НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ

Курсовая работа

Тема: «Приложение для перевода функций fptl в сетевое представление и обратно»

Группа: А-13м-23

Студент: Грищенков Г.С.

Руководитель: Кутепов В.П.

Москва 2024

Содержание

1. ВВЕДЕНИЕ

Язык FPTL [1, 2] построен на композиционной основе, он достаточно прост в освоении, поскольку сохраняет общепринятую форму задания функций посредством использования четырех простых операций композиции функций и рекурсивных определений, задаваемых в виде систем функциональных уравнений. Он строго типизирован и позволяет однообразно, как и функции, определять абстрактные типы данных в виде систем уравнений. FPTL самодостаточен в том смысле, что не требует введения, как в LISP, множества базисных функций; они непосредственно извлекаются из определения типов данных аргументов и результатов функций, как конструкторы этих данных и обратные к ним деструкторы. Кроме того, в FPTL в качестве базисных функций можно использовать арифметические и другие уже реализованные в компьютере функции. Но самое важное с точки зрения построения параллельных программ состоит в том, что для задания параллелизма в них не требуется использование каких либо специальных примитивов и он автоматически легко распознается при выполнении программы. Это является следствием того, что три из четырех операций композиции функций являются параллельными. Эти особенности FPTL создали основу для разработки качественных параллельных программ, не прибегая к применению других средств задания параллелизма в программе [1]

2. Теоретическая основа языка FPTL

Формально программы и данные FPTL определяются как композиционные структуры, определяемые на множестве «исходных» функций и данных путем применения к ним определенных операций композиции и рекурсивных определений.

Пусть $F = \{f\_i\hat{}((m\_i, n\_i)) \mid i = 1,2, \dots\}$ $P = \{p\_i\hat{}((m\_i, 1)) \mid i = 1,2, \dots\}$ - множество исходных функций и предикатов указанных арностей $((mi, ni), (mi, 1))$, где mi – длина кортежа (упорядоченного множества) аргументов функции или предиката, ni – длина кортежа выходных значений.

Интерпретация:

$f_i^{(m_i,n_i)}: D_1 \times D_2 \times \ \ldots \ \times D_{m_i} \to D'_1 \times D'_2 \times \ \ldots \ \times D'_{n_i}$ (однозначное преобразование);

$$p_i^{(m_i,1)}: D_1 \times D_2 \times \ \ldots \ \times D_{m_i} \to \{true, false\}$$

Для mi = 0 функции и предикаты интерпретируются как константы. Функция $2^{(0,1)}$ – константа 2, $true^{(0,1)}$ константа true.

Общая форма задания функций в FPTL – система уравнений

$$Xi^{(mi,ni)} = \tau^{(mi,ni)}$$

i = 1, 2, …, n, где τ_i – термы.

Пусть – $f^{(m,n)}$ (m, n)-арная функция, $f(\alpha)$ – результат ее применения к кортежу α. f_1, f_2 - заданные функции, α, β, γ – обозначение кортежей. [3]

2.1. Последовательная композиция (.)

$$f^{(m,n)} = f_1^{(m,k)} \cdot f_2^{(k,n)}$$

$$f(\alpha) = f_2(f_1(\alpha))$$

2.2. Операция конкатенации кортежей значений функций (*)

$$f^{(m,n_1+n_2)} = f_1^{(m,n_1)} * f_2^{(m,n_2)}$$

$$f(\alpha) = f_1(\alpha) f_2(\alpha)$$

2.3. Операция условной композиции

$$f^{(m,n)} = f_1^{(m,k)} \to f_2^{(m,n)}$$

$$f(\alpha) \begin{cases} f_2(\alpha), \text{если } f_1(\alpha) \text{ отлично от значения ложь или } \omega \\ \omega \text{ инчае} \end{cases}$$

Где ω – неопределенное значение

2.4. Операция объединения (графиков) ортогональных функций

$$f^{(m,n)} = f_1^{(m,n)} * f_2^{(m,n)}$$

$$f(\alpha) \begin{cases} f_1(\alpha), \text{если } f_1(\alpha) \text{ определена} \\ f_2(\alpha), \text{если } f_2(\alpha) \text{ определена} \end{cases}$$

Функции f_1, f_2 считаются ортогональными, если для всякого кортежа данных α определена не более чем одна из них

3. Сетевое представление

Формально сеть есть графическое представление функций, которое строится по правилам, приведенным на рисунке 1 [5].

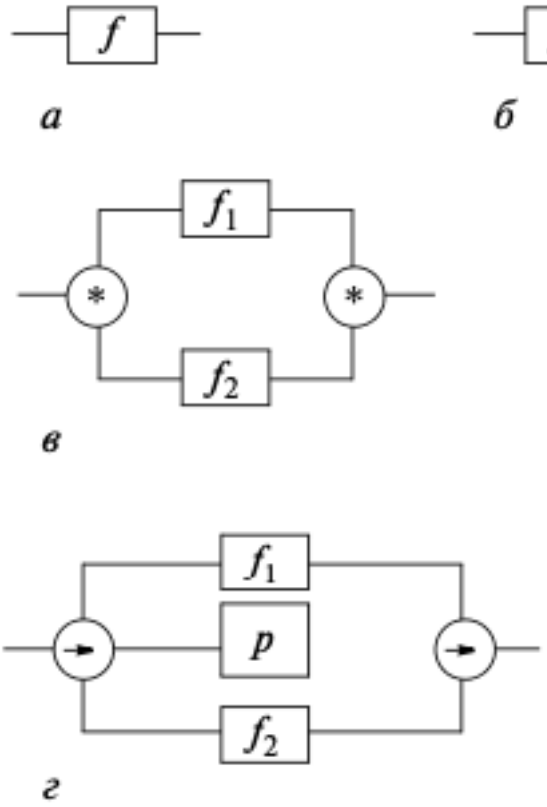

Рисунок 1 Графическое представление а) – базисной функции б) – последовательной композиции в) – параллельной композиции г) – условной композиции

Например, функция вычисления факториала будет иметь сетевой вид, приведенный на рисунке 2

```
scheme Fact {
      Fact = ([1] * 0).equal 1, (([1] * 1).sub.Fact * [1]).mul;
}
```

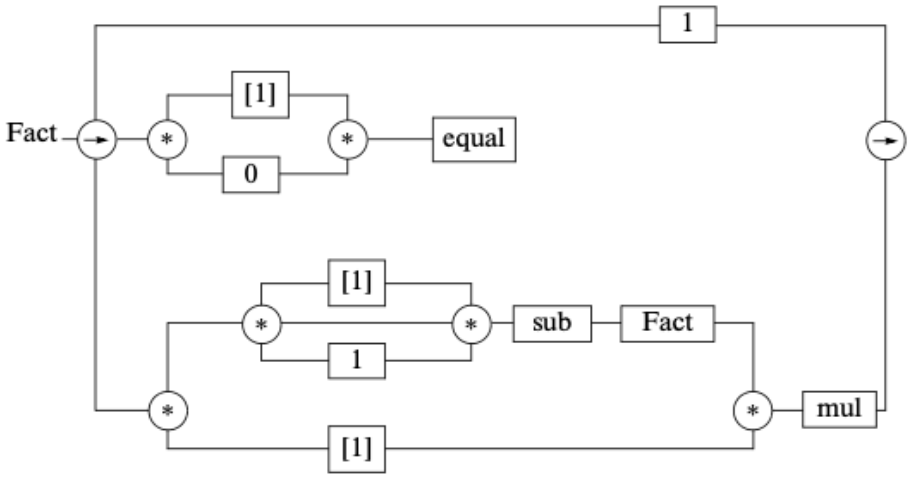

Рисунок 2 сетевое преставление факториала

4. Программная реализация

Программа реализована на языке java с применением платформы JavaFX и библиотеки JavaFXSmartGraph. Возможно преобразование функции в граф и обратно. В описании функции допустимо использовать операции “.”, “*”, “->” и литералы – последовательности букв латинского алфавита и цифр.

4.1. Парсинг выражения

Сначала входное выражение разбивается на токены – символ операции одной из композиций (последовательная – “.”, параллельная – “*”, условная – “->”) или литерал – последовательность из латинских букв и символов.

После токенизации, строится постфиксное представление выражения с помощью алгоритма сортировочной станции [4].

По постфиксной записи строится дерево, представляющее заданную функцию. Дерево имеет 4 типа узлов:

- Композиция (.) – имеет двух детей, левое и правое выражение композиции
- Конкатенация (*) – имеет двух детей, левое и правое выражение конкатенации
- Тернарная операция (->) – имеет трех детей: условие, истинную ветку и ложную ветку.
- Литерал – не имеет детей, хранит представление литерала

4.2. Преобразование в граф

Для преобразования в граф дерево выражение рекурсивно обходится и создается граф

```
Функция expressionToGraph(exp)
    Если exp является композицией(.) тогда
         right = expressionToGraph(exp.right)
         left = expressionToGraph(exp.left)
         связать left и right
         вернуть left
    Если exp является конкатенацией(*) тогда
         right = expressionToGraph(exp.right)
         left = expressionToGraph(exp.left)
         создать или получить вершины для открытия и закрытия операции(*)
         соединить left и right с вершинами открытия и закрытия
         вернуть вершину открытия
    Если exp является тернарной операцией(->) тогда
         trueBranch = expressionToGraph(exp.truBranch)
         falseBranch = expressionToGraph(exp.falseBranch)
         condition = expressionToGraph(exp.condition)
         создать вершины открытия и закрытия операции (->)
         соединить trueBranch, falseBranch, condition с вершинами
открыти/закрытия
         вернуть вершину открытия
    Если exp является литералом тогда
         Создать и вернуть вершину литерала
```

После для полученного графа вызывается рекурсивная функция, расставляющая узлы, так чтобы они не накладывались друг на друга, после чего граф отображается на экране.

Для функции факториала, будет отображен граф, приведенный на рисунке 3

```
(a1 * 0).equal -> 1, ((a1 * 1).sub.Fact * a1).mul
// через a1 обозначен первый аргумент функции, т.к. символы “[“, “]” не
допустимы в текущей реализации
```

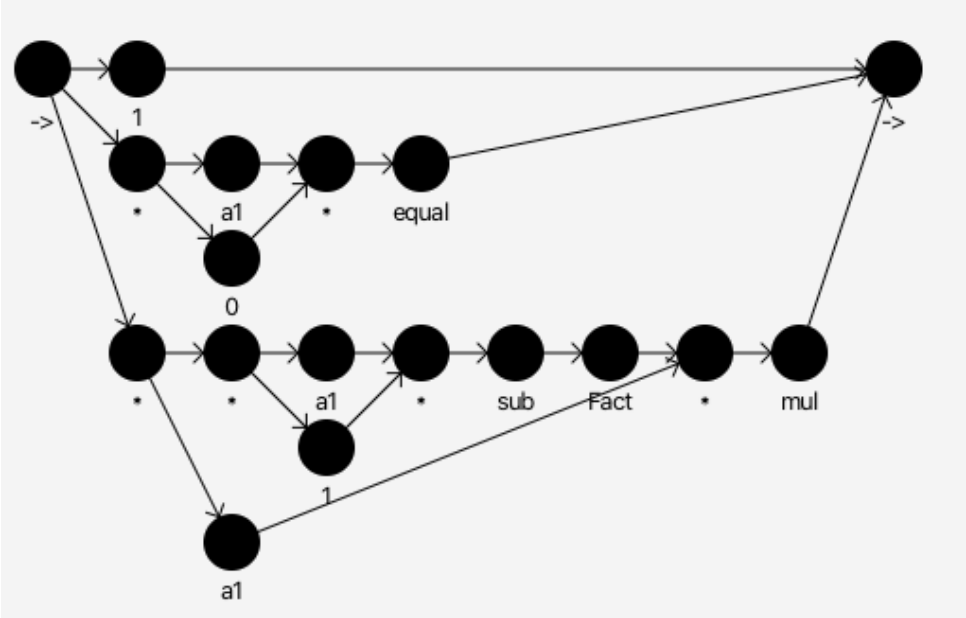

Рисунок 3 сетевое представление, созданное приложением

4.3. Преобразование графа в выражение

Чтобы преобразовать сетевое представление функции в текстовое, сначала с помощью графического редактирования создается граф. На рисунке 4 приведен интерфейс программы. После выбирается узел, с которого надо начать преобразование графа в выражение и для него вызывается рекурсивная функция, которая строит дерево выражения.

```
Функция graphToExpression(ExpressionVertex curr)
    Если curr является открывающей * тогда
        Преобразовать детей curr в выражения
        Объединить эти выражения в конкатенацию
        Вернуть выражение или композицию со следующей операцией, если она
есть
    Если curr является открывающей -> тогда
        Преобразовать условие, true-ветвь и false-ветвь в выражения
        Создать тернарное выражение из этих выражений
        Вернуть выражение или композицию со следующей операцией, если она
есть
     Если curr является литералом тогда
        Создать литеральное выражение из curr
        Вернуть это литеральное выражение или композицию со следующей
операцией, если она есть
```

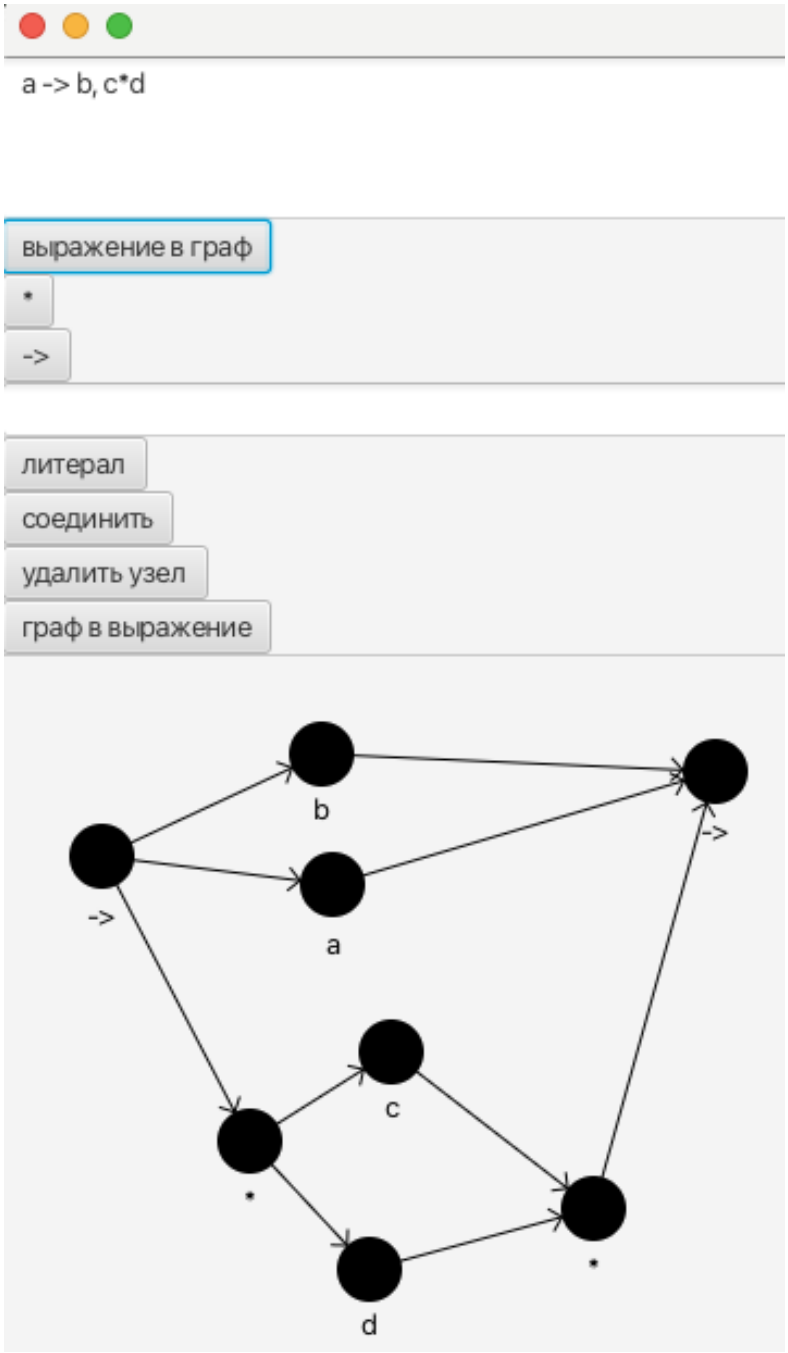

Рисунок 4 Интерфейс приложения

5. Заключение

В работе была разработана программа для преобразования функции, заданной на языке FPTL, в графическое представление и обратно. Программа реализована на языке Java с использованием платформы JavaFX и библиотеки JavaFXSmartGraph.

В работе были рассмотрены основные операции композиции функций в языке FPTL. Было показано, как эти операции могут быть представлены в графическом виде.

Программа позволяет пользователю вводить функцию в текстовом виде, преобразовывать ее в графическое представление и редактировать граф. Кроме того, программа может преобразовывать графическое представление функции обратно в текстовое.

6. Литература

1. Бажанов С.Е., Кутепов В.П., Шестаков Д.А. Язык функционального параллельного программирования и его реализация на кластерных системах // Программирование. 2005. № 5.
2. Бажанов С.Е., Кутепов В.П., Шестаков Д.А. Структурный анализ и планирование процессов параллельного выполнения функциональных программ // Изв. РАН. ТиСУ. 2005. № 6.
3. Кутепов В. П., Шамаль П. Н. Реализация языка функционального параллельного программирования FPTL на многоядерных компьютерах //Известия Российской академии наук. Теория и системы управления. – 2014. – №. 3. – С. 46-46.
4. Обратная польская запись // wikipedia URL: https://ru.wikipedia.org/wiki/Обратная_польская_запись (дата обращения: 8.11.2024).

7. ПРИЛОЖЕНИЕ

1. Исходный код - https://github.com/G0-G4/fptlVisualizer