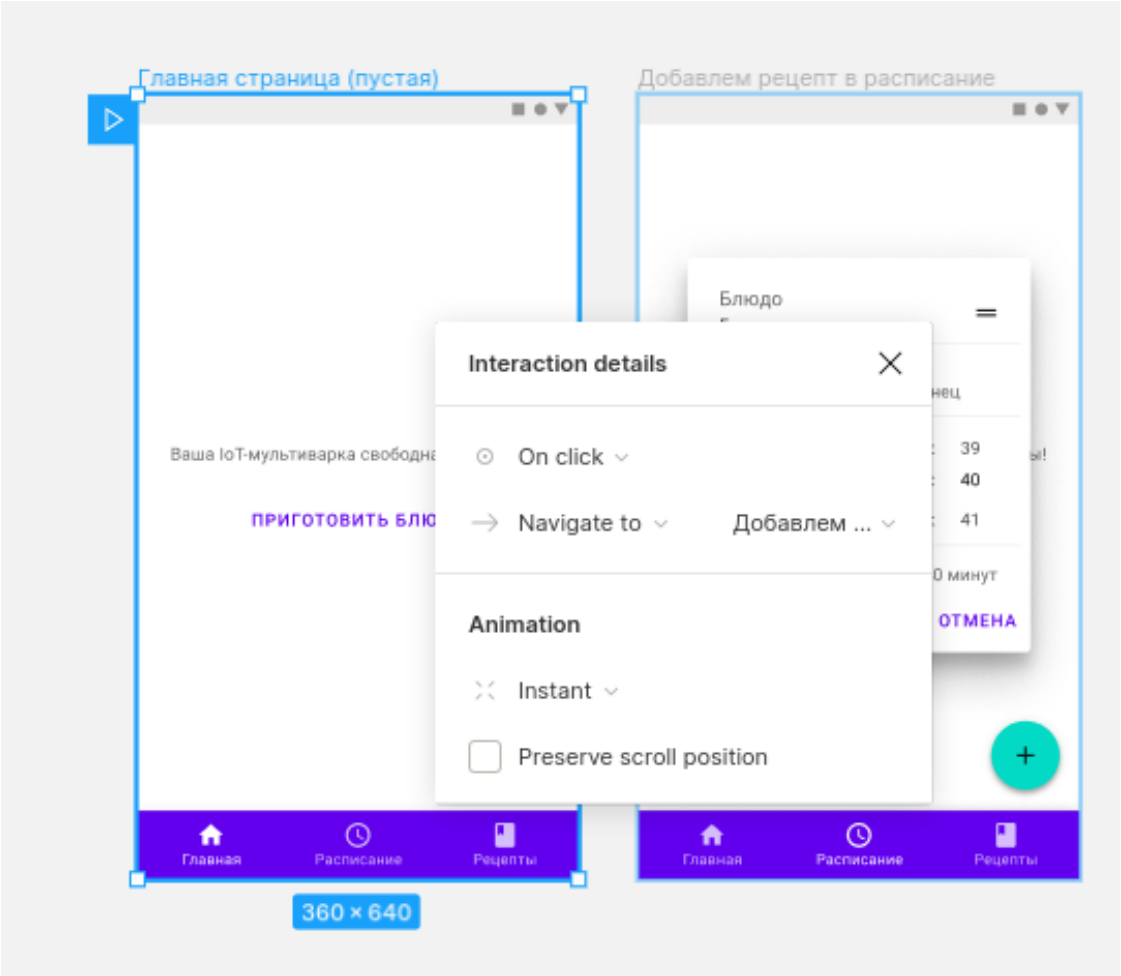
- Проектирование визуального языка функционального программирования
- Проектирование и разработка интегрированной среды визуального программирования на ранее спроектированном языке
- Проектирование и разработка оптимизирующего компилятора программ на спроектированном языке
- Анализ технологий и алгоритмов автоматической оптимизации рекурсии в трансляторах

Мотивация



Интерактивное прототипирование в Figma ограничивается переходами между кадрами.

Мотивация



Можно выбрать лишь инициатора события и направление перехода

Терминология

Визуальное программирование

Способ создания программы для ЭВМ путём манипулирования графическими объектами вместо написания её текста.

Функциональное программирование

Парадигма программирования, в которой вычислительный процесс представлен получением значений функций.

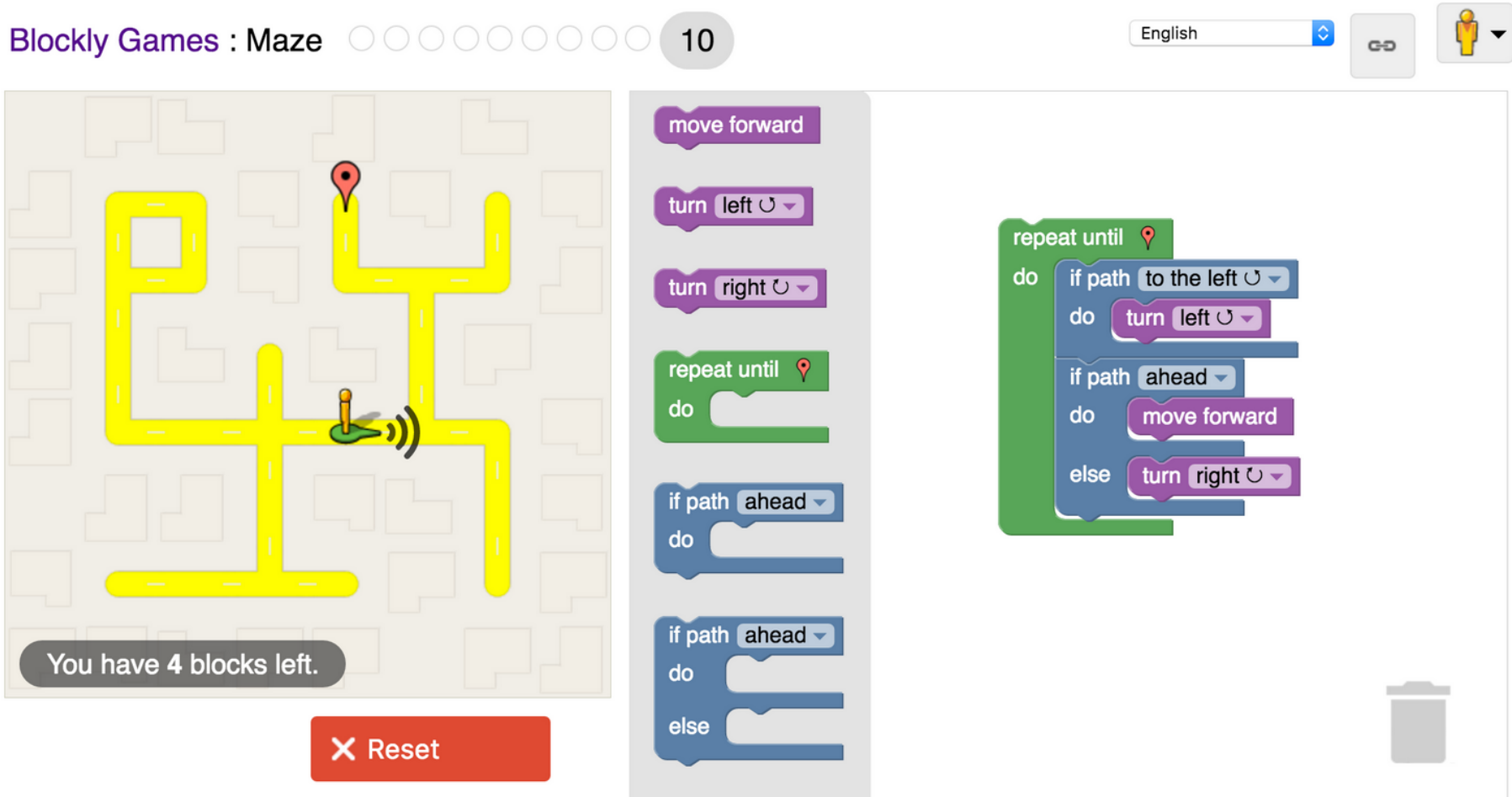
Flovver

В работе представлен язык функционального программирования Flovver.

Визуальные языки

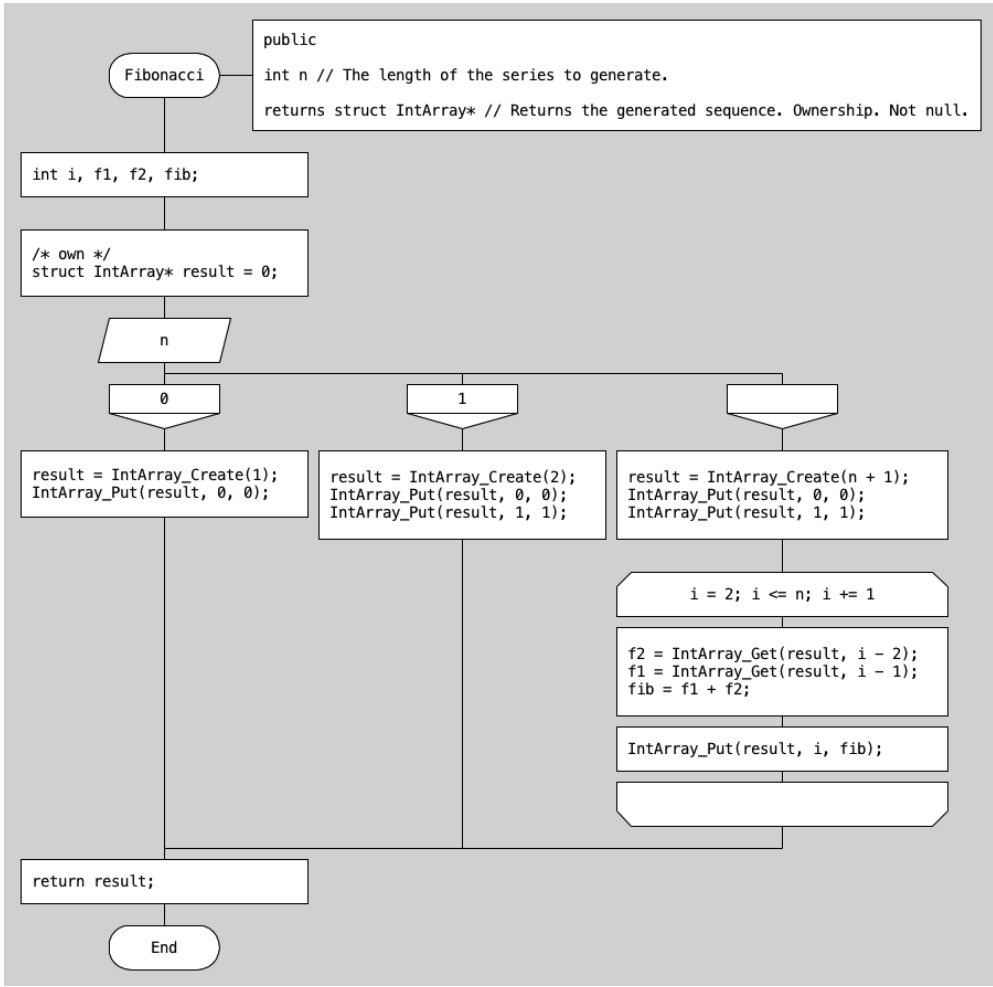
- Блочные (Scratch, Blockly)
- Диаграммные (ДРАКОН)
- На основе потоков данных (Quartz Composer)
- Функциональные (Enso, Flovver, ??)

Блочные визуальные языки



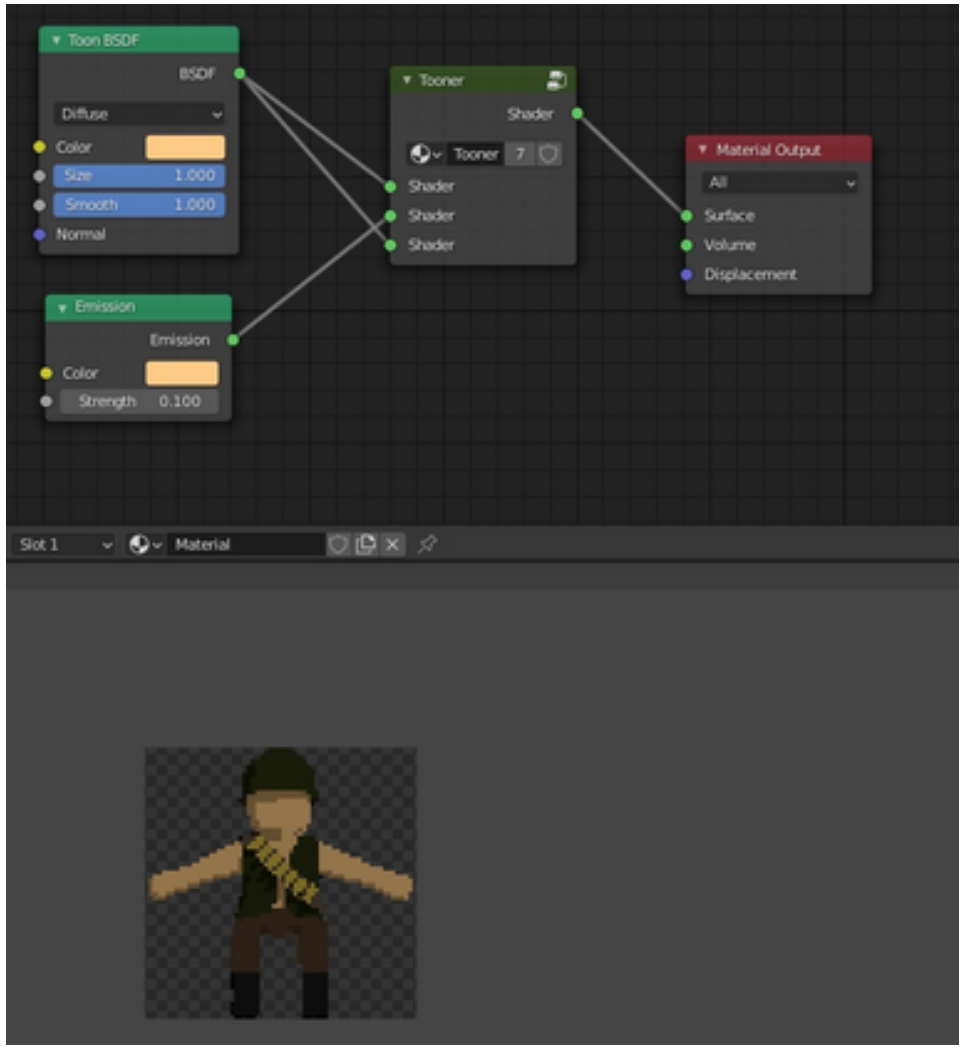
Язык программирования Blockly

Диаграммные языки



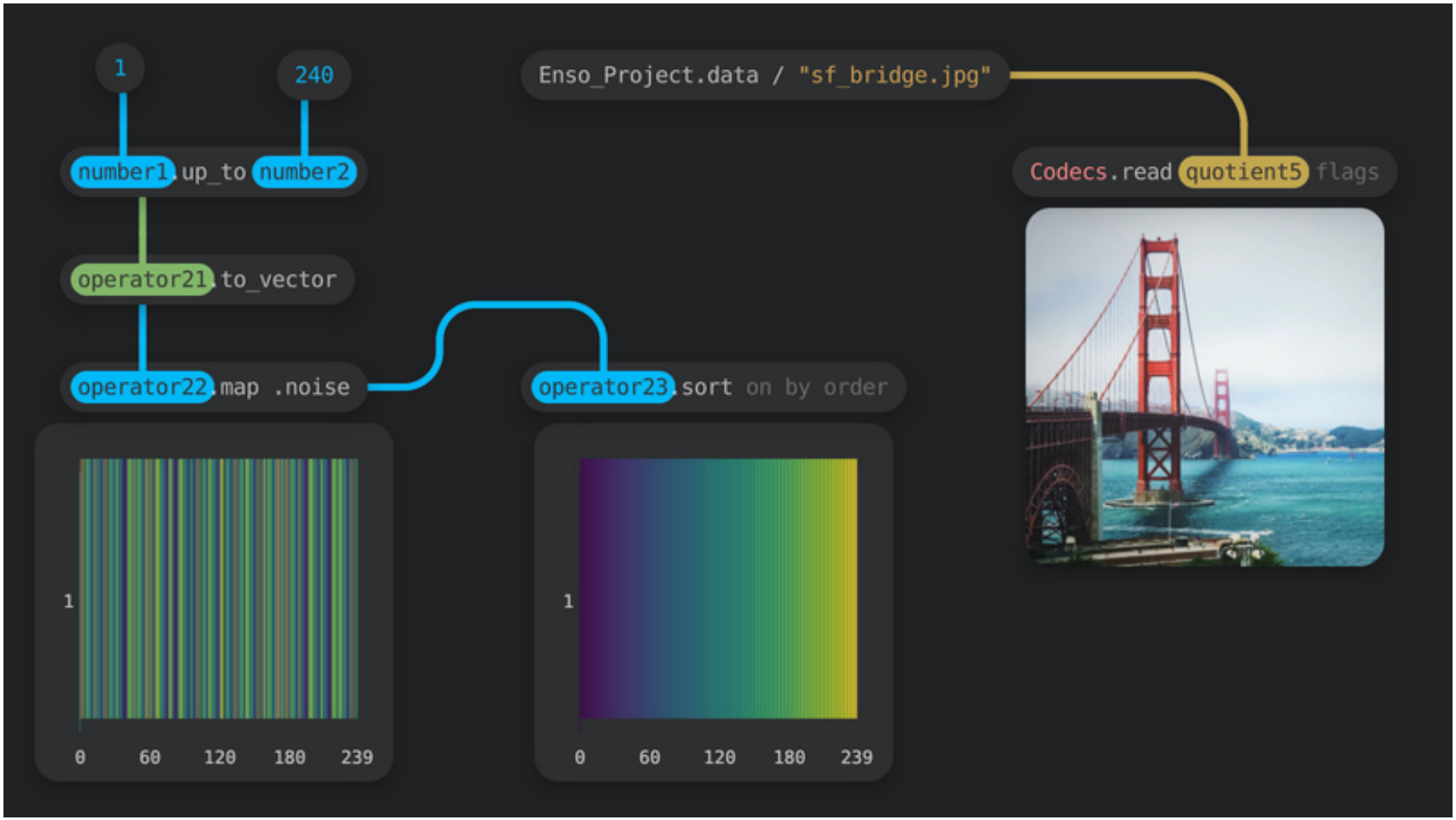
Язык программирования ДРАКОН

Языки программирования потоков данных



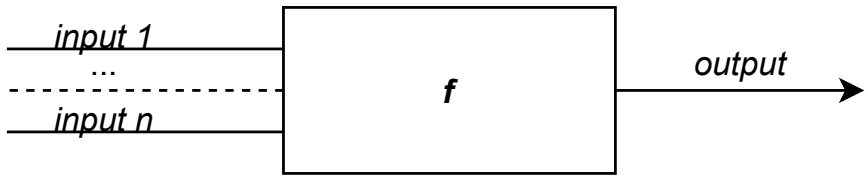
Настройка материалов в Blender

Функциональные визуальные языки

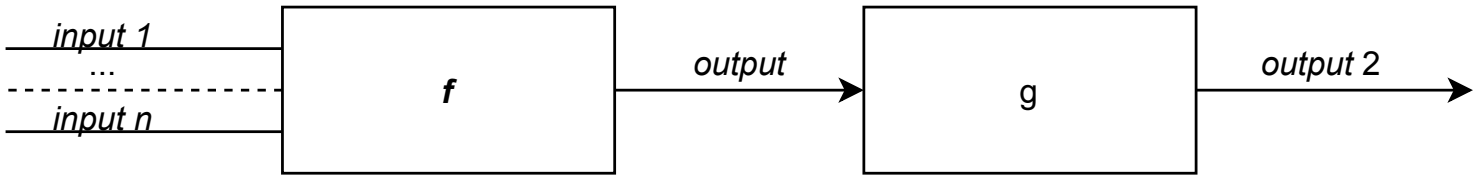


Язык программирования Enso

Flovver – функции



Терм $\lambda x_1 ... x_n \ . \ f(x_1, ..., x_n)$ *в Flovver*



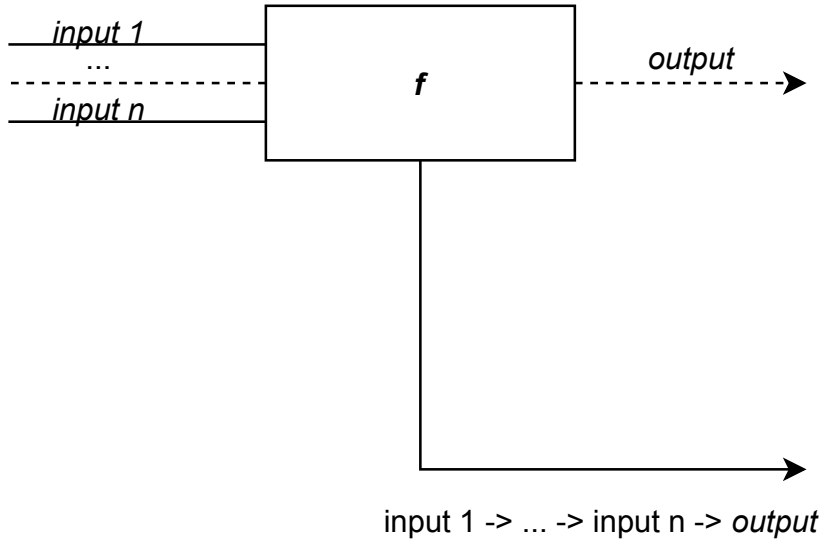
Представление функции $g(f(x_1, ..., x_n))$ *в Flovver (1)*

Семантика конструкции (1)

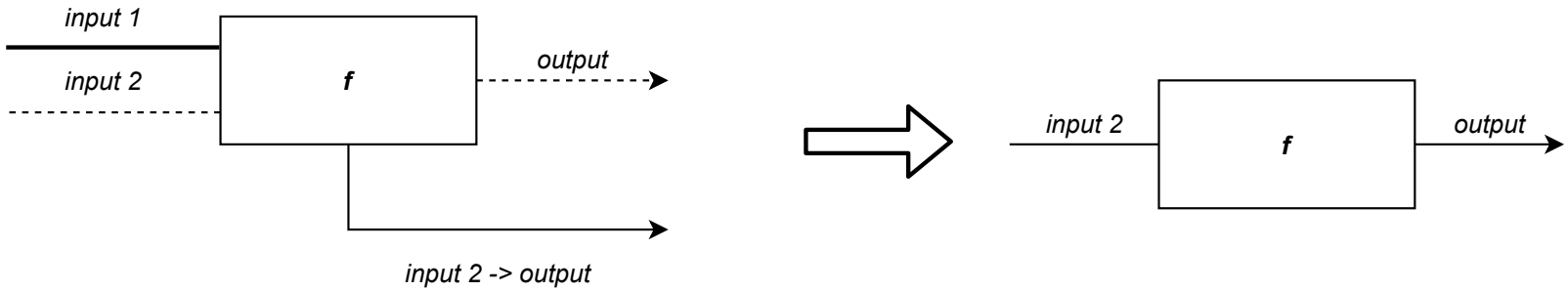
$f' := ((((\lambda x_1 ... x_n \ . \ f(x_1, ..., x_n))v_1)...)v_n)$

$g' := ((\lambda x \ . \ g(x))f')$

Flovver – частичное применение

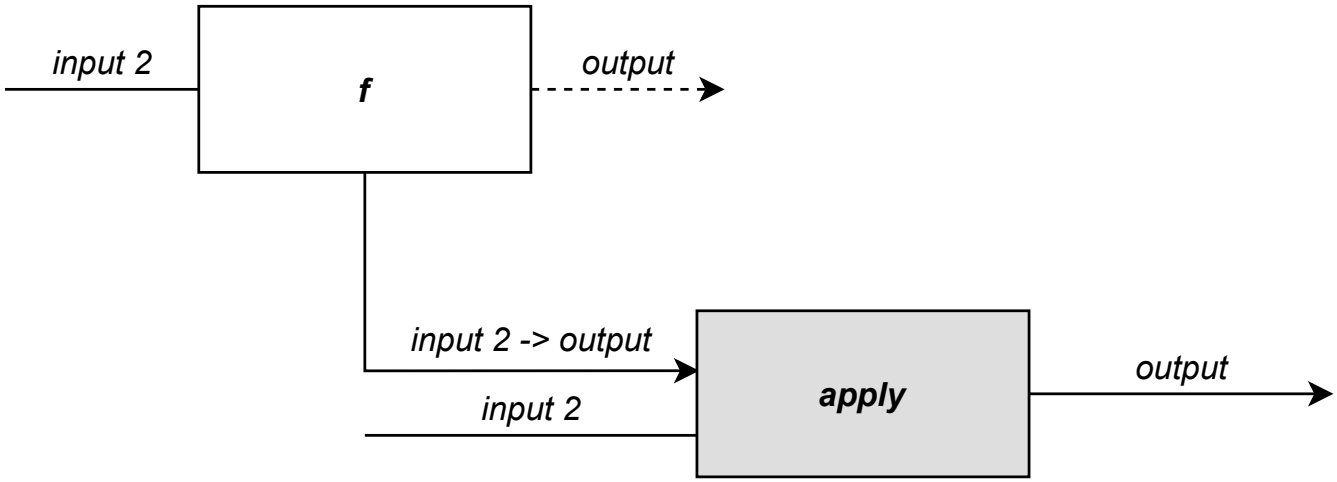


Дуга из нижней стороны прямоугольника - передача функции в качестве значения

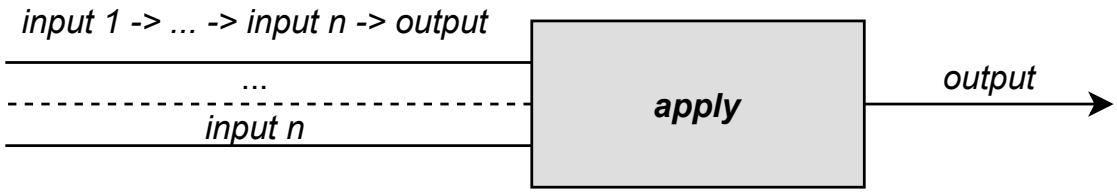


В результате получим функцию от меньшего числа аргументов

Flovver – комбинатор apply

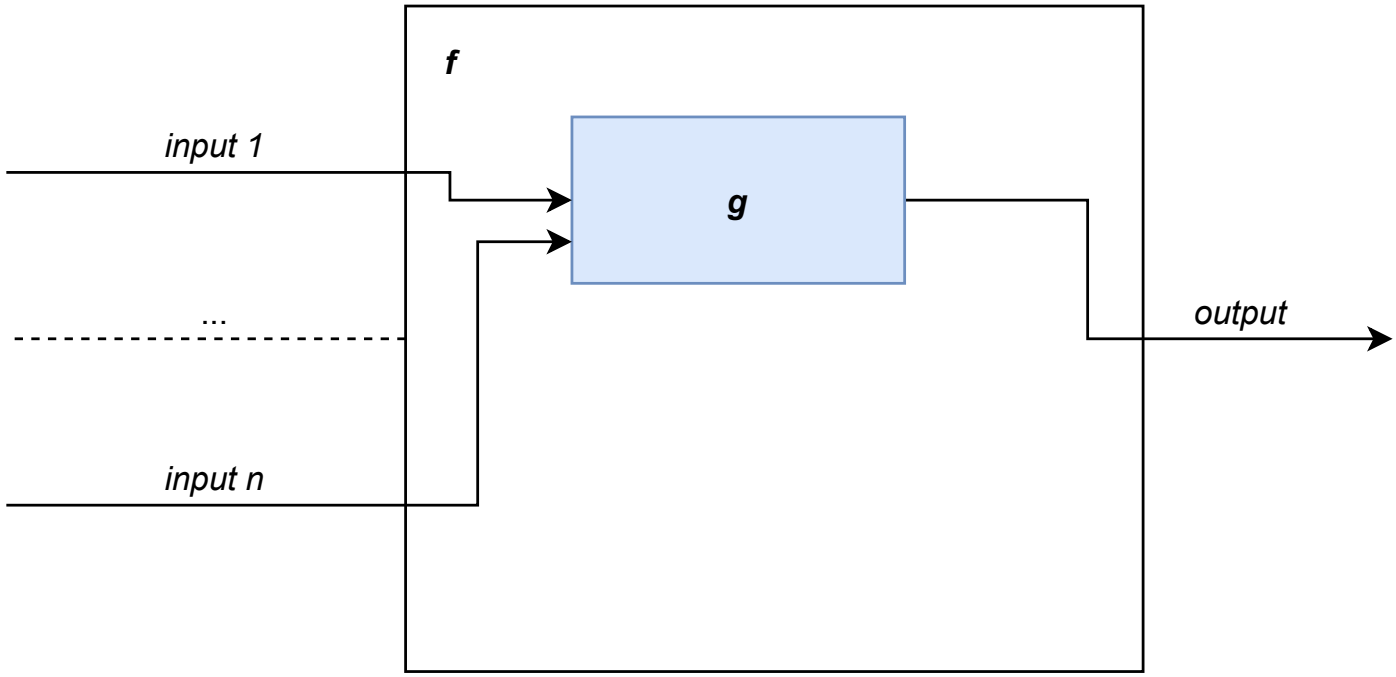


Чтобы вычислить функцию, передаваемую как значение, нужно использовать специальную функцию



Сигнатура функции apply

Flovver – построение новых функций

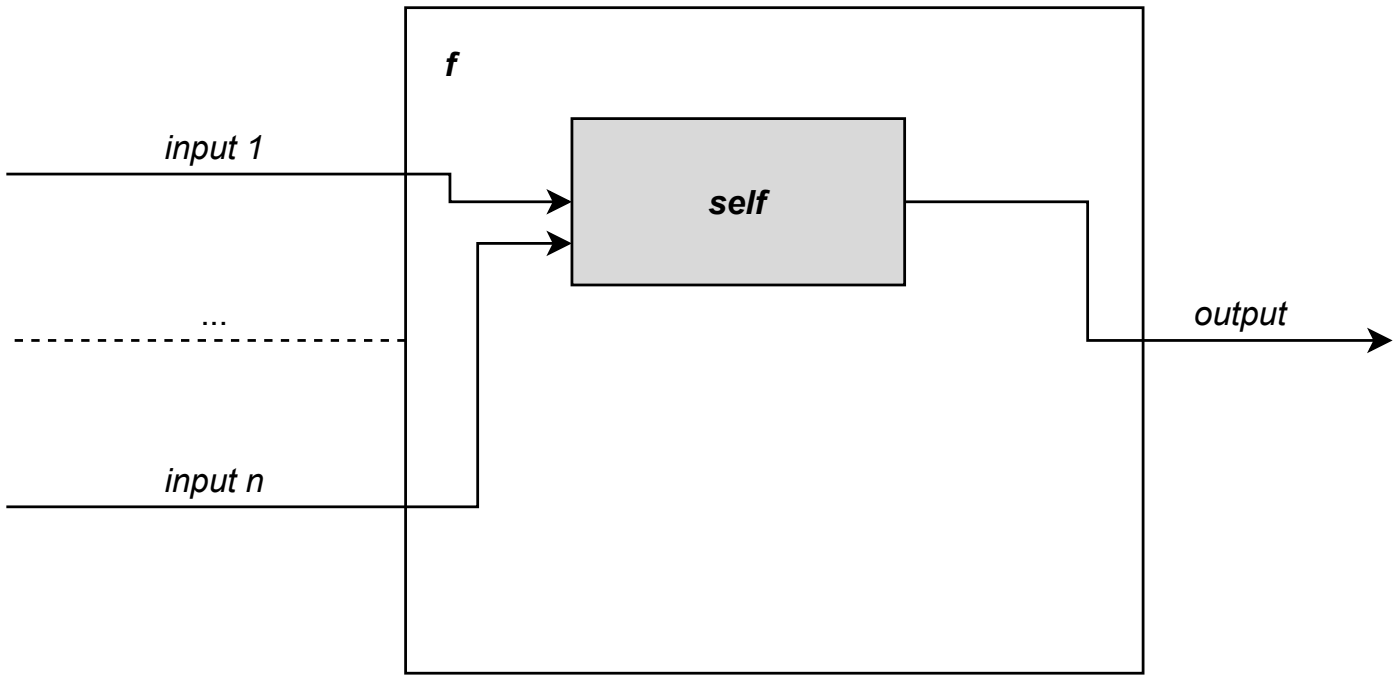


Определение новой функции (2)

Семантика конструкции (2)

$f := \lambda x_1 ... x_n \ . \ g(x_1, ..., x_n)$
В общем случае:
$fun := \lambda x_1 ... x_n \ . \ T$,
где T — некоторый зависимый от $x_1, ..., x_n$ терм.

Flovver - поддержка рекурсии



Специальная функция self (3)

Семантика конструкции (3)

Можно считать, что рекурсивные определения в Flovver применены к комбинатору неподвижной точки.

Y-комбинатор

Реализация аналогичной семантики в Racket с помощью макросов

```
#lang racket

;; https://rosettacode.org/wiki/Y_combinator#Scheme
(define Y
  (lambda (f)
    ((lambda (g) (g g))
     (lambda (g)
       (f  (lambda a (apply (g g) a)))))))

;; Macro emulating Flouver self semantics
(define-syntax-rule (combine self args f)
  (Y (lambda (self) (lambda args f))))
```

Y-комбинатор

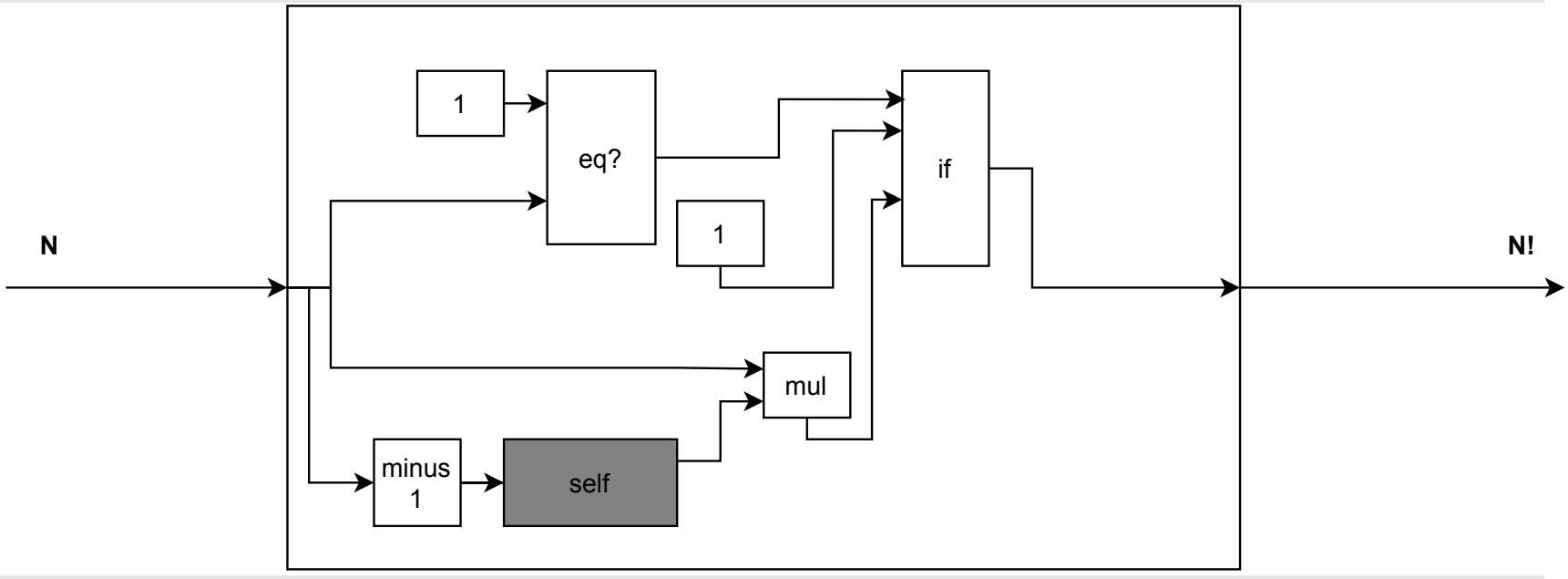
Пример использования макроса

```
;; Factorial written with ‘self‘
(define fac
  (combine self (x)
           (if (< x 2)
               1
               (* x (self (- x 1))))))

;; Fibonacci function written with ‘self‘
(define fib
  (combine self (x)
           (if (< x 2)
               x
               (+ (self (- x 1)) (self (- x 2))))))

(fac 10)
(fib 5)
```

Пример программы на Flovver

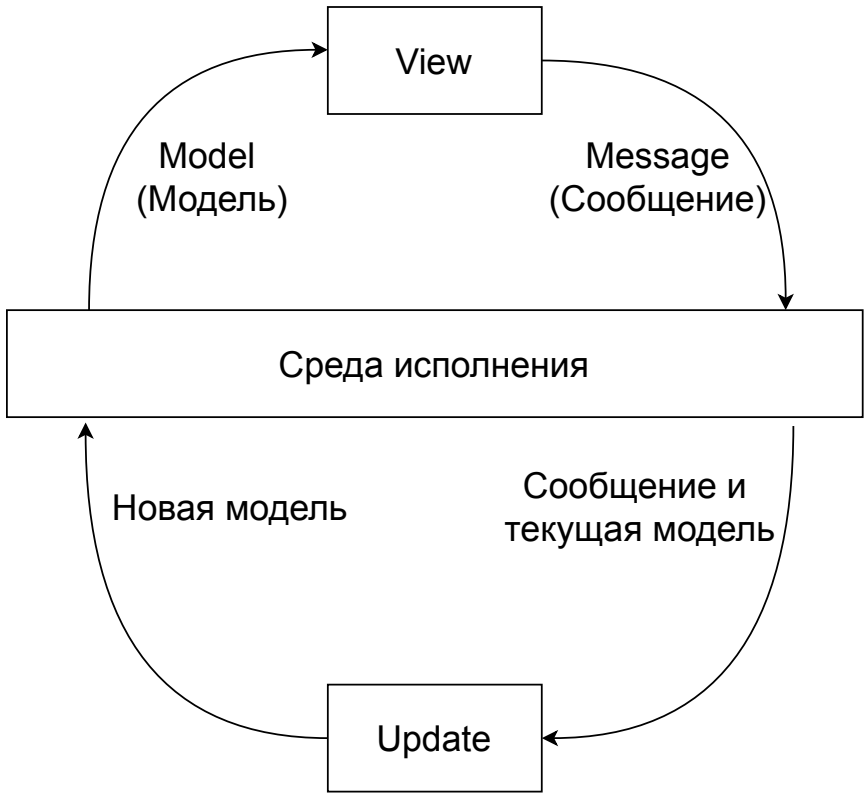
$$n! = \begin{cases} 1 & n = 0, \\ n \cdot (n-1)! & n > 0. \end{cases} \tag{1}$$

Факториал на Flovver



Описание факториала на языке Flovver

Архитектура Elm



Жизненный цикл приложения, использующего Elm-архитектуру

1. На основе модели в *View* отрисовывается интерфейс;
2. При пользовательском действии, в *Update* посылается сообщение;
3. В функции *Update* обрабатывается текущая модель данных программы и посланное сообщение, функция возвращает новую модель, инициируется перерисовка *View*.

Архитектура Elm – модель

Тип-модель:

```
Model of Int
```

Тип-сообщение:

```
Msg of Variant

NewInput of String
ComputeFactorial of Unit
```

Типы программы «Факториал»

Variants

1 name NewInput type String

2 name ComputeF type Unit

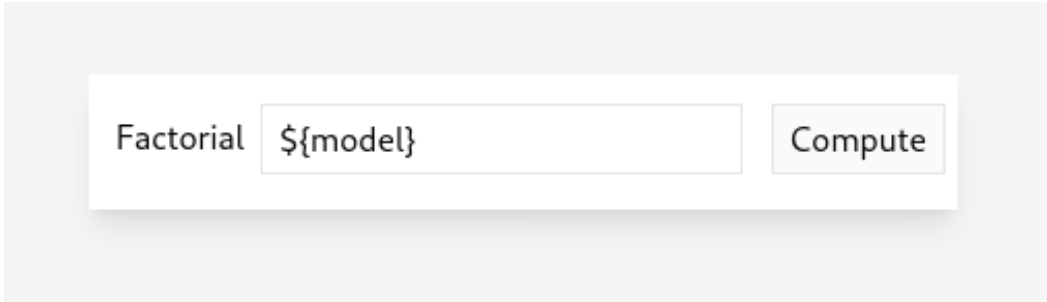
\+

Role

Set as Model | Set as Message

Управление типами в редакторе

Архитектура Elm – представление

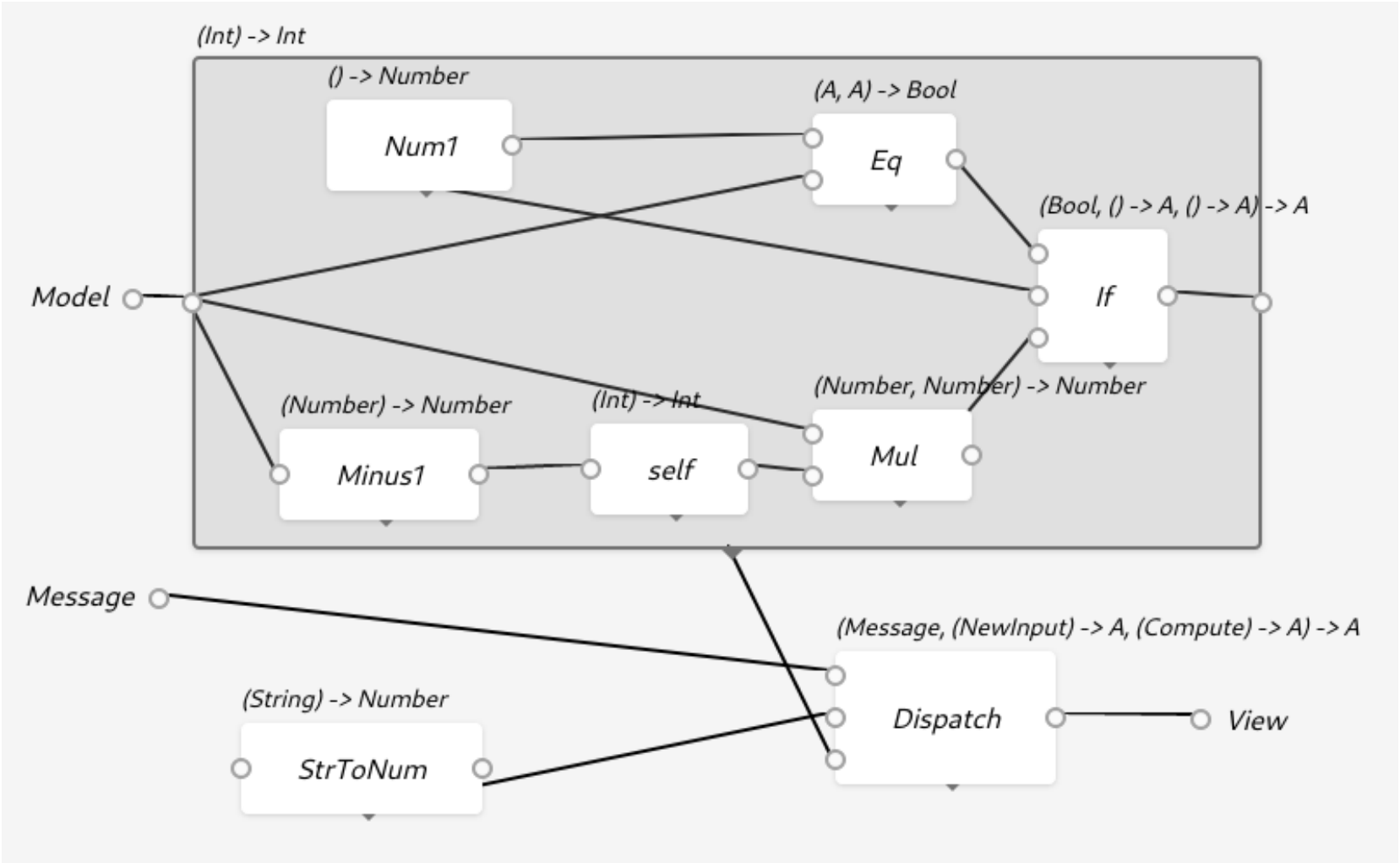


Интерфейс программы «Факториал»

```
Events

onclick  signal({ tag: 'Compute' })
```

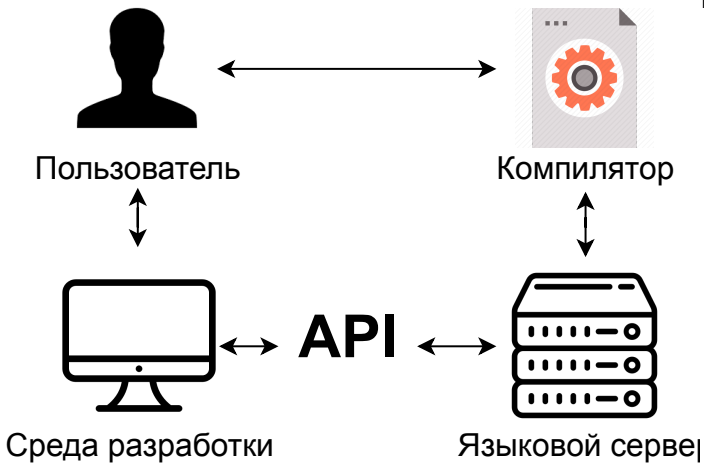
Определение пользовательских действий в редакторе

Архитектура Elm – логика программы

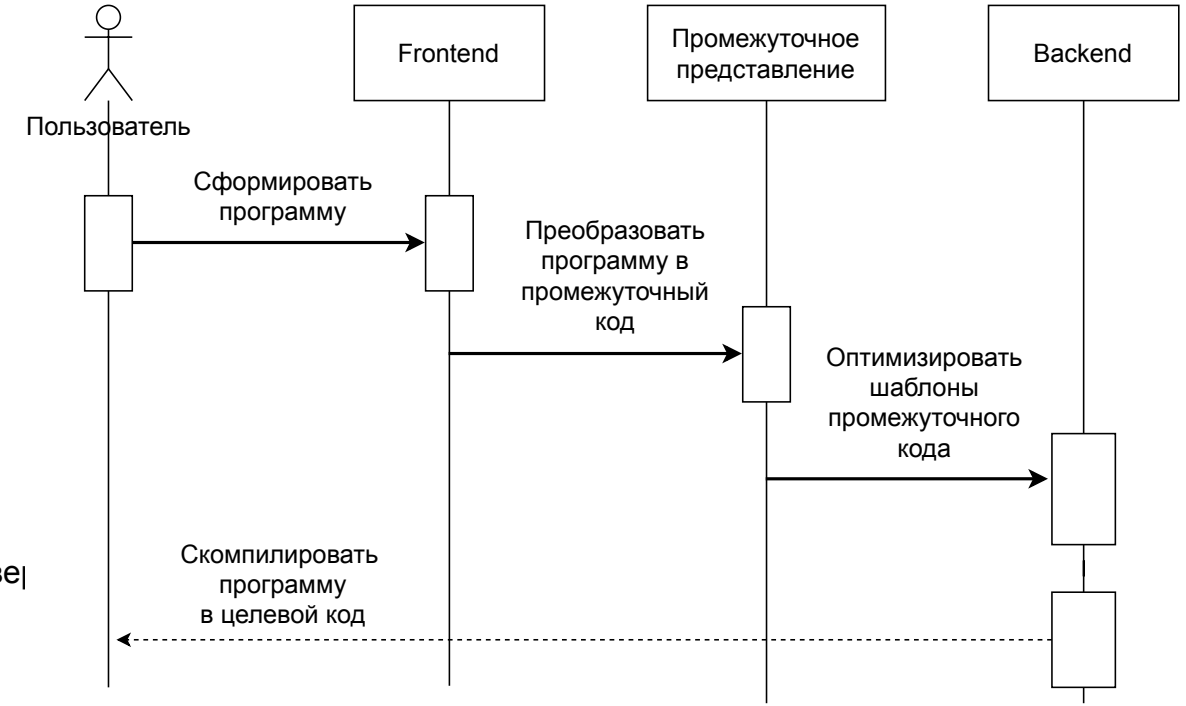


Программа «Факториал» на Flovver

Архитектура разрабатываемого ПО

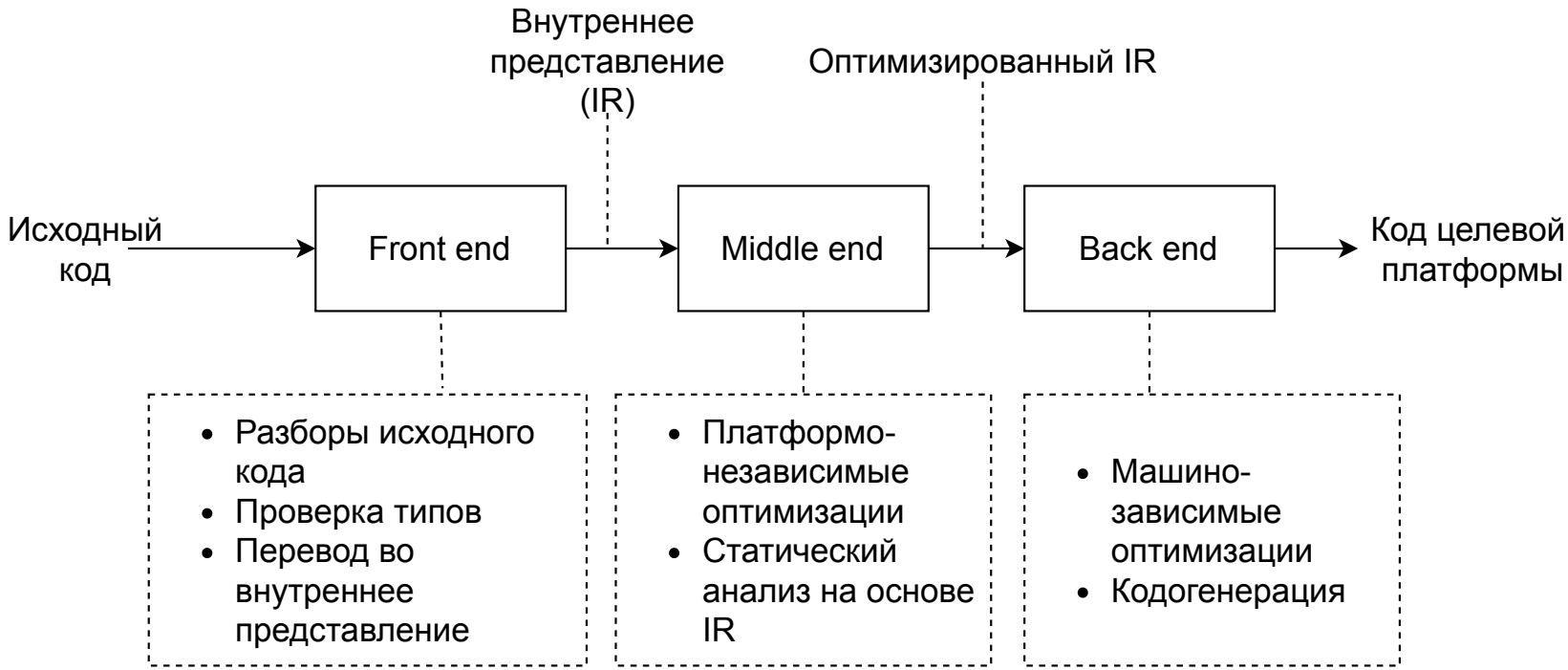


Общая архитектура разрабатываемого ПО



Взаимодействие между пользователем и модулями

Многопроходный оптимизирующий компилятор



Структура многопроходного компилятора

Внутреннее представление программы в компиляторе

Промежуточное представление (англ. - intermediate representation)

Структура данных, используемая внутри компилятора для представления исходного кода программы.

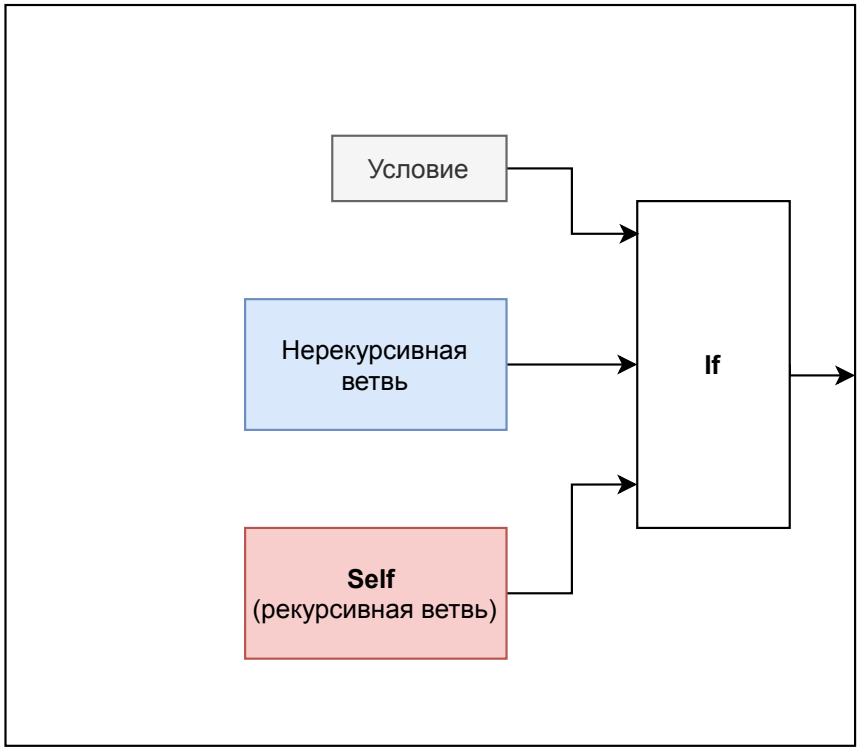
Flovver использует графовое внутреннее представление, т.е.:

- два набора данных — вершины и связи между ними
- связи могут быть внешними и внутренними; "по значению"и "по имени"
- вершины отражают применения, определения функций; рекурсивные вызовы

Плюсы Flovver IR:

- однозначно отображает структуру программы
- удобно представляет данные для потокового анализа
- быстро перестраивается в форму, удобную для трансляции

Устранение хвостовой рекурсии - 1



Часто встречаемый шаблон хвостовой рекурсии

Алгоритм устранения хвостовой рекурсии (в первом приближении):

1. Пометить все функции, которые возвращают шаблон хвостовой рекурсии как хвосто-рекурсивные;
2. На этапе кодогенерации для таких функций генерировать в теле конструкцию while (...) { ... }, в которой менять параметр и переменную-аккумулятор, не генерировать хвостовой рекурсивный вызов.

Устранение хвостовой рекурсии - 2

Как определить шаблон хвостовой рекурсии?

Понятие правильных хвостовых вызовов формализовано в работе William Clinger. Proper tail recursion and space efficiency. Специфицировано в отчётах к *хвосто-рекурсивному* языку Scheme (R6RS: 5.11, 11.20).

- Контекст хвостовой рекурсии — место, в котором рекурсивный вызов **гарантированно** хвостовой.
- В рамках работы интересуют два контекста — конец функции, условное выражение в конце функции.

Almost tail expression grammar

$< almost\ tail\ expr > \rightarrow < rec.\ call > \mid < if > \mid < expr >$

$< if > \rightarrow if < expr >< almost\ tail\ expr >< almost\ tail\ expr >$

Все $< rec.call >$ **в дереве выражений, порожденном грамматикой, являются хвостовыми.**

Оптимизация рекурсии общего вида

Мемоизация

Сохранение результатов выполнения функций для предотвращения повторных вычислений.

В Flovver можем мемоизировать любые вычисления, т.к. язык чисто-функциональный.

Пример (сгенерированный код **до** и **после** оптимизации)

```
const f = (x, y) => {
    hardComputations(x, y);
}
```

```
const f = (() => {
  const f_st = {};
  const f_w = (x, y) => {
   hardComputations(x, y);
  }
return (x, y) =>
  f_st[[x, y]] =
    f_st[[x, y]] ||
    f_w(x, y);
})();
```

Подготовка к кодогенерации – укладка IR

Возникает необходимость упорядочить вершины графа IR в такой последовательности, что определения значений в ней идут раньше, чем их использования в функциях.

Граф зависимостей

Ориентированный граф, связи в котором представляют зависимости между объектами.

- Граф IR является графом зависимостей (по значениям)
- Граф IR – ориентированный ациклический граф (*англ.* – *direct-acyclic graph (DAG)*)
- Для DAG можно упорядочить вершины по следованию: *источник* $\rightarrow v_1 \rightarrow \ldots \rightarrow$ *сток* (топологическая сортировка)
- Упорядоченный граф легко транслировать в целевой код платформы

SSA-форма (value numbering)

Статическое одиночное присваивание

Промежуточное представление, используемое компиляторами, в котором каждой переменной значение присваивается лишь единожды.

Код до SSA

y := 1
y := 2
x := y

Код, приведенный к SSA

y1 := 1
y2 := 2
x1 := y2

Трансляция в код целевой платформы

Перевод внутреннего представления в код проходит в три шага:

1. Граф программы упорядочивается топологической сортировкой
2. Каждому узлу в порядке сортировки присваивается SSA-переменная:
   - Без входных параметров — получение простого значения
   - Блоку с входными параметрами — вычисление значения функции
   - Функциональному блоку (блоку с дугой из нижней части прямоугольника и описанию функции) — функция
3. По именам и номерам SSA-переменных легко генерируется код целевой платформы (в случае Flovver — *JavaScript*)

Итоги

- Спроектирован визуальный язык программирования
- Разработана визуальная среда программирования на языке с использованием паттерна программирования «Elm-архитектура»
- Разработан многопроходный компилятор языка с возможностью оптимизации хвостовой рекурсии, оптимизации общей рекурсии путём мемоизации
- Тезисы о проектировании системы опубликованы в материалах Всероссийской научно-технической конференции «Наука и молодежь – 2021»

Направления дальнейшего развития

- Формализация языка с точки зрения теории графических и функциональных языков (Arrow category correspondence ??)
- Введение статической типизации и механизма вывода типов
- Исследование общих рекурсивных шаблонов путём анализа структуры больших программных систем, использующих функциональные языки (реализация и стандартная библиотека Haskell, Scheme, OCaml); императивные (Java, C++, ??)

Ссылки

- `https://github.com/flovver/` — все репозитории, связанные с языком
- `http://www.r6rs.org/` — 6 редакция спецификации *хвосто-рекурсивного* языка Scheme
- `https://enso.org/` — один из немногих графических функциональных языков, претендующих на промышленный уровень
- `https://ruhaskell.org/posts/elm/2015/03/06/elm-architecture.html` — обзор Elm-архитектуры
- `https://xavierleroy.org/mpri/2-4/fir.2up.pdf` — хороший разбор IR компиляторов от Xavier Leroy

Контакты

- Электронная почта (личная): a.zavyalov.98@yandex.ru
- Электронная почта (университетская): a.zavyalov@g.nsu.ru
- GitHub: `https://github.com/andiogenes`