Санкт-Петербургский политехнический университет Петра Великого
Физико-механический институт
Высшая школа прикладной математики и вычислительной физики

Отчёт по проекту «Учебное инструментальное средство задания синтаксиса и семантики предметно ориентированных языков»

по дисциплинам
«Грамматики и автоматы» и «Моделирование на UML»

Выполнили студенты
группы 5030102/90201

Воротников Андрей
Кожевникова Диана
Павлов Илья

Преподаватель

Новиков Фёдор Александрович

Санкт-Петербург
2023

СОДЕРЖАНИЕ

Список иллюстраций

1 Постановка задачи

Назначение работы - создание учебного инструментального средства для создания реализаций языков предметной области (Domain Specific Language, DSL) для использования в рамках курса "Грамматики и автоматы".

Для решения задач определённой предметной области можно использовать языки программирования общего назначения (General Purpose Language, GPL). Этот подход на текущий момент является общепринятым, но имеет минусы:

1. Специалист предметной области может не знать языков общего назначения.
2. Языки программирования общего назначения не отражают специфику предметной области.

Эти проблемы можно решить написанием языка предметной области. Такой язык будет отражать специфику области, для которой создан, поэтому:

1. Специалисту предметной области не придётся изучать язык программирования общего назначения.
2. Уменьшаются трудовые и финансовые затраты на решение задач в данной предметной области.

Задачей инструмента является предоставление создателям языков предметной области удобного средства реализации языка.

2 Составные части языка и способы их задания

В работе рассматриваются 3 части языка:

1) лексика;
2) синтаксис;
3) семантика.

Опишем назначение и способы задания каждой части.

2.1 Лексика

Лексикой называется множество элементарных конструкций, называемых лексическими единицами или лексемами. Лексемы по своей сути атомарны - они не могут содержать другие лексемы.

В инструменте для описания лексем используются регулярные выражения. Таким образом, лексема состоит из её типа и регулярного выражения.

Пример. Для языка вычисления выражения над неотрицательными целыми числами с поддержкой операций суммы и произведения можно использовать следующую лексику:

Тип лексемы	Регулярное выражение
число	[1-9]\d*
операция	[\+*]
разделитель	,

Таблица 1: Лексика языка вычисления примеров

2.2 Синтаксис

Синтаксисом называются правила, по которым из лексем строятся операторы языка. Из операторов в свою очередь строится программа. В инструменте для описания синтаксиса используются контекстно-свободные грамматики.

В инструменте поддерживатся 2 способа задания синтаксиса:

1. Регулярная форма Бэкуса — Наура.
2. Диаграммы Вирта.

2.2.1 Регулярная форма Бэкуса — Наура

В основе этого способа задания синтаксиса лежит регулярная форма Бэкуса-Наура. Регулярная форма является только списком правил вывода формальной грамматики. Для инструмента было принято решение расширить этот способ задания грамматики. Целью расширения является возможность задавать лексику, ключевые выражения, аксиому грамматики и правила поведения при ошибке.

Описание грамматики в регулярной форме Бэкуса-Наура состоит из 5 обязательных блоков и 1 необязательного блока. Блоки:

1) описание лексики - содержит имена лексем и соответствующие этим именам регулярные выражения;
2) список ключевых слов и выражений - содержит список строк, которые являются выделенными для языка и непосредственно участвуют в описании синтаксиса;
3) список нетерминалов - содержит имена, которые являются именами нетерминалов порождающей грамматики;

4) аксиому грамматики - содержит имя одного из нетерминалов, этот нетерминал считается аксиомой грамматики;

5) набор правил, задающих поведение при наличии ошибки;

6) набор правил порождающей грамматики.

Необязательным блоком является блок ошибок, сейчас он игнорируется. В нём задаётся нетерминал и список нетерминалов, до которых идёт разбор в случае невозможности разбора синтаксиса.

Для описания правил порождающей грамматики используется:

1) альтернация;

2) опциональные последовательности;

3) итерация Цейтина.

Описание грамматики принятой регулярной формы Бэкуса-Науэра приведено в приложениях.

Пример. Для языка вычисления выражения над неотрицательными целыми числами с поддержкой операций суммы и произведения используется следующий синтаксис, который задан регулярной формой Бэкуса-Наура.

Примечание. Красным выделены ключевые символы и слова регулярной формы Бэкуса-Наура. Символы '+'; '*'; ',' попали и в KEYS и в operation, потому что о ключах сканнеру не известно, они появляются на этапе послесканнера.

```
TERMINALS:
    number ::= '[1-9]\d*';
    operation ::= '[\+\*]';
    terminator ::= ',.

KEYS: '+'; '*'; ',.

NONTERMINALS:
    EXPRESSIONS;
    EXPRESSION;
    TERM.

AXIOM: EXPRESSIONS.
```

RULES:
EXPRESSIONS ::= { EXPRESSION # , };
EXPRESSION ::= { TERM # + };
TERM ::= { number # * }.

2.2.2 Диаграммы Вирта.

Синтаксическими диаграммами Вирта называются особый вид орграфов, предзначенных для записи контекстно-свободных грамматик в графической форме.

Для задания синтаксических диаграмм в инструменте предлагается использовать язык описания графов DOT.

1. Граф должен быть ориентированным, в терминах DOT - digraph.
2. Все рёбра ориентированные.
3. Разрешены следющие виды вершин:
   - начальная - должна иметь тип "plaintext";
   - конечная - должна иметь тип "point";
   - нетерминальные - содержат имя нетерминала, должны иметь тип "box";
   - терминальные - содержат имя терминала, должны иметь тип "diamond";
   - ключевые - содержат ключи, должны иметь тип "oval".
4. Контекстные условия:
   - Начальные и конечные вершины должны быть в диаграмме единственными.
   - В начальную вершину не могут входить дуги.
   - Из конечной вершины не могут исходить дуги.
   - Конечная вершина должна быть достижима из начальной.

Примечание. В DOT атрибуты записываются в квадратных скобках. Указание пустого label необязательно.

Пример. Для языка вычисления выражения над неотрицательными целыми числами с поддержкой операций суммы и произведения используется следующий синтаксис, который задан синтаксическими диаграммами Вирта, полученными процессом языка DOT:

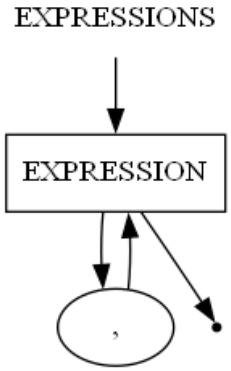

Рис. 1: Диаграмма Вирта набора выражений

Листинг кода на языке DOT, который задаёт нетерминал EXPRESSIONS:

```
digraph EXPRESSIONS {
    start [label=EXPRESSIONS shape=plaintext]
    A [label=EXPRESSION shape=box]
    B [label="," shape=oval]
    end [label="" shape=point]
    start -> A
    A -> B
    B -> A
    A -> end
}
```

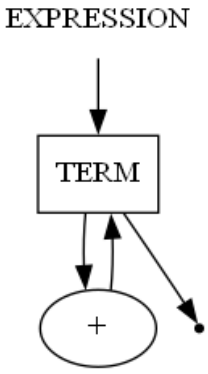

Рис. 2: Диаграмма Вирта выражения

Листинг кода на языке DOT, который задаёт нетерминал EXPRESSION:

```
digraph EXPRESSION {
    start [label=EXPRESSION shape=plaintext]
    A [label=TERM shape=box]
    B [label="+" shape=oval]
    end [label="" shape=point]
```

```
    start -> A
    A -> B
    B -> A
    A -> end
}
```

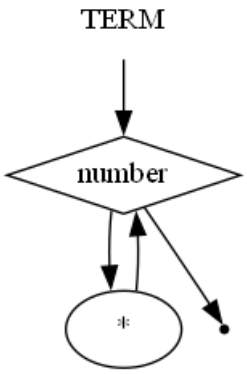

Рис. 3: Диаграмма Вирта слагаемого

Листинг кода на языке DOT, который задаёт нетерминал TERM:

```
digraph TERM {
    start [label=TERM shape=plaintext]
    A [label=number shape=diamond]
    B [label="*" shape=oval]
    end [label="" shape=point]
    start -> A
    A -> B
    B -> A
    A -> end
}
```

Такое описание грамматики не позволяет описать лексику, ключевые слова и аксиомы. Поэтому для полного описания грамматики синтаксическими диаграммами Вирта инструменту необходим файл со вспомогательной информацией (Support Grammar Information, SGI):

```
TERMINALS:
    number ::= '[1-9]\d*';
    operation ::= '[\+\*]';
    terminator ::= ','.
```

KEYS: '+'; '*'; ','.

NONTERMINALS:
EXPRESSIONS;
EXPRESSION;
TERM.

AXIOM: EXPRESSIONS.

Файл со вспомогательной информацией SGI представляет собой регулярную форму Бэкуса-Наура без блока правил.

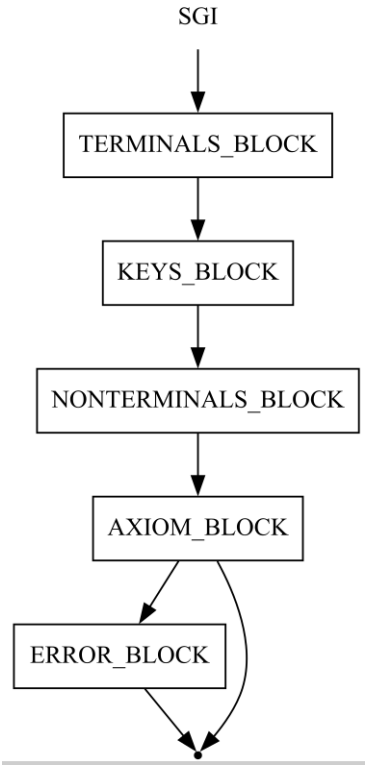

Рис. 4: Диаграмма Вирта файла со вспомогательной информацией о грамматике

2.3 Семантика

Семантикой называется описание правил приписывания смысла синтаксически правильным конструкциям языка.

В инструменте используются следующие способы задания семантики:

1. Техника атрибутных грамматик.
2. Разбор абстрактного синтаксического дерева (Abstract Syntax Tree, AST).

3. Аналог Р-технологии. Инструкции пишутся у дуг в синтаксических диаграммах Вирта.

2.3.1 Атрибутные грамматики

Технику атрибутных грамматик предложил Дональд Кнут в работе "The Genesis of Attribute Grammars". Основная идея - приписать каждому терминалу и нетерминалу дополнительное поле, называемое атрибутом.

В инструменте данная техника реализована следующим образом:

1. Атрибуты для терминалов рассчитываются на этапе послесканера по распознанной строке нетерминала.

2. Атрибуты для нетерминалов рассчитываются после формирования абстрактного синтаксического дерева. Для этого используется ассоциативный массив. Его ключами являются типы нетерминалов, а значениями - вызываемые объекты. Эти объекты должны принимать массив атрибутов дочерних элементов и задавать семантические правила.

Пример. Для языка вычисления выражения над неотрицательными целыми числами с поддержкой операций суммы и произведения можно использовать следующие атрибуты:

1. Для терминалов-чисел атрибутом является значение числа.

2. Для нетерминалов можно задать такие семантические правила:

   - EXPRESSIONS - объединить атрибуты дочерних элементов в список.
   - EXPRESSION - сложить атрибуты дочерних элементов.
   - TERM - перемножить атрибуты дочерних элементов.

Тогда для файла с содержимым "4*4+4"абстрактное синтаксическое дерево с выставленными атрибутыми выглядит так:

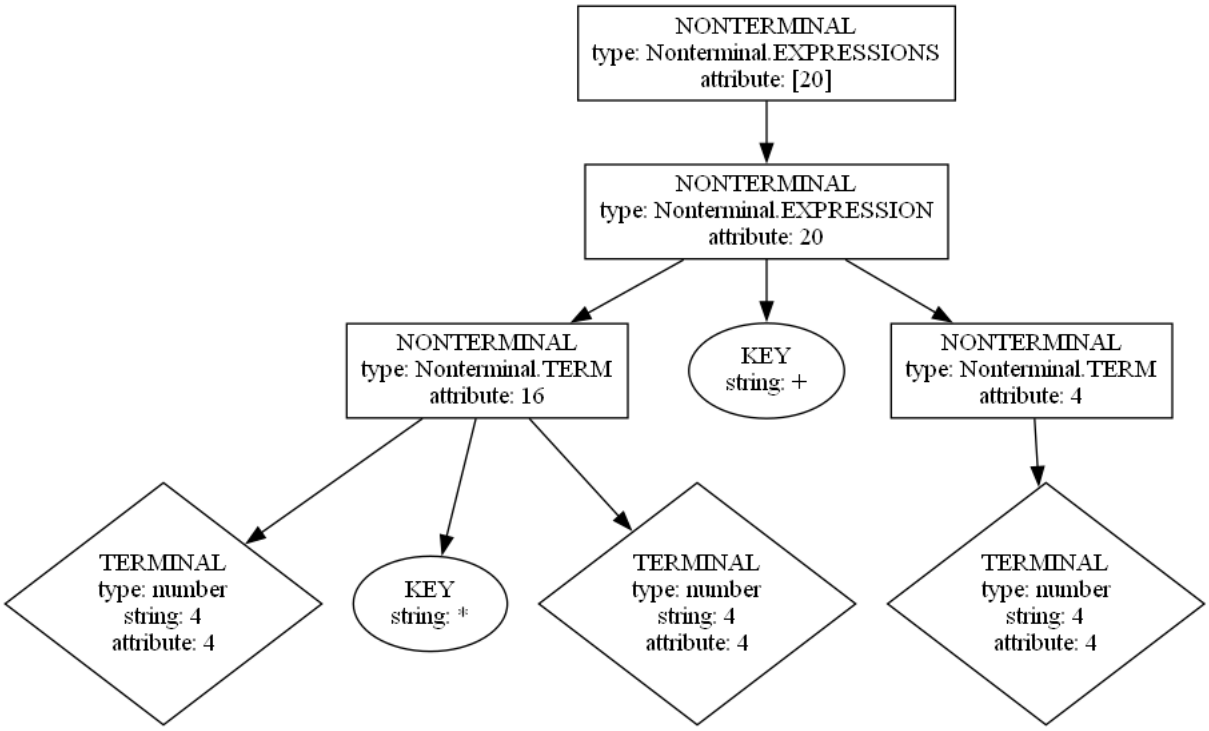

Рис. 5: Абстрактное синтаксическое дерево, в котором вычислены атрибуты

Атрибуты дерева можно использовать при задании семантики.

2.3.2 Разбор абстрактного синтаксического дерева

Инструмент строит абстрактное синтаксическое дерево. Его можно разобрать при помощи программы на языке программирования общего назначения и осуществить нужные автору языка предметной области действия.

Пример. Преобразование РБНФ в синтаксические диаграммы Вирта.
Рассмотрим правило для нетерминала EXPRESSION из примера про язык вычисления выражений.

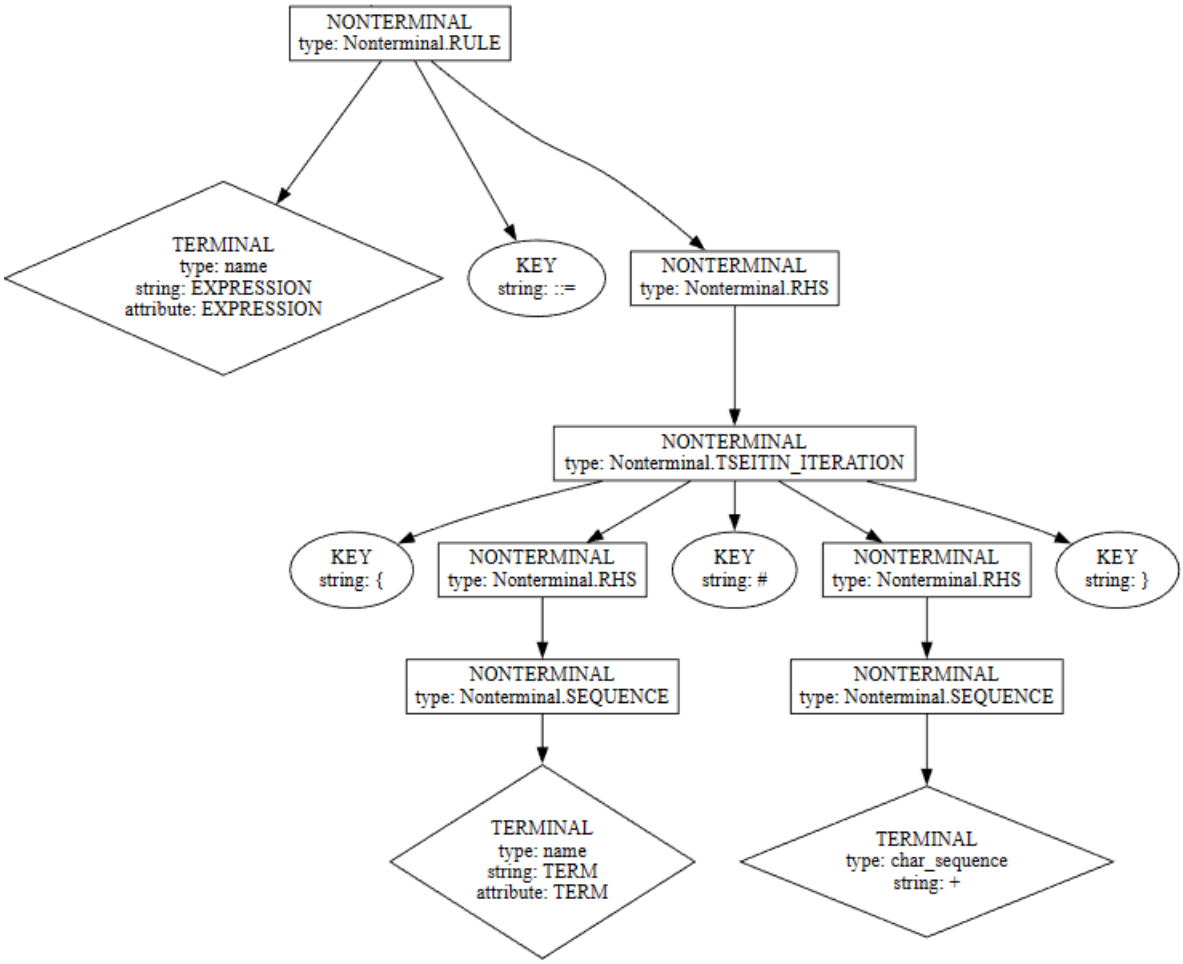

Рис. 6: Разбор нетерминала EXPRESSION в виде РБНФ

При его обходе необходимо, во-первых, запомнить имя нетерминала, в примере - EXPRESSION и, во-вторых, рекурсивно разобрать вершины с терминалами:

1. RHS;
2. SEQUENCE;
3. BRACKETS;
4. OPTIONAL;
5. TSEITIN_ITERATION.

Каждому терминалу соответствует своя конфигурация синтаксической диаграммы Вирта.

2.3.3 Аналог P-технологии

Пока не реализовано

P-технология предложена Вельбицким. Это мощный инструмент визуального программирования.

В инструменте реализован аналог P-технологии для диаграмм Вирта. Дуги в DOT-диаграммах можно нагрузить инструкциями, которые должны быть исполнены при данном переходе.

Референс:

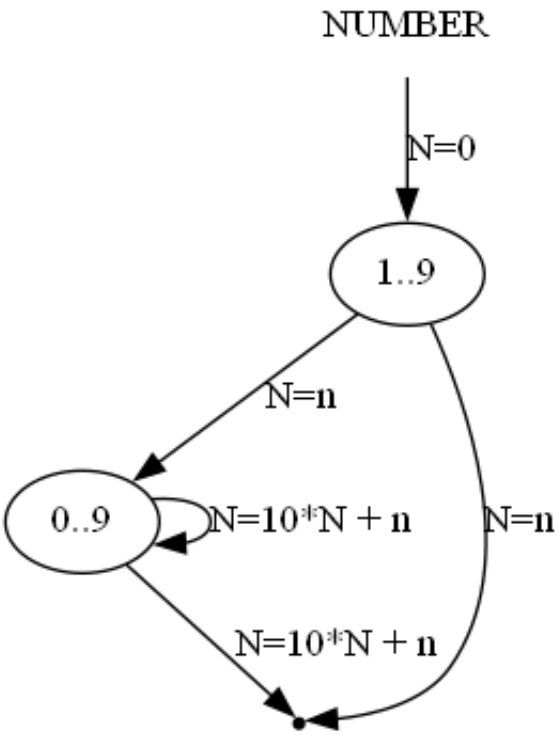

Рис. 7: Иллюстрация P-технологии

3 Архитектура

Инструмент принимает на вход текст программ. Выходом инструмента является результат, который задаётся семантикой заданного языка предметной области.

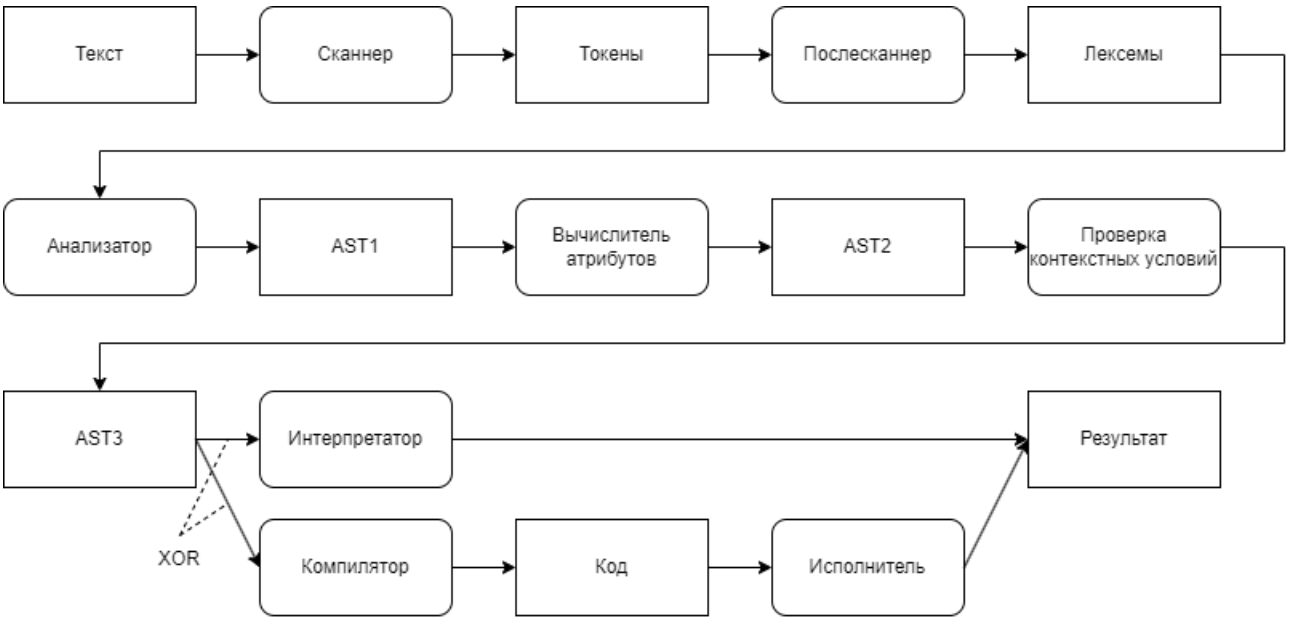

Рис. 8: Архитектура

3.1 Сканер

Модуль преобразования текста в поток токенов. Для этого используется описание лексики из регулярной формы Бэкуса-Наура или файла со вспомогательной информацией о грамматике для диаграмм Вирта.

3.2 Послесканер

Модуль определяется автором языка предметной области. Здесь осуществляется редактирование исходного потока лексем. Стандартные функции:

1. вычисление атрибутов лексем;
2. замена терминалов на ключевые слова;
3. разделение лексем на части (например, если в языке есть ключевые символы "+"и "+= то разумно разделить "+=+"на "+="и "+") происходит жадным образом, но так же это можно и регулировать.

Ключевые слова берутся из описания грамматики.

Иллюстрация работы послесканера для примера из пункта 2.3.1:

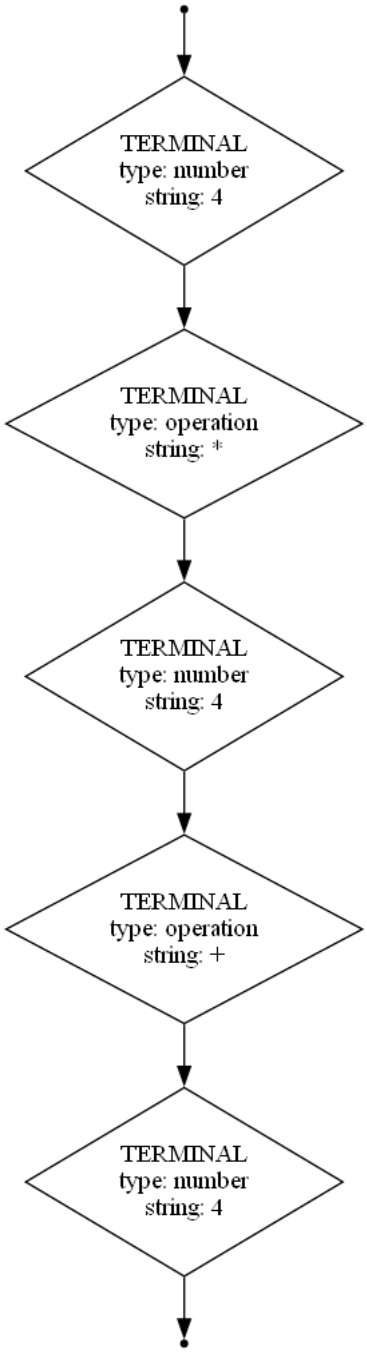

Рис. 9: Поток лексем после сканера

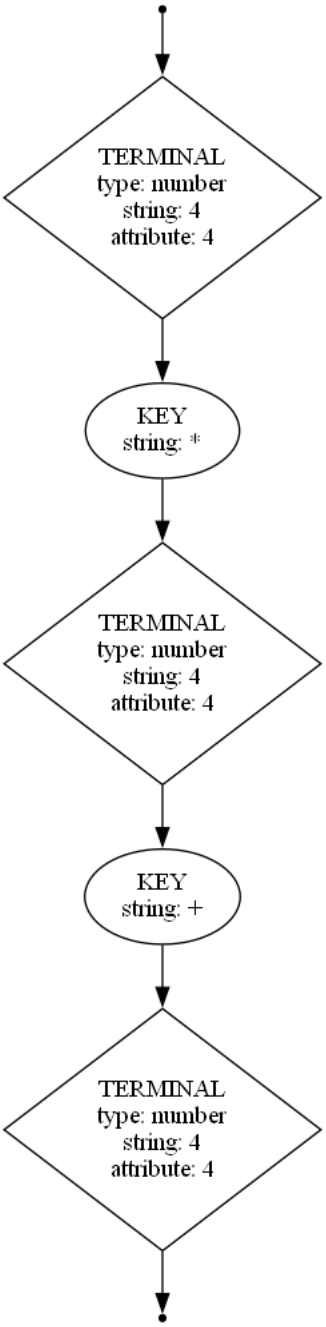

Рис. 10: Поток лексем после послесканера

Различия в двух потоках в следующем:

1. у терминалов типа number по их строковому представлению в коде были вычислены атрибуты, которые являются числами;

2. терминалы типа operation заменены на ключевые символы.

3.3 Анализатор

По описанию грамматики и потоку лексем строит абстрактное синтаксическое дерево.

3.4 Вычислитель атрибутов

Вычисляет атрибуты по абстрактному синтаксическому дереву и ассоциативному массиву. Алгоритм описан в пункте 2.3.1 "Атрибутные грамматики".

3.5 Проверка контекстных условий

Контекстными условиями называются синтаксические правила, которые невозможно или неудобно описать средствами контекстно-свободных грамматик.
Модуль пишется автором языка предметной области.

3.6 Интерпретатор

Выполняет разбор построенного абстрактного синтаксического дерева.

3.7 Компилятор

Будет строить исполняемый код по абстрактному синтаксическому дереву.

3.8 Исполнитель

Исполняет построенный компилятором код.

4 Программа и методика испытаний

4.1 Вычисление значений арифметических выражений

Для демонстрации работы атрибутных грамматик сделан разбор простых арифметических выражений. Этот пример рассматривается в пунктах 2.1-2.3.1.

4.2 Преобразование РБНФ в диаграммы Вирта

Пример осуществляет:

1. Создание шаблонов файлов:
   (a) "dsl_info.py" - содержит перечисления терминалов, нетерминалов и ключей, соответствия "ключ-терминал" и аксиому грамматики;
   (b) "aftescan.py" - шаблон послесканера;
   (c) "attribute_evaluator.py" - шаблон для модуля, содержащего правила выставления грамматик нетерминалов.
2. Создание эквивалентных правилам в регулярной форме Бэкуса-Наура синтаксических диаграмм Вирта в форме DOT-диаграмм.

Семантика задана при помощи механизма атрибутных грамматик и обхода абстрактного синтаксического дерева на языке Python. Блоки терминалов, ключей, нетерминалов и аксиомы разбираются при помощи атрибутных грамматик. Блок правил разбирается обходом соотетсввующего поддерева.

Алгоритм преобразования:

1. Для каждого правила запоминается имя нетерминала, стоящего в левой части - это будет являться именем графа.
2. Далее рекурсивно анализируется правая часть правила (Right Hand Side, RHS).

Далее идут правила для рекурсивного разбора. Для каждого нетерминала строится подграф, который встаривается в подграф дочернего элемента. Итоговая диаграмма Вирта - это подграф дочернего по отношению к нетерминалу RULE нетерминала RHS.

Описания нетерминалов РБНФ приведены в приложениях.

Нетерминал SEQUENCE, записанный в РБНФ как SEQUENCE_CHILD1 ... SEQUENCE_CHIL
где SEQUENCE_CHILDi ∈ {name, char_sequence}, порождает подграф:

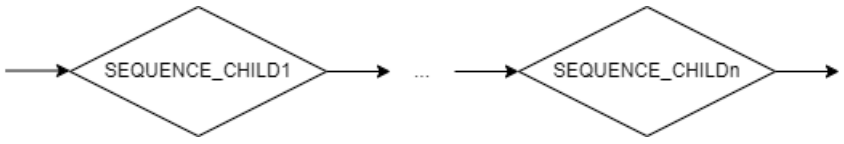

Рис. 11: Подграф для нетерминала SEQUENCE

Нетерминал BRACKETS, записанный в РБНФ как (RHS1 | ... | RHSn) порождает подграф:

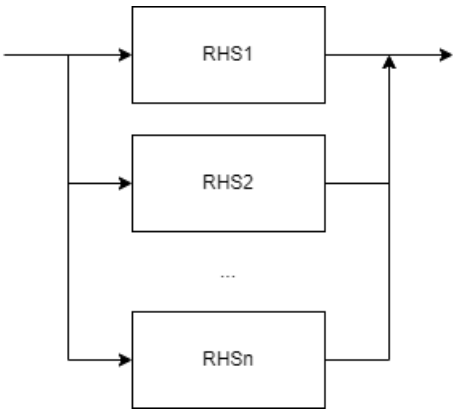

Рис. 12: Подграф для нетерминала BRACKETS

Нетерминал OPTIONAL, записанный в РБНФ как [RHS1 | ... | RHSn] порождает подграф:

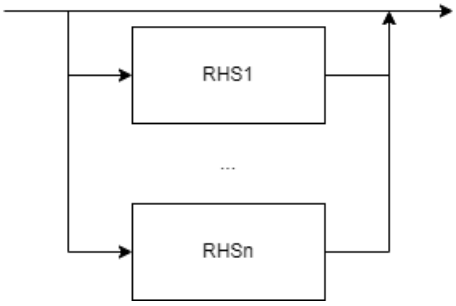

Рис. 13: Подграф для нетерминала OPTIONAL

Нетерминал TSEITIN_ITERATION, записанный в РБНФ как {RHS1 # RHS2}, где и RHS1, и RHS2 могут быть пустыми, порождает подграф:

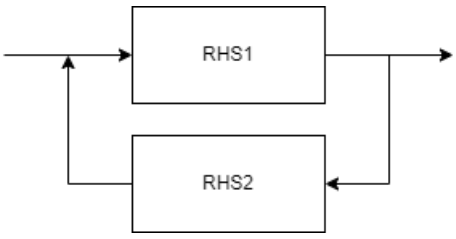

Рис. 14: Подграф для нетерминала TSEITIN_ITERATION

Анализ RHS, BRACKETS, OPTIONAL, TSEITIN_ITERATION происходит рекурсивно - для результата требуется получить подграфы дочерних элементов и встроить их в подграф текущего терминала.

5 Разделение труда

Инструмент разрабатывается тремя студентами группы 5030102/90201. Команда:

1. Воротников Андрей:

   (a) описание грамматики РБНФ на языке РБНФ и диаграммах Вирта;

   (b) реализация модуля задания синтаксиса;

   (c) реализация модуля сканера.

2. Кожевникова Диана:

   (a) метамодель РБНФ;

   (b) диаграмма функциональных требований;

   (c) грамматика языка вычисления арифметических примеров.

3. Павлов Илья:

   (a) диаграмма деятельности;

   (b) грамматика языка CIAO.

6 Результаты

В процессе разработки инструмента получены следующие результаты:

1. Реализован анализатор, который использует описание грамматики в форме синтаксических диаграмм Вирта в виде DOT-диаграмм.

2. Реализован механизм атрибутных грамматик.

3. Описана грамматика регулярной формы Бэкуса-Наура.

4. Реализован перевод грамматики в форме регулярной формы Бэкуса-Наура в диаграммы Вирта.

5. Реализован язык вычислений выражений, как иллюстация языка, в котором достаточно техники атрибутных грамматик для описания семантики.

К реализации планируется:

1. Возможность использования аналога Р-технологии для задания семантики.

2. Возможность задания автоматного объекта на языке CIAO для задания семантики.

3. Поддержка блока ошибок.

7 Выводы

Разработан инструмент реализации языка предметной области. Для реализации языка инструмент требует описать 3 части языка: лексику, синтаксис и семантику.

Для описания лексики используется широко распространнёный механизм регулярных выражений.

Инструмент использует описание языка в 2 формах:

1. В форме синтаксических диаграмм Вирта.
2. В форме регулярной формы Бэкуса-Наура.

Для задания семантики в инструменте поддерживаются:

1. Механизм атрибутных грамматик.
2. Обход абстрактного синтаксического дерева с использованием языка общего назначения.

Механизм атрибутных грамматик можно использовать как вспомогательный. Атрибуты можно учитывать при обходе дерева, что упрощяет описание семантики.

8 Приложения

8.1 Диаграммы UML

8.1.1 Функциональные требования

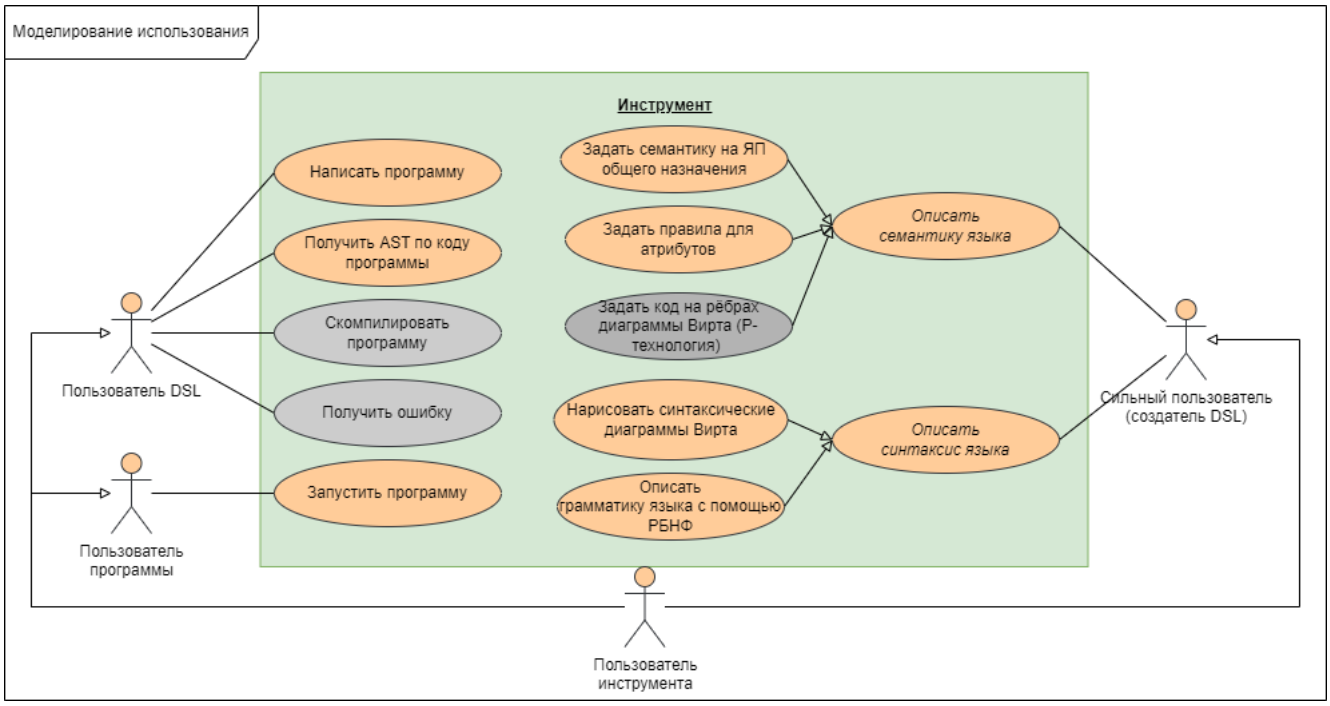

Рис. 15: Функциональные требования

8.1.2 Метамодель языка регулярной формы Бэкуса-Наура

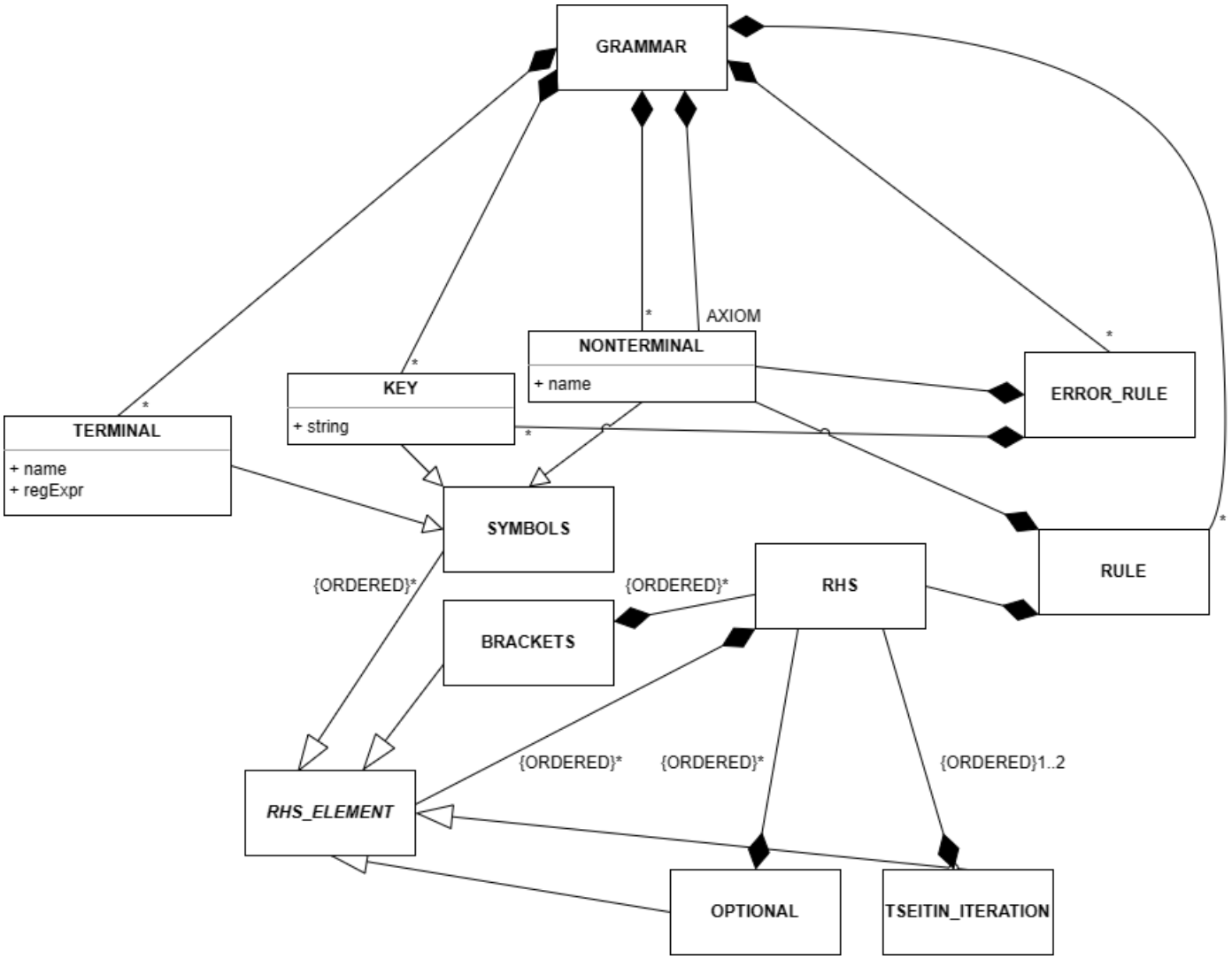

Рис. 16: Метамодель РБНФ

8.1.3 Диаграмма деятельности инструмента

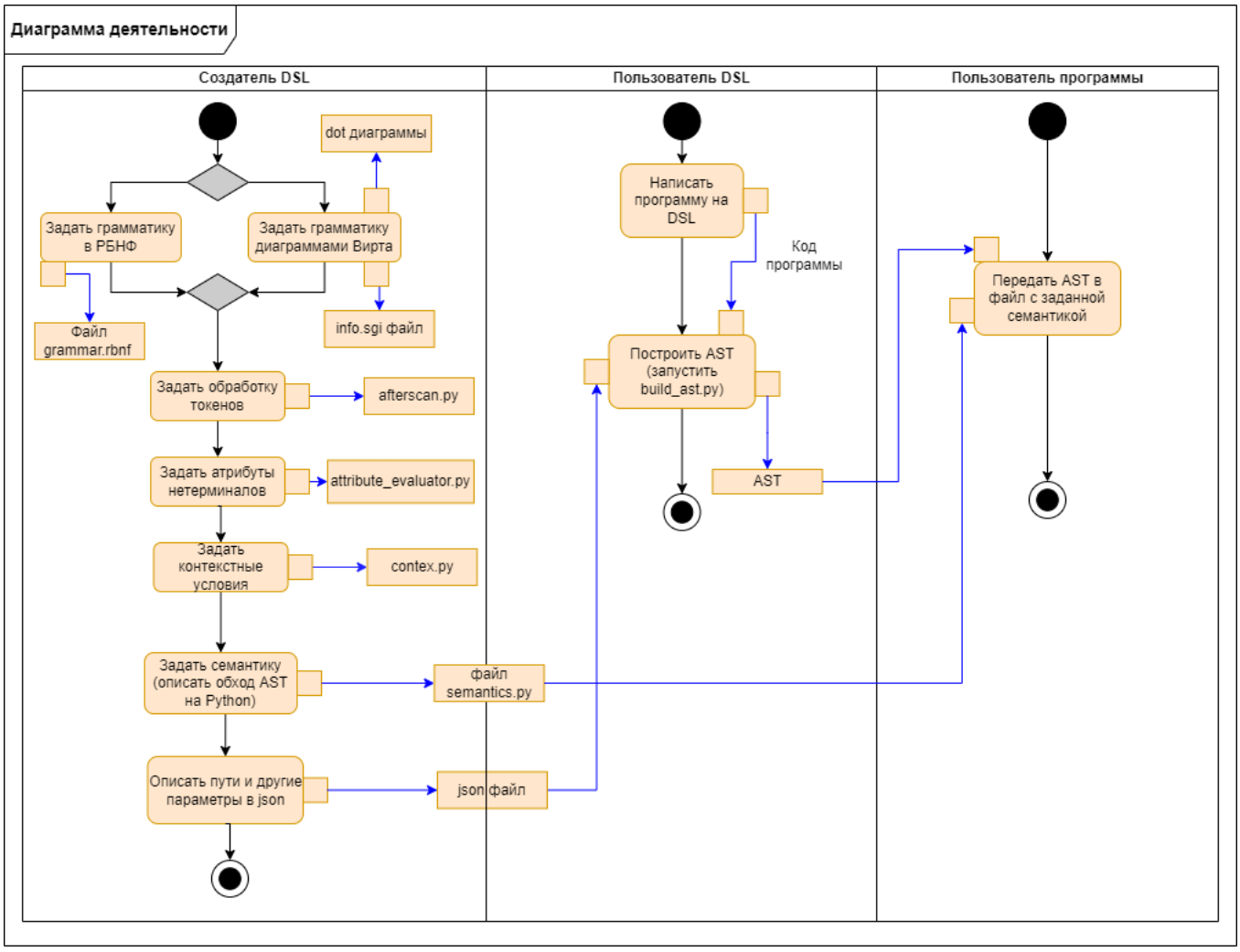

Рис. 17: Диаграмма деятельности инструмента

8.2 Описание регулярной формы Бэкуса-Наура на языке регулярной формы Бэкуса-Наура

TERMINALS:

name ::= '[\w\D][\w]*';

char_sequence ::= '[\W\S^']+';

string ::= ''(\\.|[^\\']+)*''.

KEYS:

'TERMINALS'; 'KEYS'; 'NONTERMINALS'; 'AXIOM'; 'RULES'; 'ERRORS';
'.'; ':'; '::='; ';'; '('; ')'; '|'; '['; ']'; '{'; '}'; '#'.

NONTERMINALS:

```
        GRAMMAR;
        TERMINALS_BLOCK;
        KEYS_BLOCK;
        NONTERMINALS_BLOCK;
        AXIOM_BLOCK;
        ERROR_BLOCK;
        RULES_BLOCK;
        RULE;
        RHS;
        SEQUENCE;
        BRACKETS;
        OPTIONAL;
        TSEITIN_ITERATION.

AXIOM: GRAMMAR.

ERRORS:
        TERMINALS_BLOCK '.';
        KEYS_BLOCK '.';
        NONTERMINALS_BLOCK ' .';
        AXIOM_BLOCK '.';
        ERROR_BLOCK '.';
        RULES_BLOCK '.';
        RULE ';' | '.';
        RHS ';' | '.';
        SEQUENCE ';' | '.';
        BRACKETS ';' | '.';
        OPTIONAL ';' | '.';
        TSEITIN_ITERATION ';' | '.'.

RULES:
        GRAMMAR ::=
            TERMINALS_BLOCK KEYS_BLOCK NONTERMINALS_BLOCK
            AXIOM_BLOCK [ERROR_BLOCK] RULES_BLOCK;
        TERMINALS_BLOCK ::= TERMINALS: {name ::= string # ;}.;
        KEYS_BLOCK ::= KEYS: {string # ;}.;
        NONTERMINALS_BLOCK ::= NONTERMINALS: {name # ;}.;
        AXIOM_BLOCK ::= AXIOM: name.;
        ERROR_BLOCK ::= ERRORS: {name {string # |} # ;}.;
        RULES_BLOCK ::= RULES: {RULE # ;}.;
```

RULE ::= name ::= RHS;
RHS ::= {(SEQUENCE | BRACKETS | OPTIONAL | TSEITIN_ITERATION)};
SEQUENCE ::= {(name | char_sequence)};
BRACKETS ::= ({RHS # |});
OPTIONAL ::= [{RHS # |}];
TSEITIN_ITERATION ::= {[RHS] [# RHS]}.

8.3 Описание регулярной формы Бэкуса-Наура синтаксическими диаграммами Вирта

8.3.1 Аксиома описания РБНФ

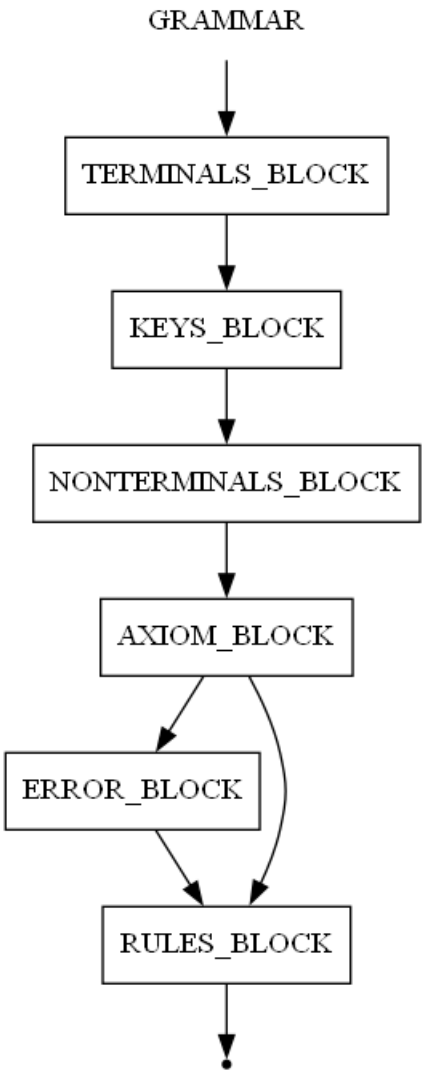

Рис. 18: Аксиома описания РБНФ

Листинг кода на языке DOT:

```
digraph GRAMMAR {
```

```
    start [label=GRAMMAR shape=plaintext]
    A [label=TERMINALS_BLOCK shape=box]
    B [label=KEYS_BLOCK shape=box]
    C [label=NONTERMINALS_BLOCK shape=box]
    D [label=AXIOM_BLOCK shape=box]
    E [label=ERROR_BLOCK shape=box]
    F [label=RULES_BLOCK shape=box]
    end [label="" shape=point]
    start -> A
    A -> B
    B -> C
    C -> D
    D -> E
    D -> F
    E -> F
    F -> end
}
```

8.3.2 Блок терминалов РБНФ

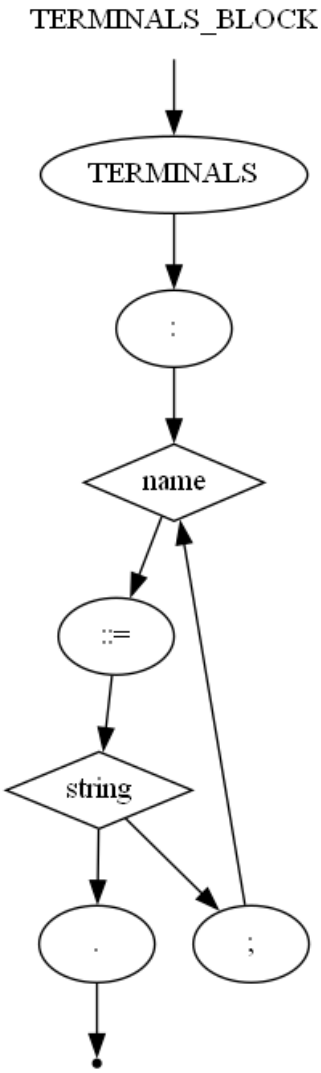

Рис. 19: Блок терминалов РБНФ

Листинг кода на языке DOT:

```
digraph TERMINALS_BLOCK {
    start [label=TERMINALS_BLOCK shape=plaintext]
    A [label=TERMINALS shape=oval]
    B [label=":" shape=oval]
    C [label=name shape=diamond]
    D [label="::=" shape=oval]
    E [label=string shape=diamond]
    F [label=";" shape=oval]
    G [label="." shape=oval]
    end [label="" shape=point]
    start -> A
    A -> B
```

```
    B -> C
    C -> D
    D -> E
    E -> F
    F -> C
    E -> G
    G -> end
}
```

8.3.3 Блок ключей РБНФ

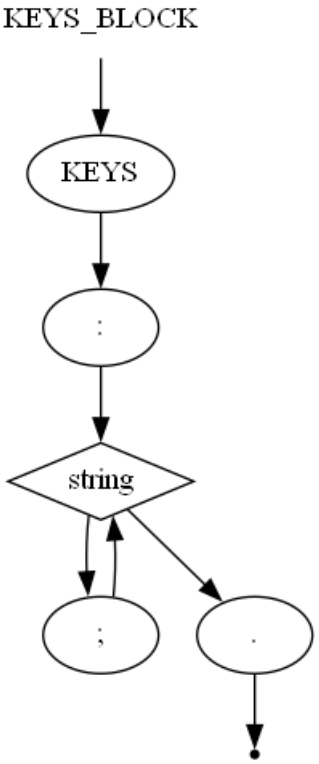

Рис. 20: Блок ключей РБНФ

Листинг кода на языке DOT:

```
digraph KEYS_BLOCK {
    start [label=KEYS_BLOCK shape=plaintext]
    A [label=KEYS shape=oval]
    B [label=":" shape=oval]
    C [label=string shape=diamond]
    D [label=";" shape=oval]
    E [label="." shape=oval]
    end [label="" shape=point]
    start -> A
```

```
    A -> B
    B -> C
    C -> D
    C -> E
    D -> C
    E -> end
}
```

8.3.4 Блок нетерминалов РБНФ

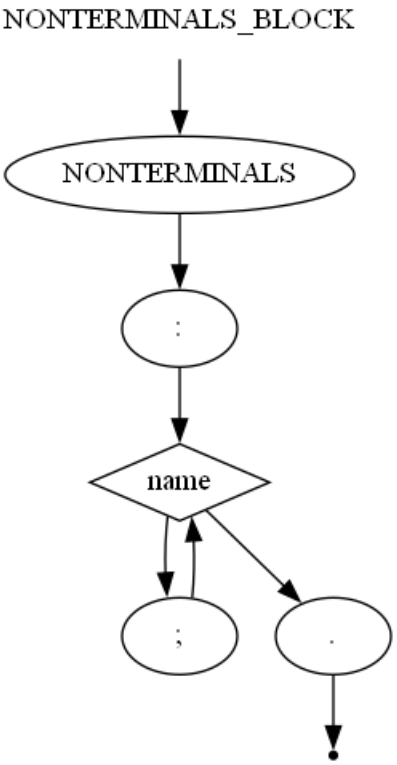

Рис. 21: Блок нетерминалов РБНФ

Листинг кода на языке DOT:

```
digraph NONTERMINALS_BLOCK {
    start [label=NONTERMINALS_BLOCK shape=plaintext]
    A [label=NONTERMINALS shape=oval]
    B [label=":" shape=oval]
    C [label=name shape=diamond]
    D [label=";" shape=oval]
    E [label="." shape=oval]
    end [label="" shape=point]
    start -> A
    A -> B
```

```
    B -> C
    C -> D
    C -> E
    D -> C
    E -> end
}
```

8.3.5 Блок аксиом РБНФ

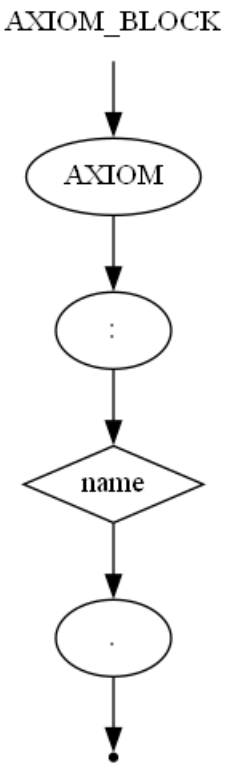

Рис. 22: Блок аксиом РБНФ

Листинг кода на языке DOT:

```
digraph AXIOM_BLOCK {
    start [label=AXIOM_BLOCK shape=plaintext]
    A [label=AXIOM shape=oval]
    B [label=":" shape=oval]
    C [label=name shape=diamond]
    D [label="." shape=oval]
    end [label="" shape=point]
    start -> A
    A -> B
    B -> C
    C -> D
```

```
    D -> end
}
```

8.3.6 Блок ошибок РБНФ

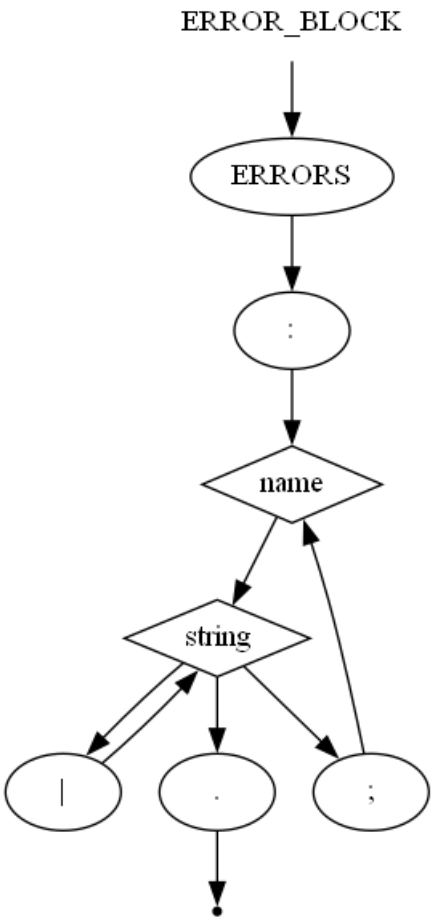

Рис. 23: Блок ошибок РБНФ

Листинг кода на языке DOT:

```
digraph ERROR_BLOCK {
    start [label=ERROR_BLOCK shape=plaintext]
    A [label=ERRORS shape=oval]
    B [label=":" shape=oval]
    C [label=name shape=diamond]
    D [label=string shape=diamond]
    E [label="|" shape=oval]
    F [label=";" shape=oval]
    G [label="." shape=oval]
    end [label="" shape=point]
    start -> A
    A -> B
```

```
    B -> C
    C -> D
    D -> G
    D -> E
    E -> D
    D -> F
    F -> C
    G -> end
}
```

8.3.7 Блок правил РБНФ

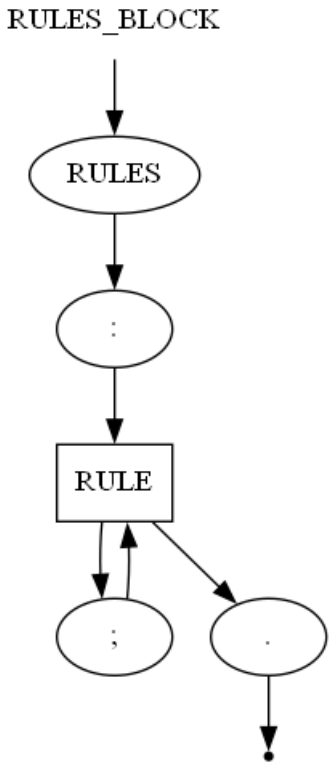

Рис. 24: Блок правил РБНФ

Листинг кода на языке DOT:

```
digraph RULES_BLOCK {
    start [label=RULES_BLOCK shape=plaintext]
    A [label=RULES shape=oval]
    B [label=":" shape=oval]
    C [label=RULE shape=box]
    D [label=";" shape=oval]
    E [label="." shape=oval]
    end [label="" shape=point]
```

```
    start -> A
    A -> B
    B -> C
    C -> D
    C -> E
    D -> C
    E -> end
}
```

8.3.8 Правило РБНФ

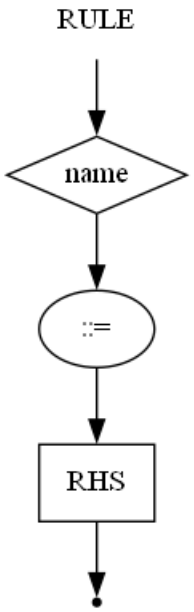

Рис. 25: Правило РБНФ

Листинг кода на языке DOT:

```
digraph RULE {
    start [label=RULE shape=plaintext]
    A [label=name shape=diamond]
    B [label="::=" shape=oval]
    C [label=RHS shape=box]
    end [label="" shape=point]
    start -> A
    A -> B
    B -> C
    C -> end
}
```

8.3.9 Правая часть правил РБНФ

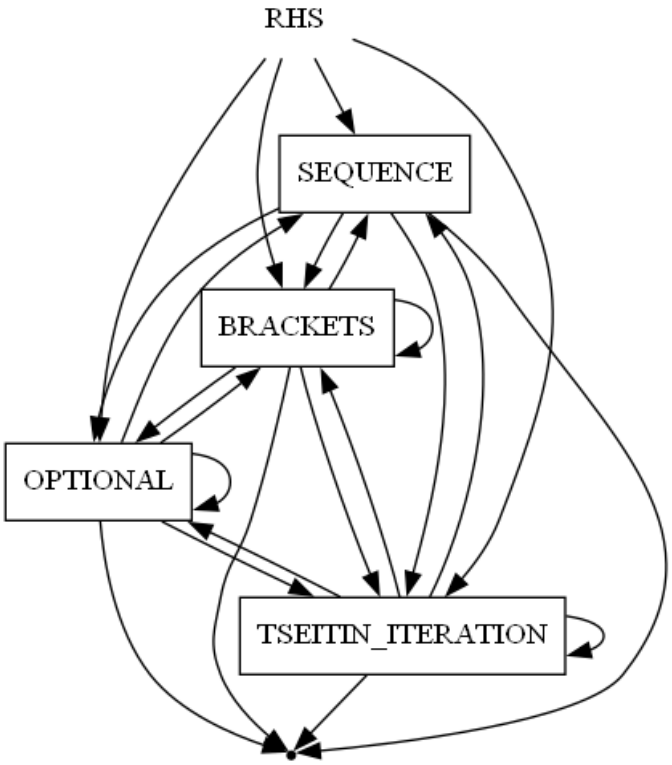

Рис. 26: Правая часть правил РБНФ

Листинг кода на языке DOT:

```
digraph RHS {
    start [label=RHS shape=plaintext]
    A [label=SEQUENCE shape=box]
    B [label=BRACKETS shape=box]
    C [label=OPTIONAL shape=box]
    D [label=TSEITIN_ITERATION shape=box]
    end [label="" shape=point]
    start -> A
    start -> B
    start -> C
    start -> D
    A -> B
    A -> C
    A -> D
    A -> end
    B -> A
    B -> B
    B -> C
```

```
    B -> D
    B -> end
    C -> A
    C -> B
    C -> C
    C -> D
    C -> end
    D -> A
    D -> B
    D -> C
    D -> D
    D -> end
}
```

8.3.10 Последовательность значений РБНФ

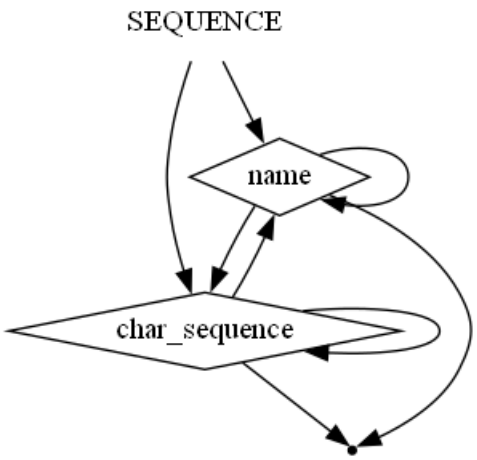

Рис. 27: Последовательность значений РБНФ

Листинг кода на языке DOT:

```
digraph SEQUENCE {
    start [label=SEQUENCE shape=plaintext]
    A [label=name shape=diamond]
    B [label=char_sequence shape=diamond]
    end [label="" shape=point]
    start -> A
    start -> B
    A -> A
    A -> B
    B -> A
```

```
    B -> B
    A -> end
    B -> end
}
```

8.3.11 Скобки в правилах РБНФ

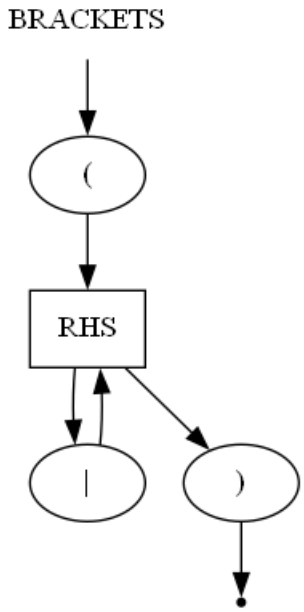

Рис. 28: Скобки в правилах РБНФ

Листинг кода на языке DOT:

```
digraph BRACKETS {
    start [label=BRACKETS shape=plaintext]
    A [label="(" shape=oval]
    B [label=RHS shape=box]
    C [label="|" shape=oval]
    D [label=")" shape=oval]
    end [label="" shape=point]
    start -> A
    A -> B
    B -> C
    B -> D
    C -> B
    D -> end
}
```

8.3.12 Квадратные скобки в правилах РБНФ

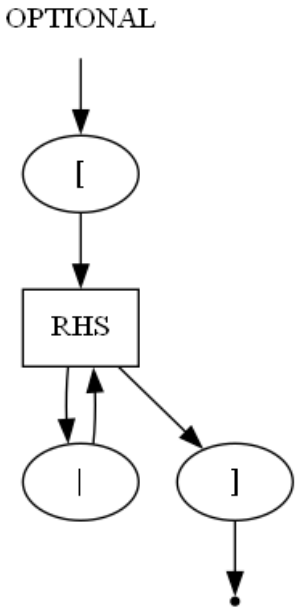

Рис. 29: Квадратные скобки в правилах РБНФ

Листинг кода на языке DOT:

```
digraph OPTIONAL {
    start [label=OPTIONAL shape=plaintext]
    A [label="[" shape=oval]
    B [label=RHS shape=box]
    C [label="|" shape=oval]
    D [label="]" shape=oval]
    end [label="" shape=point]
    start -> A
    A -> B
    B -> C
    B -> D
    C -> B
    D -> end
}
```

8.3.13 Итерация Цейтина в правилах РБНФ

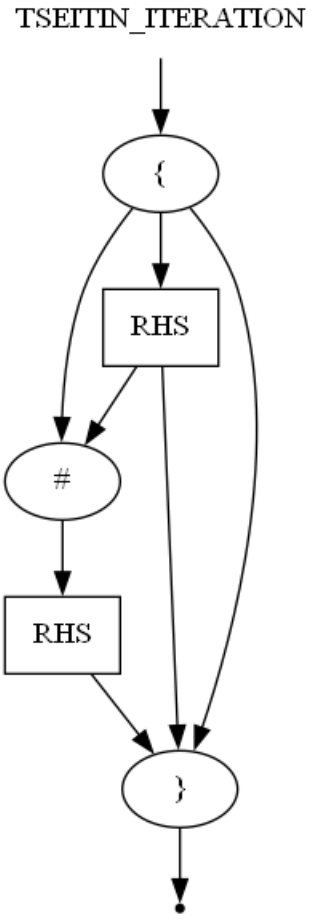

Рис. 30: Итерация Цейтина в правилах РБНФ

Листинг кода на языке DOT:

```
digraph TSEITIN_ITERATION {
    start [label=TSEITIN_ITERATION shape=plaintext]
    A [label="{" shape=oval]
    B [label=RHS shape=box]
    C [label="#" shape=oval]
    D [label=RHS shape=box]
    E [label="}" shape=oval]
    end [label="" shape=point]
    start -> A
    A -> B
    A -> C
    A -> E
    B -> C
    B -> E
    C -> D
```

```
    D -> E
    E -> end
}
```

8.4 Грамматика языка CIAO

8.4.1 Вспомогательная информация о грамматике

```
TERMINALS:
    number ::= '[0-9]+(.[0-9]*)?';
    name ::= '[\w\D][\w]*';
    string ::= '"(\\.|[^\\"]+)*"';
    code ::= '\{[^\}]*\}';
    char_key ::= '[\/\(\),.:(->)(:=)]';

KEYS: 'VAR','REQUIRED','PROVIDED','INNER','STATE','else','/', '(', ')', ',', '.', ':', '->',
':='.

NONTERMINALS:
    CIAO;
    AUTOMATA_OBJECT;
    VAR_BLOCK;
    REQUIRED_BLOCK;
    PROVIDED_BLOCK;
    INNER_BLOCK;
    FUNCTION;
    STATE_BLOCK;
    TRANSITION_DESCRIPTION.

AXIOM: CIAO.
```

8.4.2 Аксиома языка CIAO

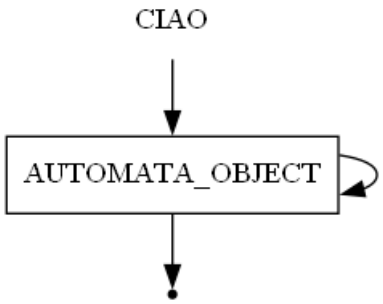

Рис. 31: Аксиома описания CIAO

Листинг кода на языке DOT:

```
digraph CIAO {
    start [label=CIAO shape=plaintext]
    A [label=AUTOMATA_OBJECT shape=box]
    end [label="" shape=point]
    start -> A
    A -> A
    A -> end
}
```

8.4.3 Автоматный объект языка CIAO

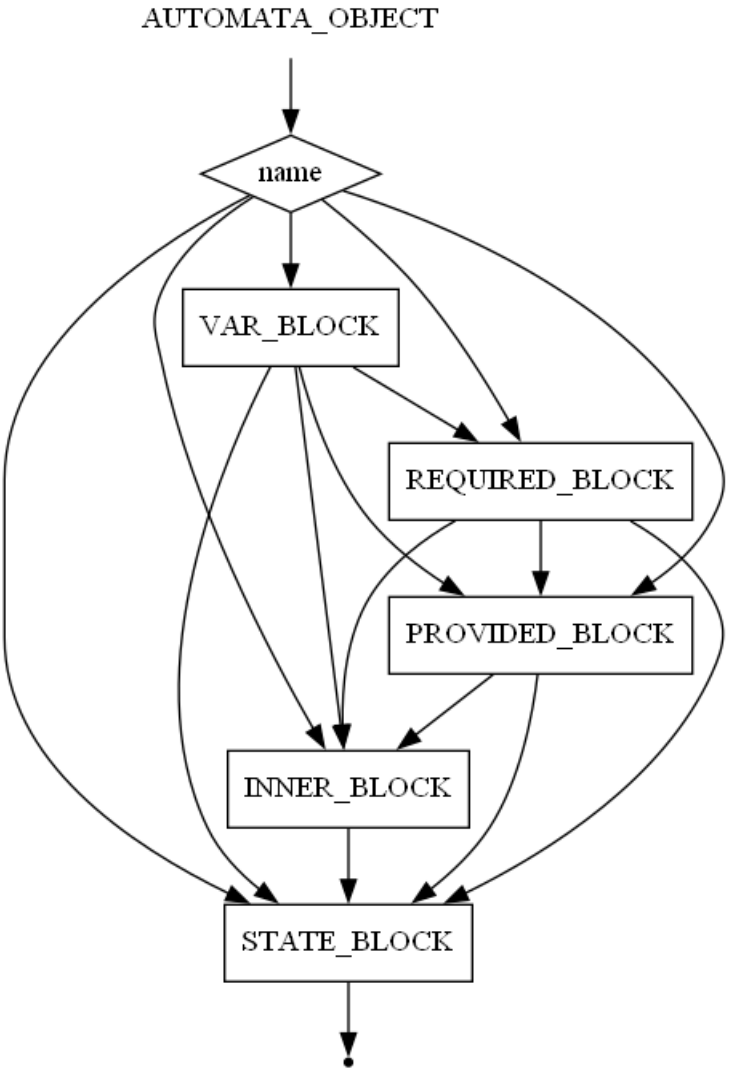

Рис. 32: Автоматный объект языка CIAO

Листинг кода на языке DOT:

```
digraph AUTOMATA_OBJECT {
    start [label=AUTOMATA_OBJECT shape=plaintext]
    A [label=name shape=diamond]
    B [label=VAR_BLOCK shape=box]
    C [label=REQUIRED_BLOCK shape=box]
    D [label=PROVIDED_BLOCK shape=box]
    E [label=INNER_BLOCK shape=box]
    F [label=STATE_BLOCK shape=box]
    end [label="" shape=point]
    start -> A
    A -> B
    A -> C
```

```
    A -> D
    A -> E
    A -> F
    B -> C
    B -> D
    B -> E
    B -> F
    C -> D
    C -> E
    C -> F
    D -> E
    D -> F
    E -> F
    F -> end
}
```

8.4.4 Блок переменных автоматного объекта

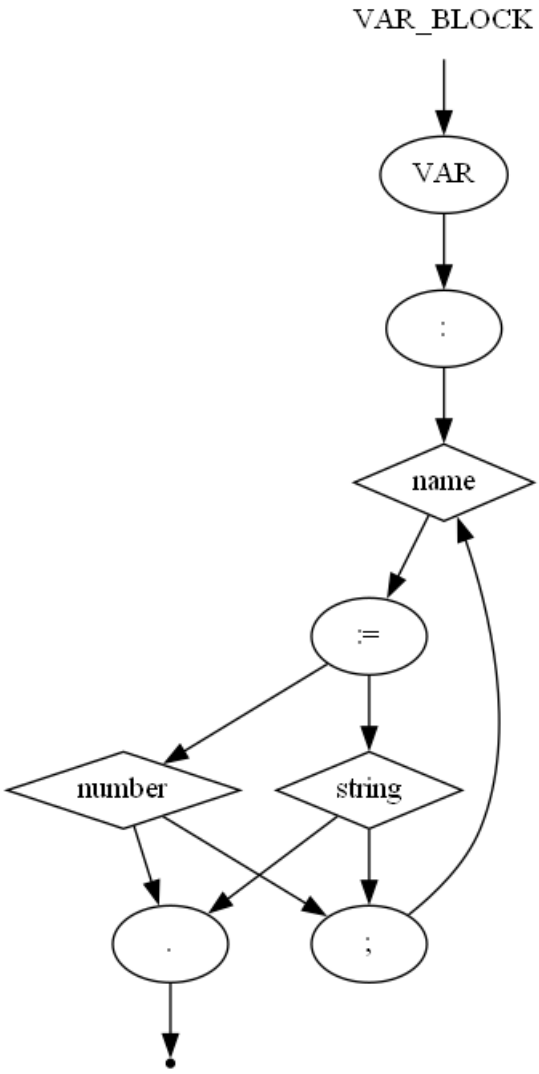

Рис. 33: Блок переменных автоматного объекта

Листинг кода на языке DOT:

```
digraph VAR_BLOCK {
    start [label=VAR_BLOCK shape=plaintext]
    A [label=VAR shape=oval]
    B [label=":" shape=oval]
    C [label=name shape=diamond]
    D [label=":=" shape=oval]
    E [label=number shape=diamond]
    F [label=string shape=diamond]
    G [label=";" shape=oval]
    H [label="." shape=oval]
    end [label="" shape=point]
    start -> A
```

```
    A -> B
    B -> C
    C -> D
    D -> E
    D -> F
    E -> G
    E -> H
    F -> G
    F -> H
    G -> C
    H -> end
}
```

8.4.5 Блок требуемых интрейфейсов автоматного объекта

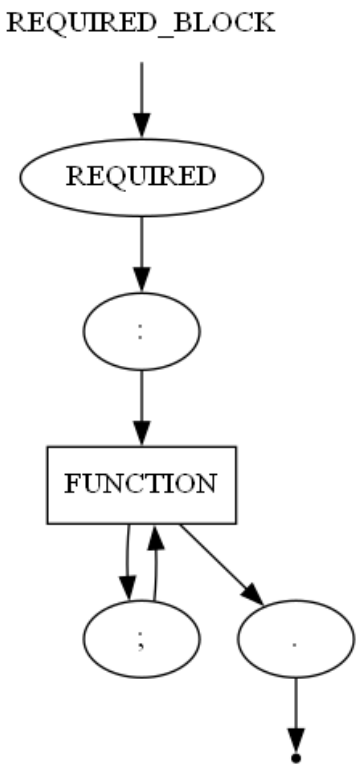

Рис. 34: Блок требуемых интрейфейсов автоматного объекта

Листинг кода на языке DOT:

```
digraph REQUIRED_BLOCK {
    start [label=REQUIRED_BLOCK shape=plaintext]
    A [label=REQUIRED shape=oval]
    B [label=":" shape=oval]
    C [label=FUNCTION shape=box]
```

```
    G [label=";" shape=oval]
    H [label="." shape=oval]
    end [label="" shape=point]
    start -> A
    A -> B
    B -> C
    C -> G
    C -> H
    G -> C
    H -> end
}
```

8.4.6 Блок предоставляемых интрейфейсов автоматного объекта

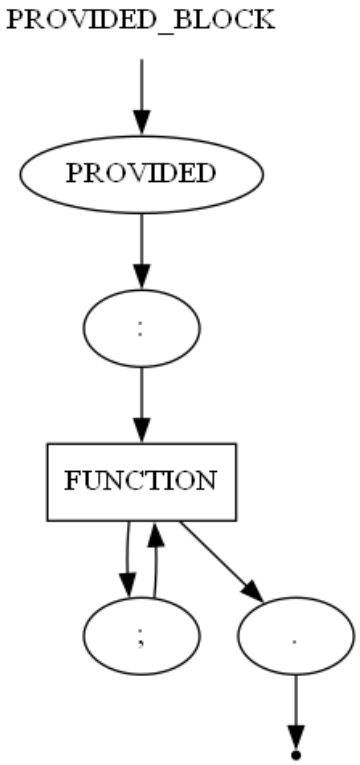

Рис. 35: Блок предоставляемых интрейфейсов автоматного объекта

Листинг кода на языке DOT:

```
digraph PROVIDED_BLOCK {
    start [label=PROVIDED_BLOCK shape=plaintext]
    A [label=PROVIDED shape=oval]
    B [label=":" shape=oval]
    C [label=FUNCTION shape=box]
    G [label=";" shape=oval]
```

```
    H [label="." shape=oval]
    end [label="" shape=point]
    start -> A
    A -> B
    B -> C
    C -> G
    C -> H
    G -> C
    H -> end
}
```

8.4.7 Блок внутренних методов автоматного объекта

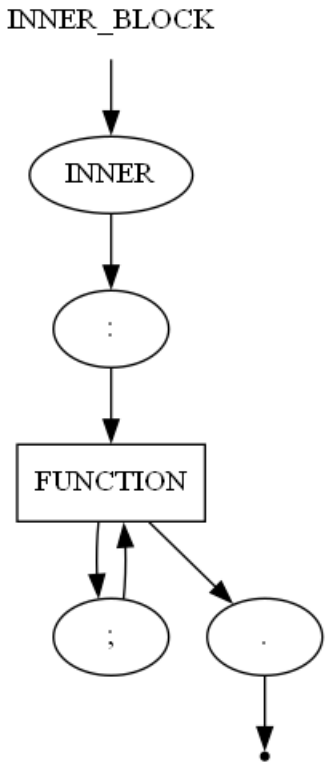

Рис. 36: Блок внутренних методов автоматного объекта

Листинг кода на языке DOT:

```
digraph INNER_BLOCK {
    start [label=INNER_BLOCK shape=plaintext]
    A [label=INNER shape=oval]
    B [label=":" shape=oval]
    C [label=FUNCTION shape=box]
    G [label=";" shape=oval]
    H [label="." shape=oval]
```

```
    end [label="" shape=point]
    start -> A
    A -> B
    B -> C
    C -> G
    C -> H
    G -> C
    H -> end
}
```

8.4.8 Объявление функции

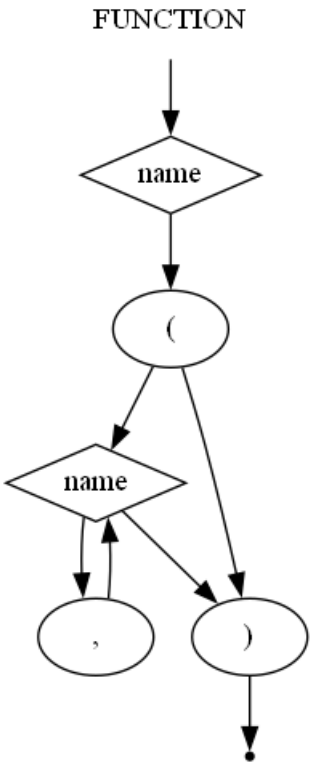

Рис. 37: Объявление функции

Листинг кода на языке DOT:

```
digraph FUNCTION {
    start [label=FUNCTION shape=plaintext]
    A [label=name shape=diamond]
    B [label="(" shape=oval]
    C [label=name shape=diamond]
    D [label="," shape=oval]
    E [label=")" shape=oval]
    end [label="" shape=point]
```

```
    start -> A
    A -> B
    B -> C
    B -> E
    C -> D
    C -> E
    D -> C
    E -> end
}
```

8.4.9 Блок состояний автоматного объекта

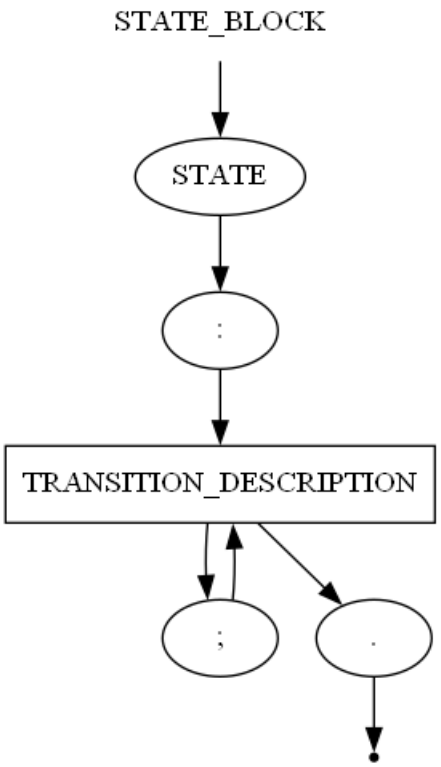

Рис. 38: Блок состояний автоматного объекта

Листинг кода на языке DOT:

```
digraph STATE_BLOCK {
    start [label=STATE_BLOCK shape=plaintext]
    A [label=STATE shape=oval]
    B [label=":" shape=oval]
    C [label=TRANSITION_DESCRIPTION shape=box]
    D [label=";" shape=oval]
    E [label="." shape=oval]
    end [label="" shape=point]
```

```
    start -> A
    A -> B
    B -> C
    C -> D
    C -> E
    D -> C
    E -> end
}
```

8.4.10 Задание переходов автоматного объекта

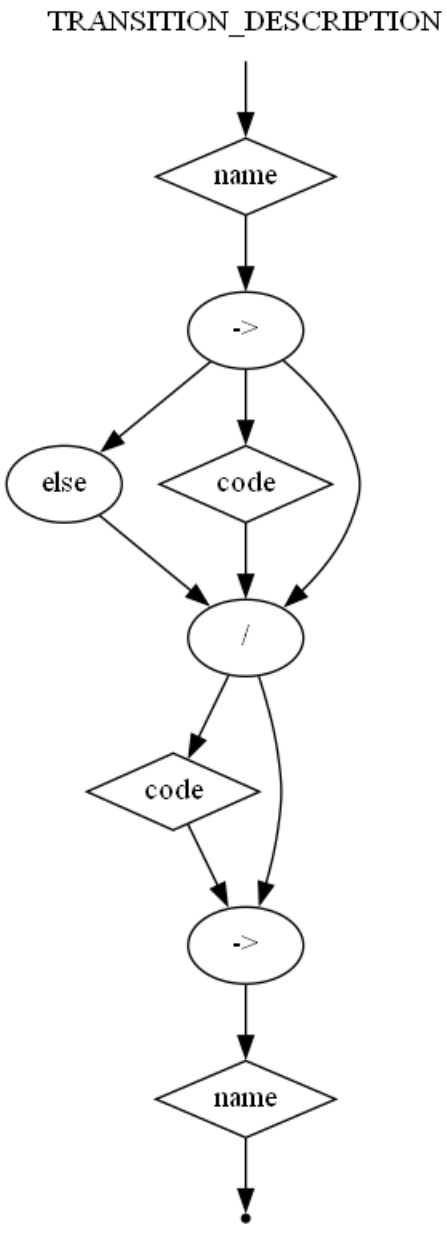

Рис. 39: Задание переходов автоматного объекта

Листинг кода на языке DOT:

```
digraph TRANSITION_DESCRIPTION {
    start [label=TRANSITION_DESCRIPTION shape=plaintext]
    A [label=name shape=diamond]
    B [label="->" shape=oval]
    C [label=else shape=oval]
    D [label=code shape=diamond]
    E [label="/" shape=oval]
    F [label=code shape=diamond]
    G [label="->" shape=oval]
    H [label=name shape=diamond]
    end [label="" shape=point]
    start -> A
    A -> B
    B -> C
    B -> D
    B -> E
    C -> E
    D -> E
    E -> F
    E -> G
    F -> G
    G -> H
    H -> end
}
```

8.5 Ссылка на репозиторий

Репозиторий инструмента на GitHub, URL: `https://github.com/aVorotnikov/dsl_generator`.