ФГБОУ ВПО

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ



КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

КУРСОВАЯ РАБОТА

По курсу «Параллельное программирование и параллельные системы»

Тема: «Визуализация схем FPTL-программ»

Группа: А-13м-20

Студент: Мелихова В.Д.

Преподаватель: Кутепов В.П.

Москва 2021

Оглавление

[Введение 3](#_Toc91008149)

[Описание схемы FPTL-программы 4](#_Toc91008150)

[Программная реализация 5](#_Toc91008151)

[Алгоритм работы программы 5](#_Toc91008152)

[Используемое программное обеспечение 5](#_Toc91008153)

[Алгоритмы визуализации графа 5](#_Toc91008154)

[Примеры работы программы 7](#_Toc91008155)

[Заключение 9](#_Toc91008156)

[Список использованный источников 10](#_Toc91008157)

[Листинг программы 11](#_Toc91008158)

# Введение

Язык FPTL – функциональный язык мелкозернистого параллельного программирования. Параллелизм в нем является следствием операций композиции функций и не требуется специальных директив, указывающих на распараллеливание[1]. Вследствие этого была поставлена задача визуализации схем FPTL-программ, так как это позволит существенно упростить пользователю процесс анализа структуры программы и последовательности выполнения действий.

# Описание схемы FPTL-программы

Схема FPTL-программы, которую предстоит визуализировать, представляет из себя сгенерированный сторонним модулем JSON-файл. В нем в формате массива записей хранится внутреннее представление FPTL-программы.

У каждой записи есть несколько ключей, самыми важными из которых являются ID и тип. С помощью уникального ID и атрибута children устанавливаются связи между записями.

Записи бывают нескольких типов (например, Constant, Parallel, Condition и т.д.). В зависимости от типа у записей может быть различный набор ключей. Одним из важных ключей является complex, указывающий содержит ли узел рекурсию или сложную функцию.

В таблице 1 приведены примеры записей разных типов.

Таблица 1

|  |  |
| --- | --- |
| Объявление константы | Запись, указывающая на параллельное выполнение операций с ID=15 и 16 |
| {  "id": 1,  "type": "Constant",  "line": 10,  "column": 12,  "valueType": "integer",  "value": "100"  } | {  "id": 14,  "type": "Parallel",  "complex": 0,  "children": [15, 16]  } |

# Программная реализация

Уникальный ID записи можно использовать в качестве ID узла графа. С помощью значений ID из ключа children можно определить список дочерних узлов. Таким образом, существует возможность построить графическую визуализацию полученного JSON-файла, т.е. визуализацию схемы FPTL-программы, которую он описывает.

## Алгоритм работы программы

В качестве входных данных программа получает JSON-файл.

1. Производится парсинг JSON-файла. Массив записей из файла превращается в массив объектов класса FPTLNode. Сортировка массива объектов.
2. На основании данных об объекте строятся вершины и ребра графа.
3. В зависимости от характеристик объекта-вершины ему «присваивается» виджет с определенного вида.

Стоит отметить, что класс FPTLNode спроектирован таким образом, чтобы в случае изменения структуры записи – добавления новых атрибутов, удаления старых, изменение типов данных – было легко и быстро подстроить его под новые требования или даже не приходилось делать этого вовсе.

## Используемое программное обеспечение

Для выполнения данного задания был выбран фреймворк Flutter, позволяющий создавать нативные приложения под Windows, Apple и Linux с использованием языка программирования Dart.

## Алгоритмы визуализации графа

Для отображения данных в виде графа использовалась библиотека GraphView, поддерживающая три алгоритма визуализации графа:

1. Алгоритм Буххайма-Уолкера (Buchheim-Walker Algorithm);
2. Силовой алгоритм Фрухтермана — Рейнгольда;
3. Алгоритм послойного рисования графа (алгоритм Сугиямы).

Алгоритм Буххайма-Уолкера – алгоритм, предназначенный для визуализации только связных ациклических графов – деревьев. Его временная сложность составляет O(n), где n — число узлов входного графа[2].

Силовой алгоритм Фрухтермана — Рейнгольда предназначен для рисования любых графов. Однако большим недостатком таких алгоритмов является то, что в общем случае они имеют время работы, эквивалентное O(n3).

Послойное рисование графа или иерархическое рисование графа (алгоритм Сугиямы) — это способ визуализации ориентированных ациклических графов. Вершины графа рисуются горизонтальными рядами или слоями с рёбрами, преимущественно направленными вниз. В своей простейшей форме алгоритмы послойной визуализации графов могут требовать время O(mn) на графах с n вершинами и m рёбрами, поскольку может быть создано большое количество добавочных вершин. Однако, для некоторых вариантов алгоритма, можно смоделировать эффект дополнительных вершин без фактического их добавления, что приводит к реализации алгоритма с почти линейным временем выполнения [3].

Таким образом, можно видеть, что наиболее удачными из представленных являются алгоритмы Сугиямы и Буххайма-Уолкера.

Так как модуль, генерирующий JSON-файл, находится в процессе разработки, нельзя гарантировать, что граф будет иметь вид дерева, поэтому алгоритм Буххайма-Уолкера может не обеспечить корректную визуализацию. Вследствие этого было принято решение использовать алгоритм Сугиямы, так как он позволит визуализировать не только графы-деревья. Подобное решение также позволит упростить процесс тестирования генератора файлов-схем.

Однако стоит учитывать, что алгоритм Сугиямы по времени выполения проигрывает алгоритму Буххайма-Уолкера. Если в дальнейшем структура JSON-файла изменится, и граф всегда будет представлять из себя дерево, предпочтительно сменить алгоритм и использовать алгоритм Буххайма-Уолкера.

Примеры работы программы  
Пример работы программы можно видеть на рис. 1.

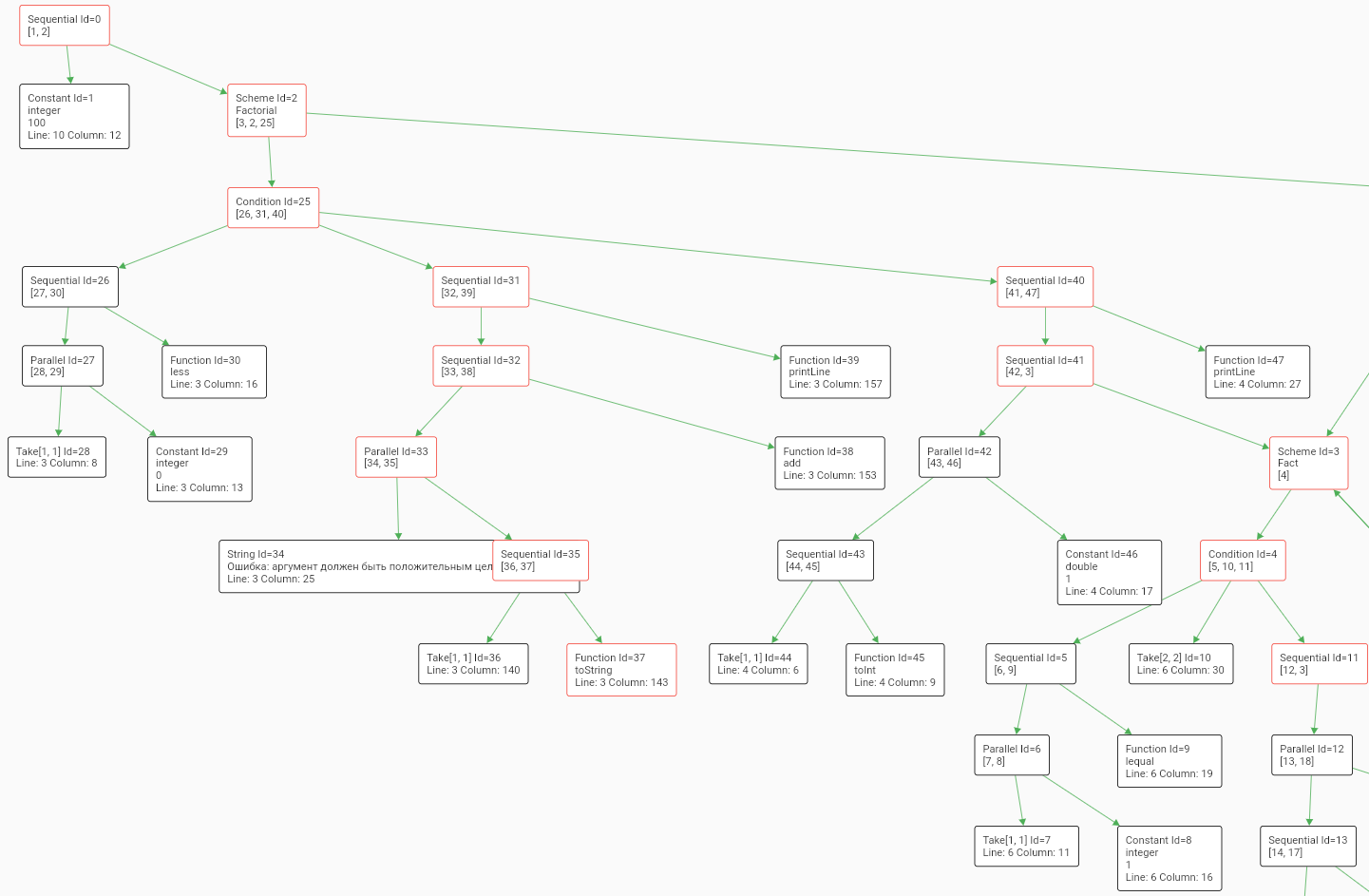


Рис. 1. Построенный граф

На рис. 2 можно видеть часть графа, построенную по записи c ID=14 из таблицы 1.

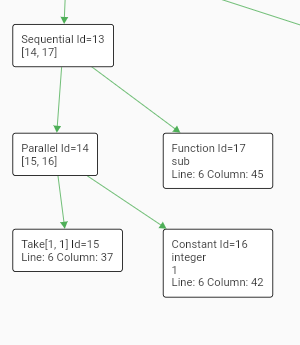


Рис. 2. Узел Parallel ID=14 с двумя дочерними узлами

Можно видеть, что каждому узлу соответствует виджет, содержащий основную информацию о нем: тип узла, ID; если есть – список дочерних узлов», значение, тип переменной и др.

Если узел содержит рекурсию или сложную функцию, он выделяется красным цветом (рис. 3).

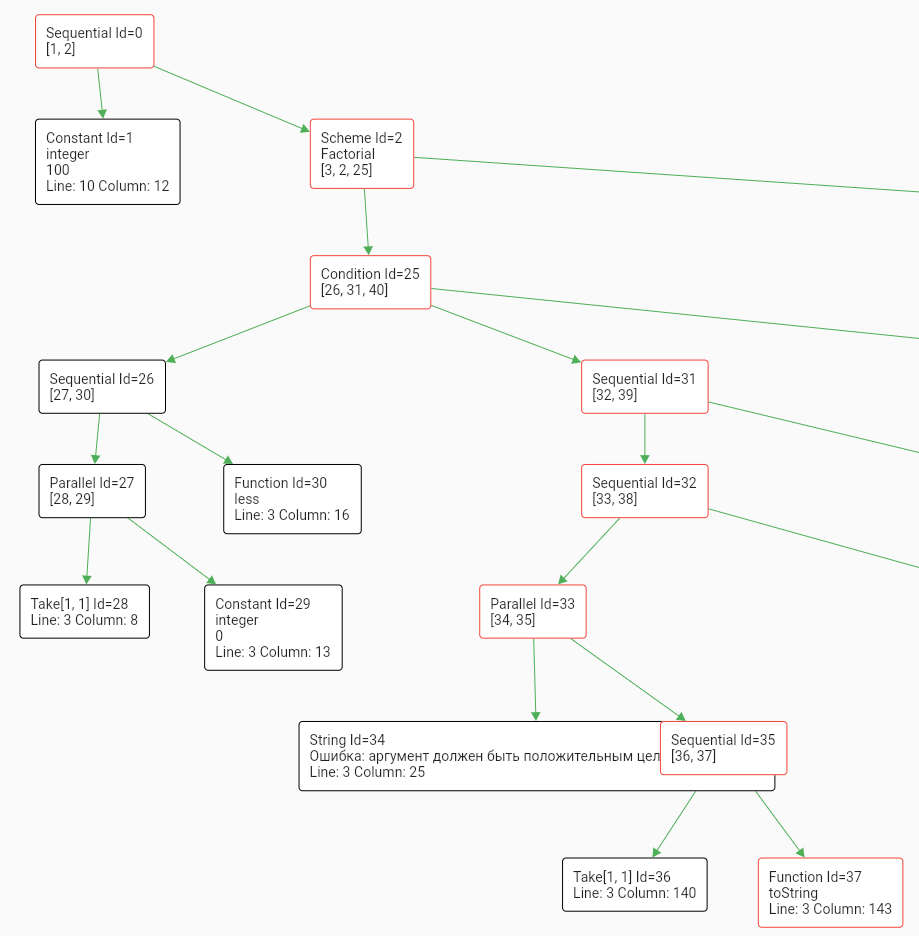


Рис. 3. Фрагмент графа

# Заключение

Был разработан программный комплекс, позволяющий визуализировать схему FPTL-программы, поступающую в формате JSON-файла. На схеме можно видеть основную информацию о каждом узле, понять, какие узлы выполняются последовательно/параллельно, содержат ли они рекурсию или сложную функцию.

# Список использованный источников

1. Кутепов В.П. Курс лекций «Параллельные системы и параллельное программирование».
2. Buchheim C., Junger M., Leipert S. Improving Walker’s Algorithm to Run in Linear Time [Электронный ресурс]. - ttps://link.springer.com/content/pdf/10.1007%2F3-540-36151-0\_32.pdf
3. Markus Eiglsperger, Martin Siebenhaller, Michael Kaufmann. An efficient implementation of Sugiyama’s algorithm for layered graph drawing // Graph Drawing, 12th International Symposium, GD 2004, New York, NY, USA, September 29-October 2, 2004, Revised Selected Papers. — Springer-Verlag, 2005. — Т. 3383.

# Листинг программы

main.dart

import 'package:flutter/material.dart';  
import 'package:flutter/services.dart' show rootBundle;  
import 'package:flutter/services.dart';  
import 'package:flutter/widgets.dart' show WidgetsFlutterBinding;  
import 'package:visual\_v1/classes.dart';  
import "dart:convert";  
  
import 'trying.dart';  
  
List<FPTLNode> smth = [];  
  
void main() async {  
 WidgetsFlutterBinding.*ensureInitialized*();  
  
 runApp(MaterialApp(  
 home: FutureBuilder(  
 future: loadAsset(),  
 builder: (context, snapshot) {  
 if (snapshot.hasData) {  
 return MyApp(doc: snapshot.data.toString());  
 } else {  
 return const Text('No data was loaded');  
 }  
 })  
 )  
 );  
}  
  
Future<String> loadAsset() async {  
 String doc = await rootBundle.loadString('assets/hell.json');  
  
 smth = (json.decode(doc) as List)  
 .map((e) => FPTLNode(e))  
 .toList();  
 smth.sort((a, b) => a.id.compareTo(b.id));  
 return doc;  
}

trying.dart

import 'package:flutter/material.dart';  
import 'package:graphview/GraphView.dart';  
  
import 'classes.dart';  
import 'main.dart';  
  
class MyApp extends StatelessWidget {  
 final String doc;  
  
 MyApp({required this.doc});  
  
 @override  
 Widget build(BuildContext context) => MaterialApp(  
 home: TreeViewPage(doc: doc),  
 );  
}  
  
class TreeViewPage extends StatefulWidget {  
 String doc;  
  
 TreeViewPage({required this.doc});  
  
 @override  
 \_TreeViewPageState createState() => \_TreeViewPageState(doc: doc);  
}  
  
class \_TreeViewPageState extends State<TreeViewPage> {  
  
 String doc;  
 \_TreeViewPageState({required this.doc});  
  
  
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 body: Column(  
 mainAxisSize: MainAxisSize.max,  
 children: [  
 Expanded(  
 child: InteractiveViewer(  
 constrained: false,  
 boundaryMargin: EdgeInsets.all(100),  
 minScale: 0.01,  
 maxScale: 5.6,  
  
 child: GraphView(  
 graph: graph,  
 *// algorithm: FruchtermanReingoldAlgorithm(),  
 // algorithm: BuchheimWalkerAlgorithm(builder, TreeEdgeRenderer(builder)),* algorithm: SugiyamaAlgorithm(builder2),  
 paint: Paint()  
 ..color = Colors.*green* ..strokeWidth = 1  
 ..style = PaintingStyle.stroke,  
 builder: (Node node) {  
 *// I can decide what widget should be shown here based on the id* var a = node.key!.value as int;  
 return nodeWidget(smth[a]);  
 *// if ([Func, Sequential, Parallel, Scheme, Condition].contains(obj.runtimeType) && obj.complex) {}* },  
 )),  
 ),  
 ],  
 ));  
 }  
  
 Widget nodeWidget(FPTLNode a) { *//int a* return InkWell(  
 onTap: () {  
 print('clicked');  
 },  
 child: Container(  
 padding: const EdgeInsets.all(10),  
 decoration: BoxDecoration(  
 color: Colors.*white*,  
 shape: a.type == 'Ref' ? BoxShape.circle : BoxShape.rectangle,  
 border: Border.all(  
 color: a.complex ? Colors.*red* : Colors.*black*,  
 width: 1,  
 ),  
 borderRadius: a.type == 'Ref' ? null : BorderRadius.circular(4),  
 ),  
 child: Text(a.str)), *//прописать Node* );  
 }  
  
 final Graph graph = Graph()..isTree = false; *//true для walker* BuchheimWalkerConfiguration builder = BuchheimWalkerConfiguration();  
 SugiyamaConfiguration builder2 = SugiyamaConfiguration();  
  
 @override  
 void initState() {  
 super.initState();  
  
 for (var element in smth) {  
 for (var child in element.children??[]) {  
 if (element.id == child) {  
 continue;  
 } else */\*if (element.id < child)\*/* {  
 graph.addEdge(Node.Id(element.id), Node.Id(child));  
 } */\*else {  
 var node = FPTLNode({"id":child,"type":"Ref"});  
 node.complex = smth[child].complex;  
 smth.add(node);  
 graph.addEdge(Node.Id(element.id), Node.Id(smth.length - 1));  
 }\*/* }  
 }  
  
  
 *// builder  
 // ..siblingSeparation = (100) //неадекватно, надо как-то в проге менять  
 // ..levelSeparation = (250)  
 // ..subtreeSeparation = (100)  
 // ..orientation = (BuchheimWalkerConfiguration.ORIENTATION\_TOP\_BOTTOM);* builder2  
 ..levelSeparation = (50)  
 ..nodeSeparation = (50)  
 ..iterations = (4)  
 ..orientation = (SugiyamaConfiguration.*ORIENTATION\_TOP\_BOTTOM*);  
 }  
}

classes.dart

class FPTLNode {  
  
 late int id;  
 late String type;  
 late int? from;  
 late int? to;  
 late String? name;  
 late String? valueType;  
 late String? value;  
 late int? line;  
 late int? column;  
 late bool complex;  
 late List? children;  
  
 late String str;  
  
 FPTLNode(node) {  
 id = node['id'];  
 type = node['type'];  
 from = node['from'];  
 to = node['to'];  
 name = node['name'];  
 valueType = node['valueType'];  
 value = node['value'];  
 line = node['line'];  
 column = node['column'];  
 complex = node['complex'] == 1;  
 children = node['children'];  
  
 String fromTo = from == null  
 ? (to == null ? "" : "[, $to]")  
 : (to == null ? "[$from,]" : "[$from, $to]");  
  
 String lnCn = line != null && column != null ? '\nLine: $line Column: $column' : '';  
  
 str = "$type$fromTo Id=$id"  
 "${name == null ? '' : '\n$name'}"  
 "${valueType == null ? '' : '\n$valueType'}"  
 "${value == null ? '' : '\n$value'}"  
 "$lnCn"  
 "${children == null ? '' : '\n$children'}"  
 ;  
 }  
}