Санкт-Петербургский государственный университет

Кафедра системного программирования

Батоев Константин Аланович

Структурный анализ диаграмм и генерация кода в среде программирования роботов TRIK Studio

Курсовая работа

Научный руководитель:
ст. преп. Я. А. Кириленко

Санкт-Петербург
2018

Оглавление

Введение

Среда TRIK Studio позволяет программировать кибернетические модели. Она поддерживает такие конструкторы роботов, как TRIK, Lego Mindstorms NXT, EV3, квадрокоптер Геоскан Пионер. Среда предоставляет два основных способа создания программы для роботов: визуальный и текстовый. В режиме визуального программирования создается диаграмма, состоящая из разных блоков и связей между ними, которые задают последовательность действий, выполняемых роботом. Альтернативным способом является написание кода на конкретном целевом языке (Javascript для TRIK, NXT OSEK C для Lego NXT, ассемблер-подобный код для Lego EV3).

В процессе программирования может потребоваться комбинирование текстового и визуального способов, поэтому возникает задача получения кода из диаграммы, например, в образовательных целях. Стоит заметить, что всегда можно сгенерировать *неструктурированный* код для диаграммы, если использовать операторы безусловного перехода (GOTO, JUMP) или если эмулировать машину состояний, где состояниями будут блоки диаграммы. Однако получение структурированного кода для диаграммы является приоритетной задачей, поскольку известно, что структурированный код легче понимать и дальше сопровождать. Генерация структурированного кода в текущей версии среды не всегда возможна для некоторых синтаксически корректных диаграмм, поэтому возникает потребность расширить класс диаграмм, для которых можно сгенерировать эквивалентный структурированный текстовый код.

Перед генерацией структурированного кода идет фаза *структуризации* графа потока управления, который задается самой диаграммой. Под структуризацией понимается получение абстрактного синтаксического дерева, узлы которого хранят метку-тип управляющей структур. Примеры таких структур: «if», «switch», «for», «while» и т.д. Далее будем называть такое абстрактное синтаксическое дерево *деревом управляющих структур*. В связи с тем, что генерация кода в высокоуровне-

вые языки не всегда возможна, имеет смысл рассмотреть способы анализа графа потока управления и реализовать новый алгоритм структуризации.

Постановка задачи

Целью данной курсовой работы является улучшение внутренней компоненты генерации кода в среде программирования роботов TRIK Studio. В рамках данной курсовой работы были поставлены следующие задачи.

- Реализовать алгоритм структуризации графа потока управления диаграммы.
- Провести апробацию алгоритма на диаграммах.

1. Обзор

1.1. Анализ графа потока управления

Два подхода к анализу графа потока управления рассмотрены в седьмой главе «Control flow analysis» книги [2]. Оба метода сначала вычисляют *отношение доминирования* на вершинах ориентированного корневого графа. Нахождение множества доминаторов может быть сделано разными алгоритмами. Первый использует рекуррентное соотношение, которое вытекает из определения доминатора:

$$Dominators(root) = \{root\},$$

$$Dominators(v) = \{v\} \cup \bigcap_{u \in pred(v)} Dominators(u)$$

Алгоритм заключается в нахождении неподвижной точки. Асимптотика этого алгоритма равна $O(n^2 * e)$, где n – количество вершин, а e – количество рёбер. Второй подход был рассмотрен в статье [1]. Он развивает дополнительную теорию на графах: вводит термин *полудоминатор*, для которого доказывает свойства и леммы, необходимые для доказательства корректности нового алгоритма, чья асимптотика равна $O(e * log(n))$.

После вычисления доминаторов можно применять различные методы анализа. Первый — *итеративный*, занимается поиском циклов при помощи доминаторов и не подходит для решения нашей задачи, поскольку не поддерживает поиск остальных структур кроме циклов.

Второй — *интервальный*, выполняет более глубокое исследование. Он пытается последовательно заменять некоторые вершины графа на одну согласно шаблонам трансформации и строит дерево управляющих структур.

1.2. Структурный анализ

Улучшенной версией интервального анализа является *структурный анализ*, который отличается тем, что определяет больший набор шаб-

лонов — областей вершин графа, которые он может идентифицировать. Области делятся на два больших класса: ациклические и циклические. Отношение доминирования используется в структурном анализе для нахождения циклических структур. Стоит отметить, что анализ не всегда завершается успешно.

2. Алгоритм

За основу алгоритма структурного анализа взят алгоритм из книги [2]. Он получает на вход ориентированный граф потока управления, а на выходе возвращает дерево управляющих структур, если анализ завершился успешно (см. алгоритм 1).

```
Data: Граф
Result: Указатель на дерево
Найти доминаторы;
Построить изначальный лес деревьев;
Обойти граф обходом в глубину и запомнить время посещения
  вершин в обратном порядке;
Запомнить максимальное время;
while (вершин в графе больше одной) И (текущее время меньше,
  либо равно максимальному) do
    foreach шаблон do
        Проверять соответствие подграфа, содержащему вершину
          в качестве головы, данному шаблону;
        if соответствие прошло успешно then
            Заменить необходимые вершины на новую;
            Обновить структуры;
            break;
        end
    end
    if не произошло ни одного соответствия then
        Увеличить текущее время;
    end
end
if в графе одна вершина then
    Вернуть указатель на дерево, соответствующее данной
      вершине;
else
    Вернуть нулевой указатель, свидетельствующий о том, что не
      смогли структурировать граф;
end
```

Algorithm 1: Псевдокод алгоритма структурного анализа

2.1. Структуры данных

Алгоритму требуются различные структуры данных. А именно нужно знать начальную вершину в графе, а для каждой вершины должны быть определены:

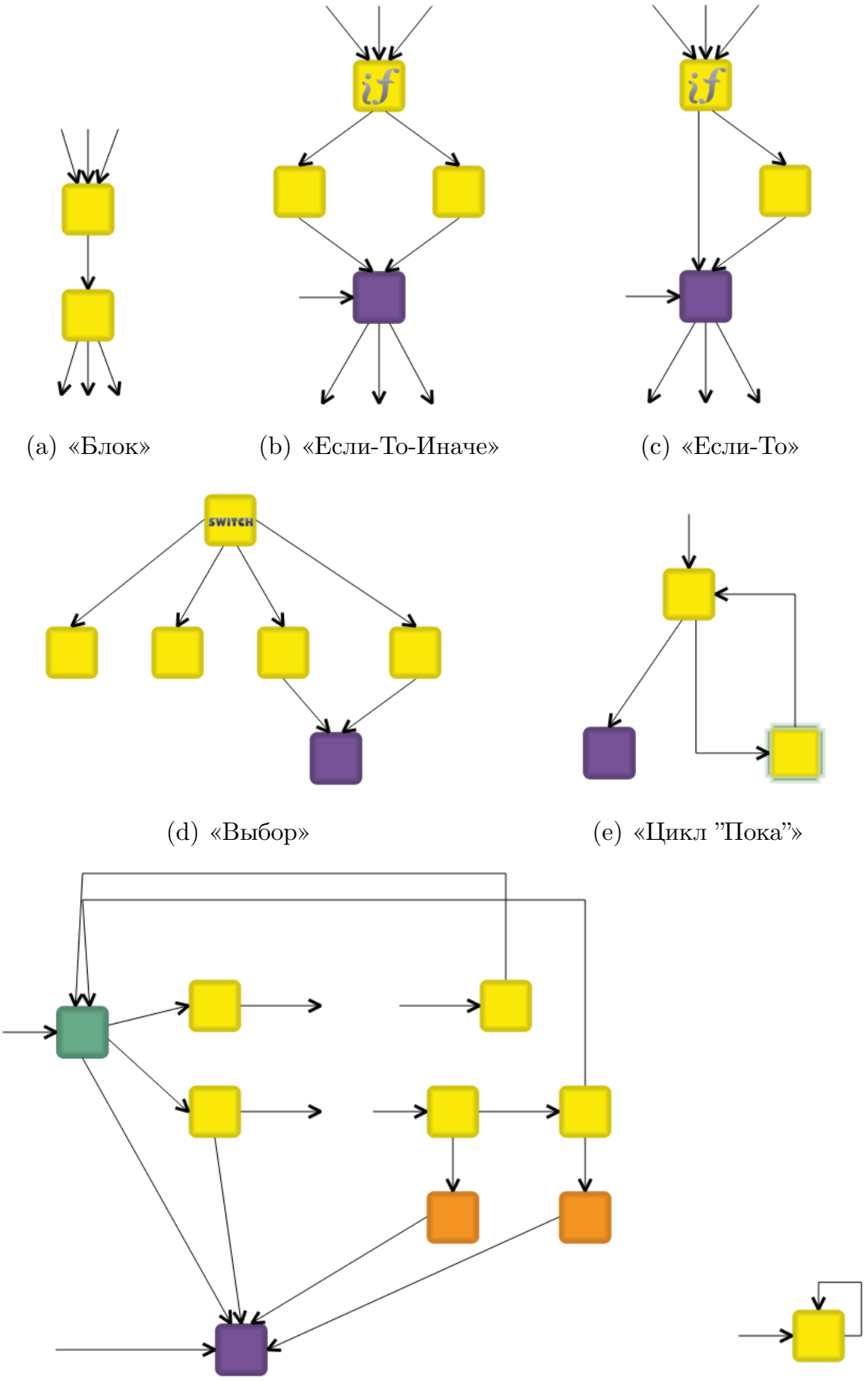
- множества последующих и предшествующих вершин;
- множество вершин, которые доминируют над ней;
- номер вершины в обратном порядке;
- соответствующее дерево управляющих структур.

Перед основным ходом алгоритма следует их заполнить. Начальная вершина – вершина, в которую не входит ни одно ребро. Такая найдется ровно одна из структуры самой диаграммы. Первые два множества создаются при построении графа. Для вычисления отношения доминирования воспользуемся определением доминатора. Это не самый эффективный алгоритм, однако его время работы приемлемо в связи с не очень большими размерами графа. Анализ целесообразно проводить снизу вверх, поэтому занумеруем вершины графа в обратном порядке. Таким образом, время потомка будет меньше времени предка. Для нахождения номера вершины в обратном порядке проведем обход в глубину из начальной вершины.

2.2. Шаблоны

Перейдем к основной части алгоритма. Для каждой вершины графа алгоритм пытается по очереди сопоставлять шаблоны. Все шаблоны, кроме «Цикла "Пока" с инструкциями прерывания», были взяты из книги. Сопоставление происходит при помощи проверки количеств исходящих и входящих ребер, а также вершин, которые они соединяют. Если произошло соответствие, то происходит изменение графа. Для шаблона «Цикл "Пока" с инструкциями прерывания» есть своя стратегия обработки, в то время как для остальных случаев распознанный

фрагмент графа заменяется на новую «абстрактную» вершину с соответствующим обновлением структур данных.



(a) «Блок» (b) «Если-То-Иначе» (c) «Если-То»

(d) «Выбор» (e) «Цикл ”Пока”»

(f) «Цикл ”Пока” с инструкциями прерывания» (g) «Бесконечный цикл»

Рис. 1: Шаблоны для структурного анализа.

Теперь опишем шаблоны, которые будет обнаруживать наш анализ. «Блок» (см. рис. 1a) распознает последовательные вершины графа потока управления. Его идея заключается в следующем: когда бы поток исполнения ни оказался в верхнем участке, то после него управление всегда переходит к нижнему. Шаблон «Если-То-Иначе» (см. рис. 1b) предназначен для поиска условных конструкций с двумя альтернативами. Несмотря на связь с предыдущим шаблон «Если-То» (см. рис. 1c) является самостоятельным шаблоном, так как его нельзя выразить через другие. Шаблон «Выбор» (см. рис. 1d) является расширением условных конструкций. В его голове находится вершина, из которой выходит несколько ребер, в различные альтернативы исполнения. Каждая альтернатива может содержать максимум одно исходящее ребро, которое будет вести в общее место передачи управления.

Представим теперь шаблоны, которые находят циклические структуры. Шаблон «Цикл "Пока"» (см. рис. 1e) представляет собой цикл, который после исполнения передает управление какой-то другой вершине. Шаблон «Бесконечный цикл» (см. рис. 1g) представляет точку в программе, которая всегда передает управление себе.

Во всех шаблонах вершины, отмеченные желтым цветом, заменяются новой вершиной. Фиолетовая вершина помогает определить шаблон.

«Цикл "Пока" с инструкциями прерывания» (см. рис. 1f) — распознает циклы, которые могут из нескольких вершин переходить к точке, которой передается управления после цикла. Для его обнаружения нужно найти тело цикла, а затем для каждой вершины из тела рассматривать все ребра и анализировать вершины, в которые они идут. Зеленая вершина представляет голову цикла. Желтые — тело. Оранжевые — «промежуточные вершины», а фиолетовая — «выход». Далее будет введены определения для «промежуточной вершины» и «выхода».

Результатом успешного сопоставления является множество вершин и множество ребер, которые нужно удалить.

2.3. Редукция

Опишем процесс преобразования графа и структур данных. После обнаружения всех шаблонов, кроме шаблона «Цикл "Пока" с инструкциями прерывания», происходит замена вершин графа новой вершиной. Если в графе есть ребра, которые ведут к удаляемым вершинам или исходят из них, но не принадлежат множеству удаляемых вершин, то перенаправим их к новой вершине, чтобы сохранить связность графа.

Затем обновляется отношение доминирования. Доминаторы для новой абстрактной вершины — это доминаторы вершины в голове шаблона. Однако еще нужно обновить отношение для старых вершин. А именно для тех, над которыми доминирует голова шаблона. Для каждой их них нужно убрать заменяемые вершины и добавить новую.

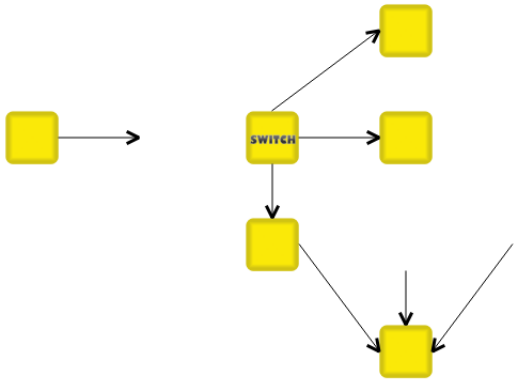
Остается привести к актуальному виду оставшиеся структуры: множество деревьев и соответствие времени обратного обхода вершине графа. Во множество деревьев добавляется новое дерево, которое содержит в качестве сыновей деревья заменяемых вершин и хранит тип встреченного шаблона. Для новой абстрактной вершины в качестве времени обратного порядка естественно взять максимум среди заменяемых вершин. Затем все времена вершин приводятся к промежутку без «дырок», которые возникают при удалении времен заменяемых вершин.

Новый шаблон «Цикл "Пока" с инструкциями прерывания» обрабатывается по-другому. У него есть три основных вида вершин:

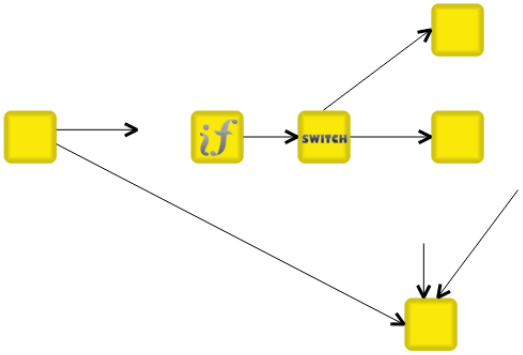
- вершина из тела цикла;
- вершина, в которую переходит управление после цикла, которую назовем «Выход»;
- вершина, которая не принадлежит телу цикла, но из которой есть ребро в вершину «Выход». Назовем её "Промежуточной вершиной", а связанную с ней вершину из тела цикла — «Условием».

Конструкция такого цикла очень похожа на шаблон «Цикл "Пока"». Для точного сопоставления мешают «Промежуточные вершины». Что-

бы упростить граф потока управления, заменим каждую пару «Условие» — «Промежуточная вершина» на новую абстрактную вершину, управляющее дерево которой будет иметь тип «Если с прерыванием» и содержать соответствующие деревья в качестве потомков. Однако возникает сложность, когда «Условие» может иметь несколько ребер, ведущих в тело цикла. Это значит, что нельзя заменить «Условие» и «Промежуточную вершину» новой вершиной, потому что потеряется «Условие» для определения пути исполнения, если поток управления останется в цикле. Естественным решением является сохранить вершину «Условие», а новую абстрактную вершину добавить строго перед ней (см. рис. 2). Если есть ребро из «Промежуточной вершины» к вершине «Выход», то оно удаляется. А вместо него добавляется ребро от головы цикла к вершине «Выход» для явного указания вершины передачи управления после ”исполнения” цикла.



(a) Выход из цикла с помощью «Выбора»



(b) Преобразование

Рис. 2: Схема обработки циклов с инструкцией выхода из вершины «Выбор»

2.4. Завершаемость алгоритма

Докажем по индукции количества вершин, почему алгоритм завершается.

- База. Если в графе всего одна вершина, то алгоритм успешно завершается.

- Переход. Все шаблоны, кроме шаблона «Цикл ”Пока” с инструкциям прерываниями», уменьшают количество вершин. Оставшийся шаблон — не увеличивает количество вершин, однако он не может быть применен к одному и тому же циклу два раза подряд. Следовательно, в конечном итоге все циклы программы будут об-

работаны, а на следующей итерации либо будет применен шаблон, который уменьшает количество вершин, либо ничего не случится. В последнем случае граф признается неструктурируемым, алгоритм завершится, и вернется нулевой указатель.

Следовательно, завершаемость алгоритма доказана.

3. Реализация

В среде TRIK Studio был реализован алгоритм структурного анализа графа потока управления, заданного диаграммой. Поскольку среда написана на языке C++ при помощи фреймворка Qt, было решено применить те же технологии.

3.1. Структуры данных

Для хранения множеств доминаторов, последующих и предшествующих вершин, а также деревьев управляющих структур были заведены объекты класса QMap, который является словарем, сопоставляющим вершине соответствующее множество вершин или соответствующее дерево.

Для сохранения типа только что созданного дерева управляющих структур и для легкого доступа к потомкам есть абстрактный класс «IntermediateNode», от которого были унаследованы конкретные представители:

- «SimpleNode», являющийся "листом" иерархии и оберткой для вершины диаграммы (типа qReal::Id);
- «BlockNode», создающийся при распознавании шаблона «Блок» и содержащий указатели типа «IntermediateNode» на первый и на второй элементы его составляющие;
- «IfNode», создающийся при распознавании шаблонов «Если-То» и «Если-То-Иначе». Он содержит указатели для "условия" (типа «SimpleNode»), для ветки "То" (типа «IntermediateNode»), и для ветки "Иначе" (типа «IntermediateNode»);
- «SwitchNode», создающийся при встрече шаблона «Выбор», содержит указатель для "Выражения" (типа «SimpleNode») и список веток, каждая из которых имеет тип «IntermediateNode»;

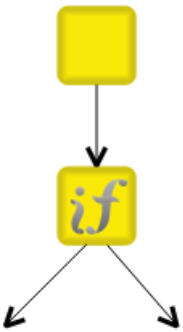
- «WhileNode», создающийся при встрече шаблона «Цикл ”Пока”», содержит указатель для ”начала” и ”тела” (оба имеют тип «IntermediateNode»);

- «SelfLoopNode», создающийся при встрече шаблона «Бесконечный цикл», содержит указатель для ”тела” (типа «IntermediateNode»);

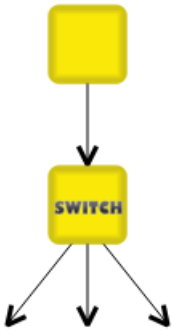
- «IfWithBreakNode», создающийся при встрече шаблона «Цикл ”Пока” с прерываниями», Он содержит указатель на «условие» (типа «SimpleNode») и на «действие», которое должно быть выполнено до вызова прерывания” (типа «IntermediateNode»).

3.2. Нахождение шаблонов

Как видно из структур, «условием» шаблонов «Если-То-Иначе», «Если-То», «Выбор» могут быть только объекты класса «SimpleNode», то есть, простые условия. Это было сделано для упрощения процесса генерации кода, а именно чтобы «условие» и «ветки», которые ему соответствуют, находились в одном дереве управляющих структур. В связи с этим был изменен шаблон ”Блок”: теперь из нижнего блока может исходить максимум одно ребро (см. рис. 3). Теперь фрагменты графа, аналогичные на рис. 3a и 3b, не будут сжиматься в новую абстрактную вершину.



(a) Плохой блок с ”Если”



(b) Плохой блок с ”Выбором”

(c) Хороший блок, у которого есть продолжение

(d) Хороший блок, у которого нет продолжения

Рис. 3: Примеры хороших и плохих фрагментов для шаблона «Блок».

Для поиска большинства шаблонов достаточно проверять представленные схемы (см. рис. 1). Однако для «Цикла ”Пока” с инструкциями прерываниями» действуем по-другому. Для него нужно найти с помощью отношения доминирования множество вершин, составляющих цикл, а затем рассматривать различные ситуации расположения вершины «Выход». Во-первых, её может не быть совсем. Во-вторых, она может находиться на втором уровне, в случае отсутствия «Промежуточных вершин». Наконец, «Выход» может располагаться на третьем уровне. Опишем достаточные условия для того, чтобы вершина была вершиной «Выход»:

- если в вершину, не принадлежащую телу цикла, входит более одного ребра, хотя бы одно из которых от вершины, принадлежащей циклу, то такая вершина является «Выходом»;

- если в вершину, не принадлежащую телу цикла, входит одно ребро, а исходит более одного ребра, то такая вершина — «Выход»;
- если имеется всего лишь одна «Промежуточная вершина», то она — «Выход».
- если есть несколько «Промежуточных вершин», которые ведут в одну общую вершину, то эта вершина — «Выход».

После нахождения вершины «Выход» выполняется проверка того, что вершина, удовлетворяющая достаточным условиям, единственная. После этого происходит соответствующее изменение графа, описанное в разделе «Редукция».

3.3. Архитектура

Конечной целью структурного анализа является дерево управляющих структур, которое используется для генерации структурированного кода. При генерации учитываются уже обнаруженные шаблоны, а также информация, содержащаяся в диаграмме: типы вершин и ребер.

Приведем диаграмму классов части системы, связанной с генерацией структурированного кода. Класс «Structurizator» инкапсулирует алгоритм структурного анализа и связанные с ним структуры данных. Основной публичный метод данного класса «performStructurization» возвращает указатель на корень полученного дерева, если алгоритм отработал успешно, и нулевой указатель, если в графе есть неструктурируемый фрагмент. Инкапсуляция алгоритма структуризации в отдельный класс было мотивирована возможностью переиспользования алгоритма и его независимостью от остальных компонентов системы TRIK Studio. Генерация кода происходит при вызове метода «generate» класса «SemanticTree».

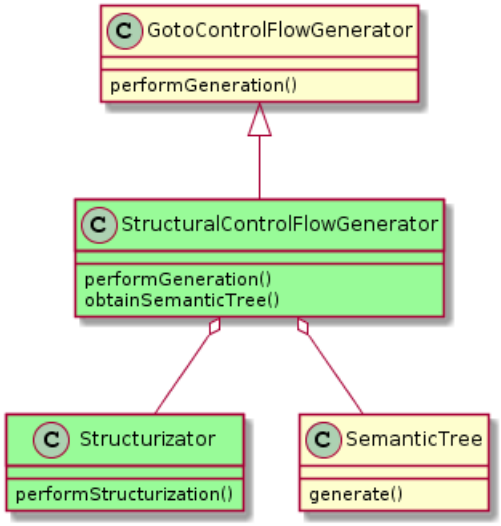


Рис. 4: Архитектура подсистемы генерации структурированного кода. Зеленым цветом показаны добавленные классы

4. Тестирование

Алгоритм был апробирован на различных диаграммах для роботов. Спектр тестов был разнообразным: начиная от простых диаграмм, содержащих только последовательности блоков и ветвления, заканчивая примерами, содержащими неочевидную циклическую структуру. Опишем наиболее интересные случаи анализа диаграмм, содержащих инструкции прерывания внутри цикла.

На рис. 5 изображена диаграмма внутри которой есть цикл, на ней нет явного "Выхода", поскольку промежуточных вершин несколько, но нет их общего потомка. На рис. 6 — диаграмма, у которой находится вершина "Выход" на третьем уровне, так как все "Промежуточные вершины" имеют только одно исходящее ребро, которое идет в нее. На рис. 7 нет промежуточных вершин, а вершина "Выход" находится на втором уровне, так как в нее входит несколько ребер из вершин, принадлежащих циклу. На рис. 8 есть две промежуточные вершины и вершина "Выход", которая определяется тем, что из нее исходит больше одного ребра.

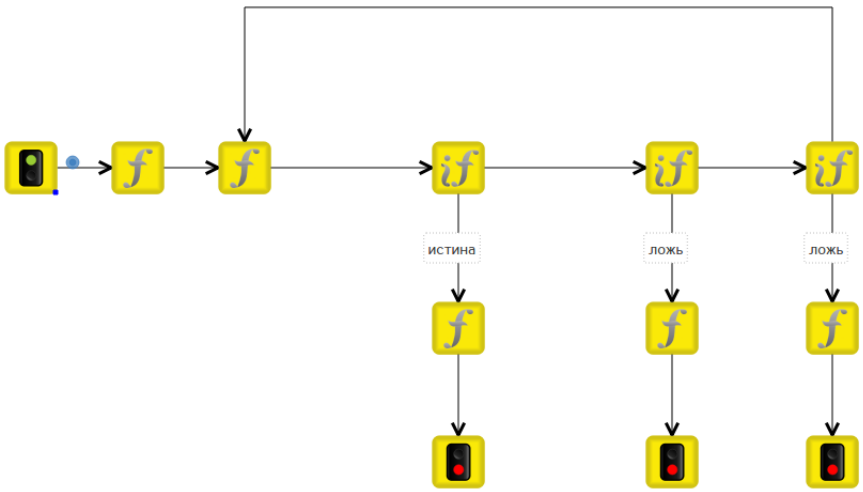


Рис. 5: Пример диаграммы, когда нет общего "Выхода" из цикла

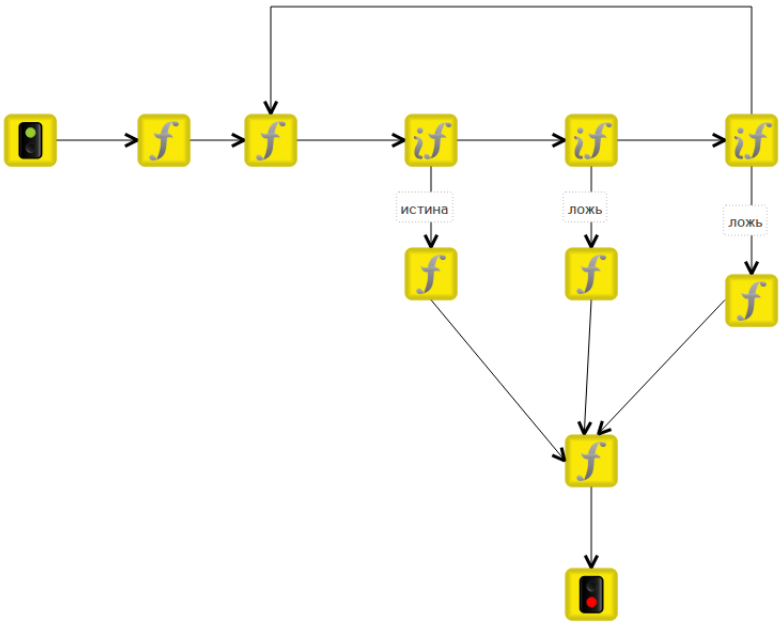


Рис. 6: Пример диаграммы, когда общий ”Выход” находится на третьем уровне

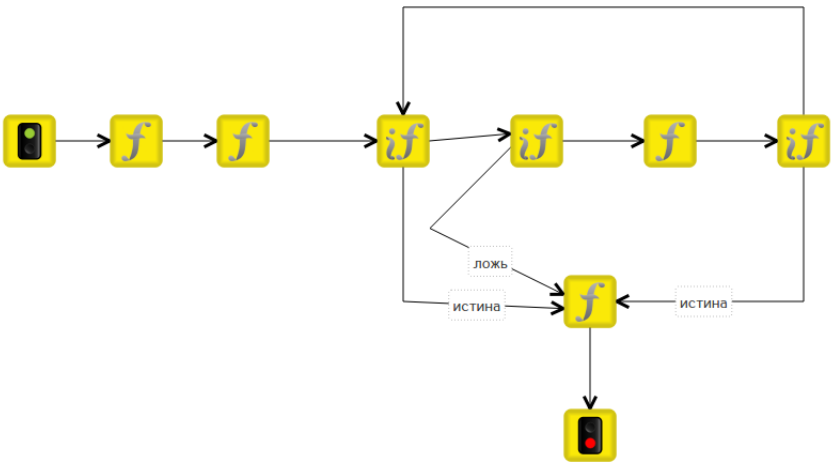


Рис. 7: Пример диаграммы, когда ”Выход” очевиден сразу

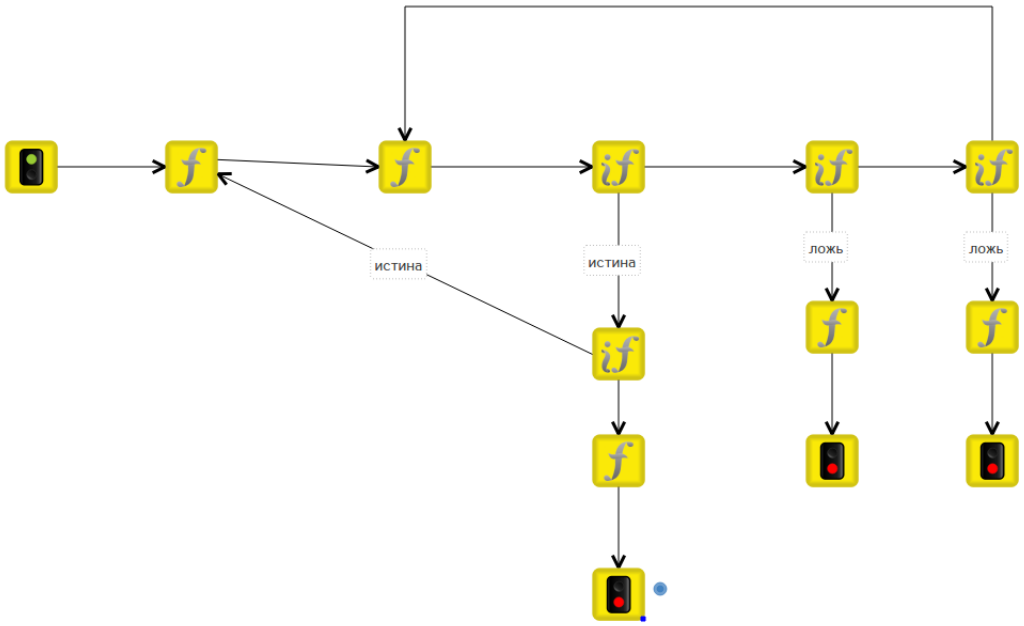


Рис. 8: Пример диаграммы, когда ”Выход” определяется только тем, что из него исходит больше одного ребра

В результате тестирования для всех диаграмм были получены деревья управляющих структур, из которых был сгенерирован код.

Заключение

В результате данной курсовой работы были выполнены следующие задачи:

- был реализован алгоритм структурного анализа в качестве алгоритма структуризации графа потока управления в среде TRIK Studio;
- алгоритм был апробирован на диаграммах для роботов TRIK.

Перспективы развития

Наметим основные дальнейшие планы:

- интегрировать алгоритм структуризации в главную ветку среды TRIK Studio;
- линеаризовать неструктурируемые участки графа потока управления.

Список литературы

[1] Lengauer Thomas, Tarjan Robert Endre. A fast algorithm for finding dominators in a flowgraph // ACM Transactions on Programming Languages and Systems (TOPLAS). — 1979. — Vol. 1, no. 1. — P. 121–141.

[2] Muchnick Steven S. Advanced compiler design implementation. — Morgan Kaufmann, 1997.