Санкт-Петербургский политехнический университет Петра Великого
Физико-механический институт
Высшая школа прикладной математики и 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 - создать учебное инструментальное средство 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, и специалисту предметной области не придётся изучать язык программирования общего назначения.

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

2. Составные части языка и спросбы 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 грамматика.

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 расширенная форма Бэкуса-Наура. Для инструмента было принято решение расширить этот способ задания грамматики.

В описании грамматики включены:

1. описание лексики;
2. список ключевых слов и выражений;

3. список нетерминалов;

4. аксиому грамматики;

5. набор правил, задающих поведение при наличии ошибки.

Для описания правил используется:

1. альтернация;

2. опциональные последовательности;

3. итерация Цейтина.

Описание грамматики принятой регулярной формы Бэкуса-Науэра можно найти в приложениях.

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

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

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

NONTERMINALS:
  EXPRESSIONS;
  EXPRESSION;
  TERM.

AXIOM: EXPRESSIONS.

RULES:
  EXPRESSIONS ::= { EXPRESSION # , };
```

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

Примечание. Красным выделены ключевые символы и слова регулярной формы Бэкуса-Наура.

2.2.2. Диаграммы Вирта.

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

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

1. Граф должен быть направленным, в терминах DOT - digraph.
2. Все рёбра направленные.
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. 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.

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

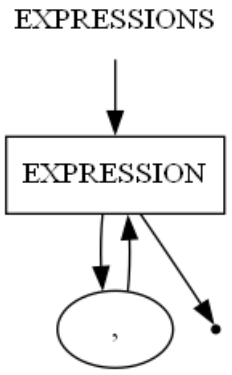

Рис. 1. Диаграмма Вирта набора выражений

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
}
```

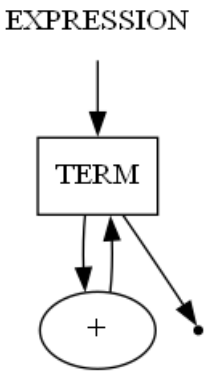

Рис. 2. Диаграмма Вирта выражения

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
}
```

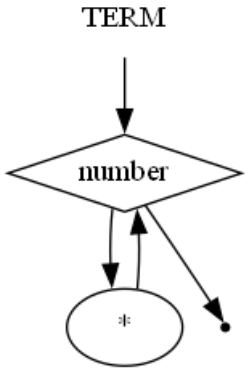

Рис. 3. Диаграмма Вирта слагаемого

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:

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

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

NONTERMINALS:
  EXPRESSIONS;
  EXPRESSION;
  TERM.

AXIOM: EXPRESSIONS.
```

Файл со вспомогательной информацией представляет из себя регулярную форму Бэкуса-Наура без блока правил.

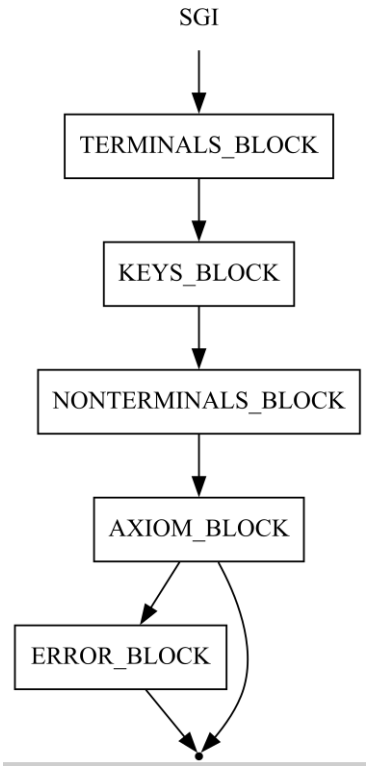

Рис. 4. Диаграмма Вирта файла со вспомогательной информацией о грамматике

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 P-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 послесканнера 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:

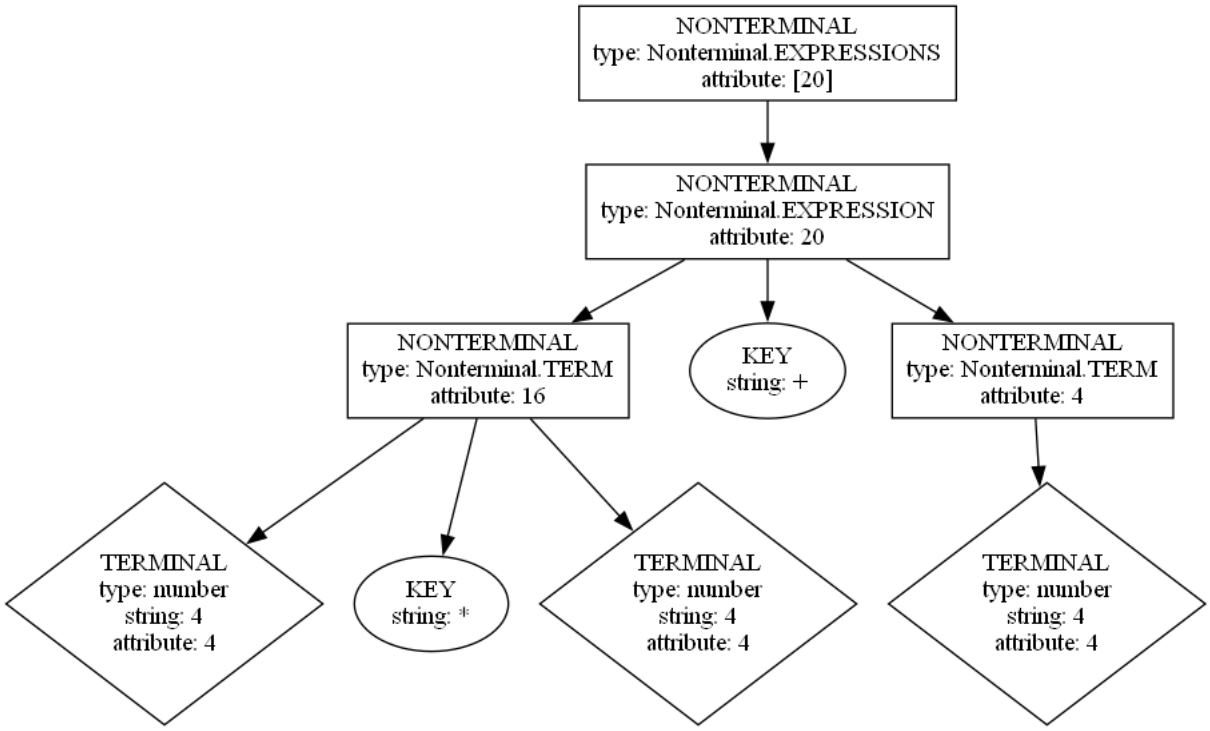

Рис. 5. Абстрактное синтаксическое дерево, в котором выставлены атрибуты

Атрибуты диаграмм можно использовать в других методах задания семантика как вспомогательные.

2.3.2. Разбор абстрактного синтаксического дерева

Инструмент строит абстрактное синтаксическое дерево. Его можно обойти на языке программирования общего назначения и при обходе 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.

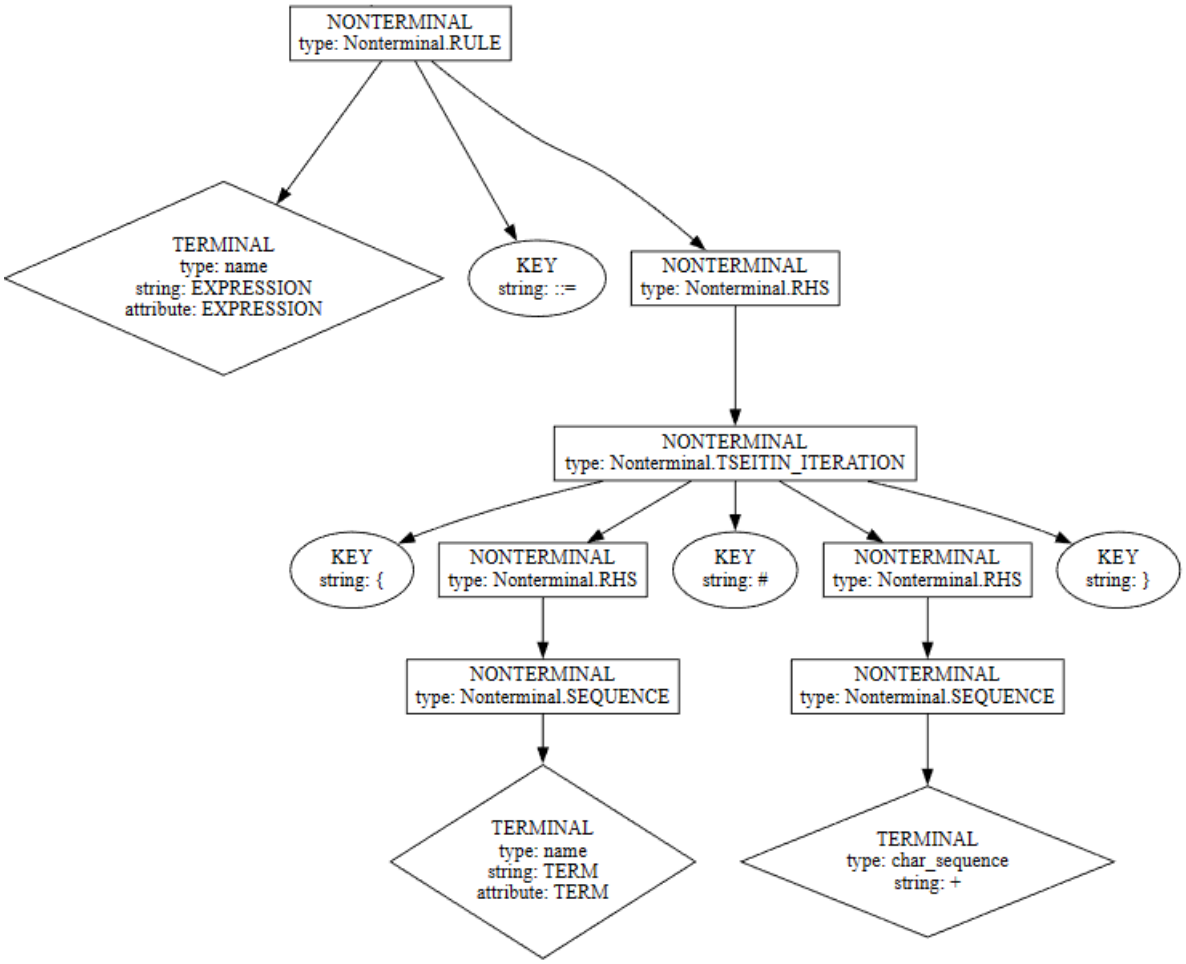

Рис. 6. Разбор нетерминала EXPRESSION в виде РБНФ

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-диаграмммах можно нагрузить инструкциями, которые должны быть исполнены при данном переходе.

Референс:

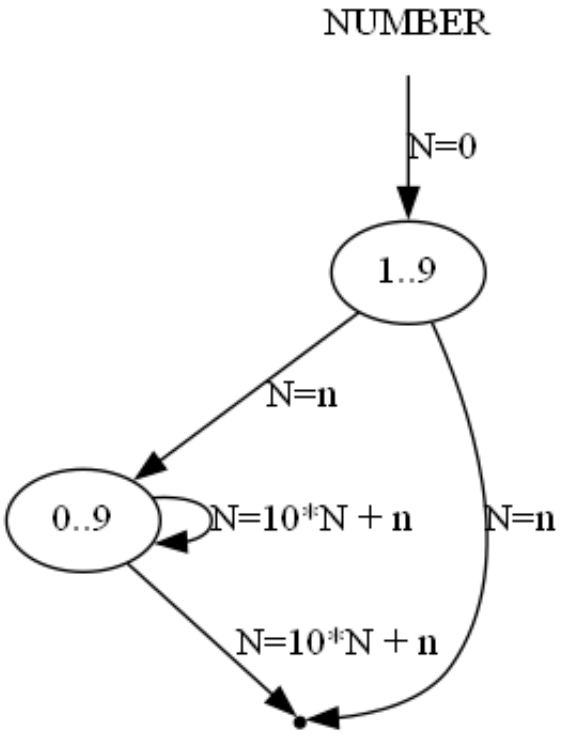

Рис. 7. Иллюстрация P-технологии

3. Архитектура

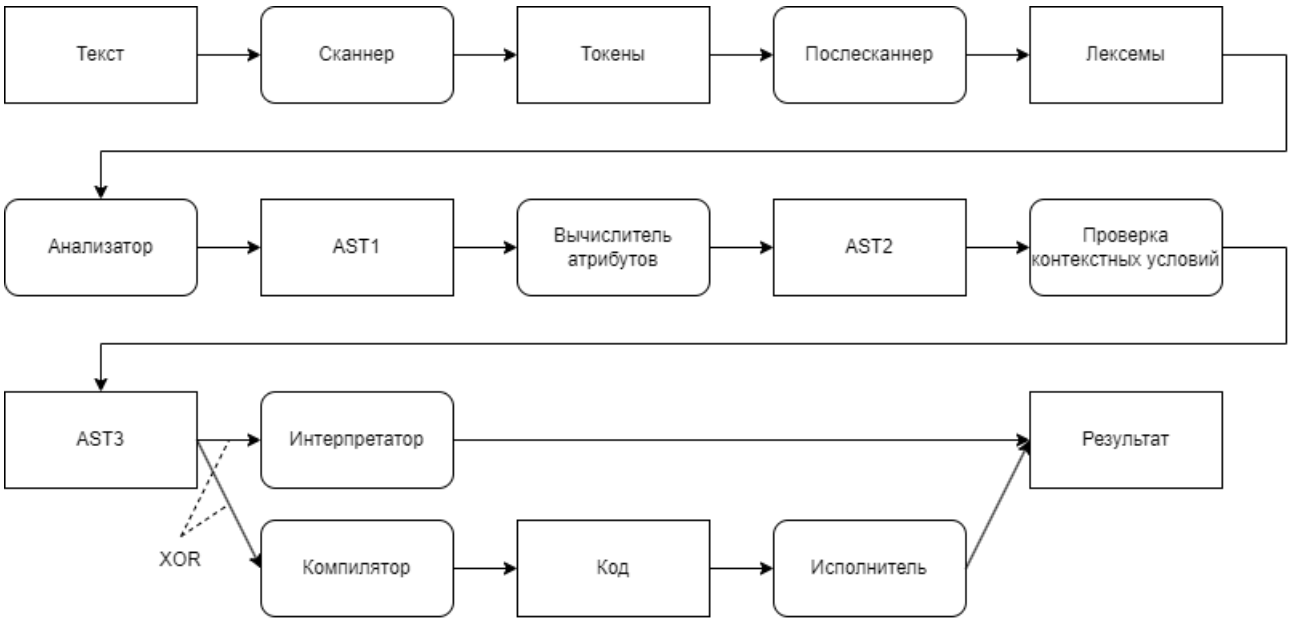

Рис. 8. Архитектура

3.1. Сканнер

Модуль преобразования текста в поток токенов. Для этого используется описание лексики.

Если найдена строка, не подходящая ни под одно регулярное выражение, то вызывается ошибка.

3.2. Послесканнер

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 "+").

Иллюстрация работы послесканнера для примера из пункта 2.3.1:

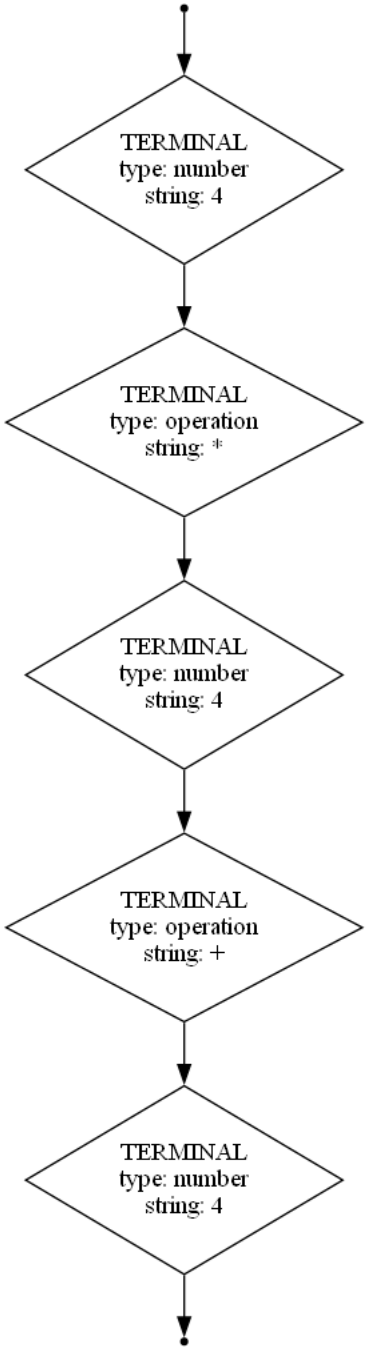

Рис. 9. Поток лексем после сканнера

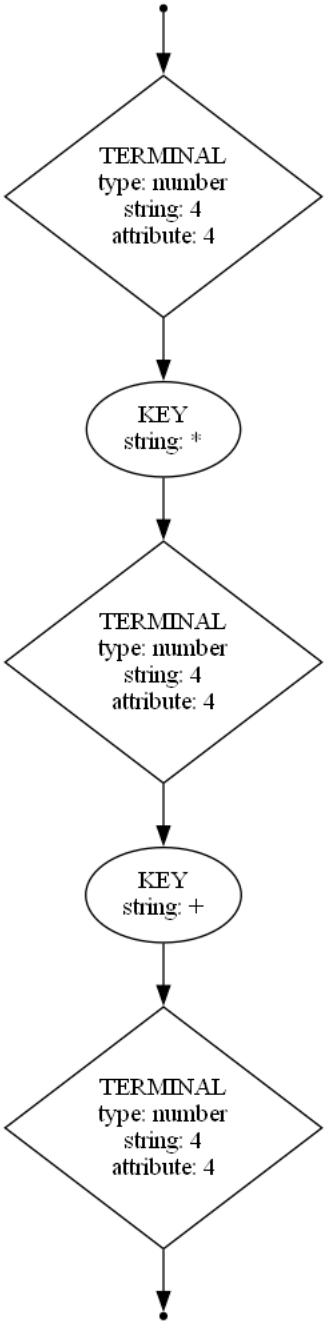

Рис. 10. Поток лексем после послесканнера

Различия в двух потоках в следующем:

1. у терминалов типа number появились 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

Строит 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 диаграммах 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 РБНФ синтаксических диаграмм Вирта в форме DOT-диаграмм.

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_CHILDn, где SEQUENCE_CHILDi $\in$ {name, char_sequence}, порождает подграф:

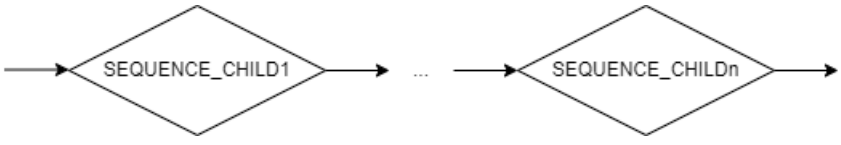

Рис. 11. Подграф для нетерминала SEQUENCE

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

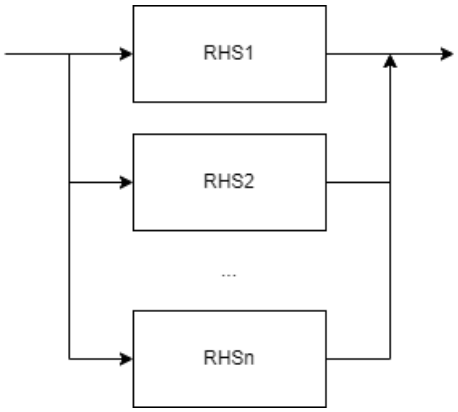

Рис. 12. Подграф для нетерминала BRACKETS

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

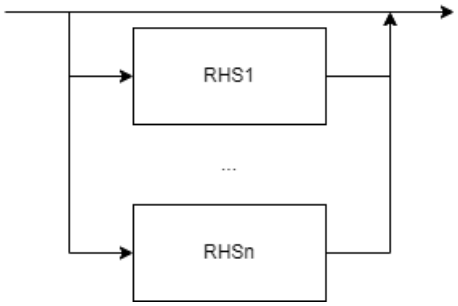

Рис. 13. Подграф для нетерминала OPTIONAL

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

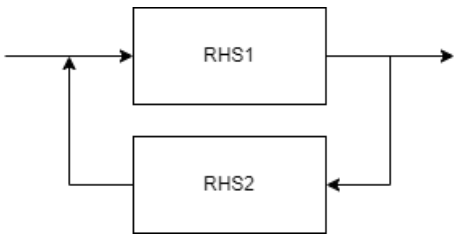

Рис. 14. Подграф для нетерминала TSEITIN_ITERATION

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

5. Приложения

5.1. Диаграммы UML

5.1.1. Функциональные требования

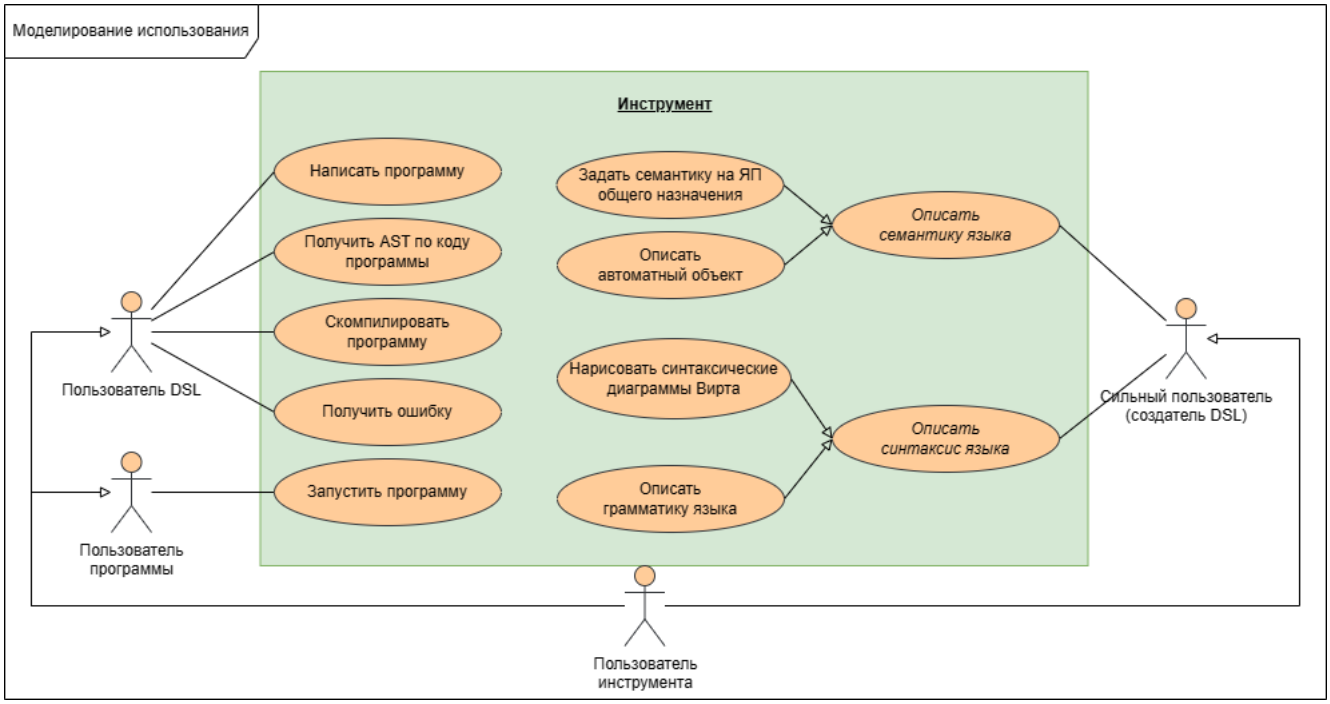

Рис. 15. Функциональные требования

5.1.2. Метамодель языка регулярной формы Бэкуса-Наура

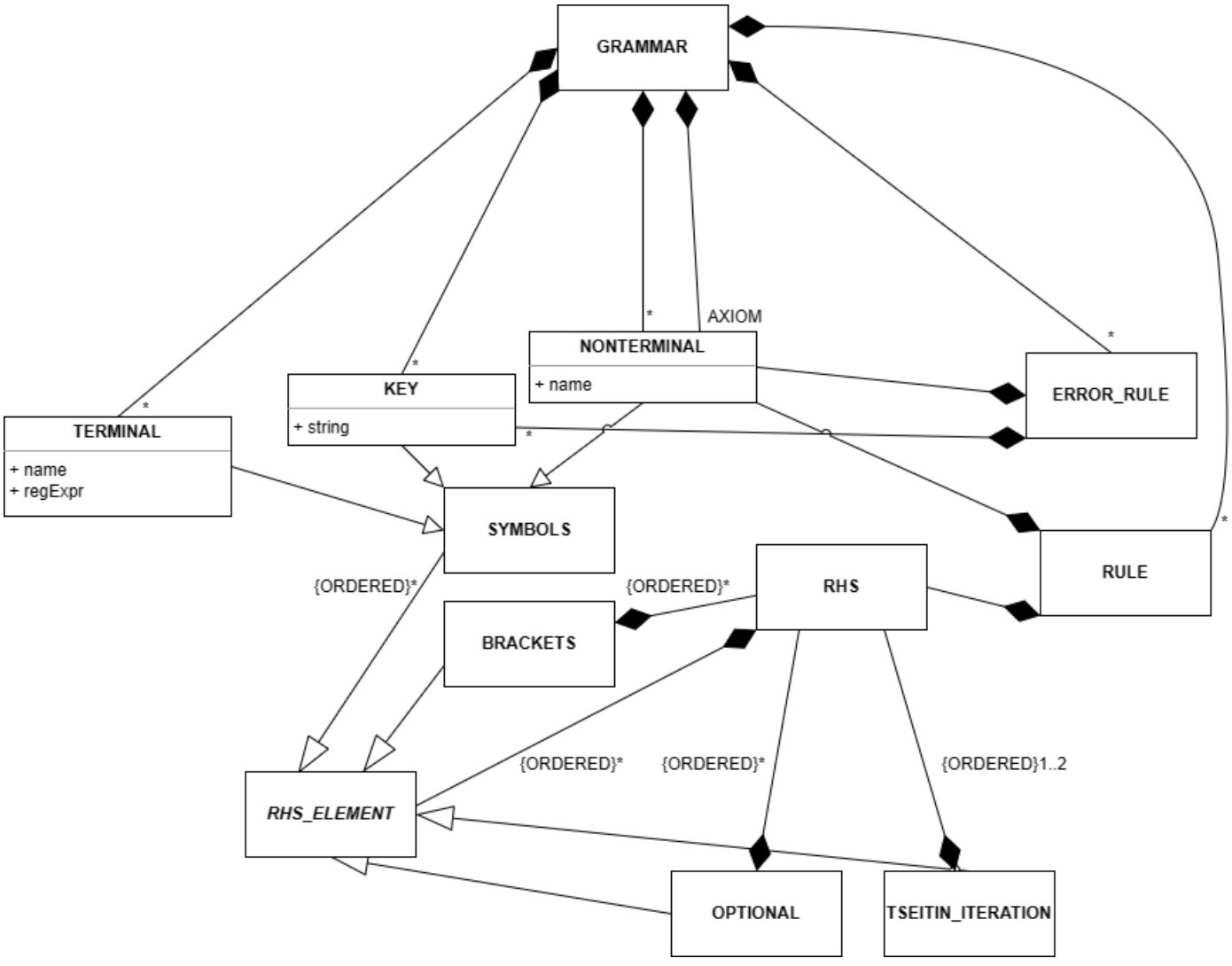

Рис. 16. Метамодель РБНФ

5.1.3. Диаграмма деятельности инструмента

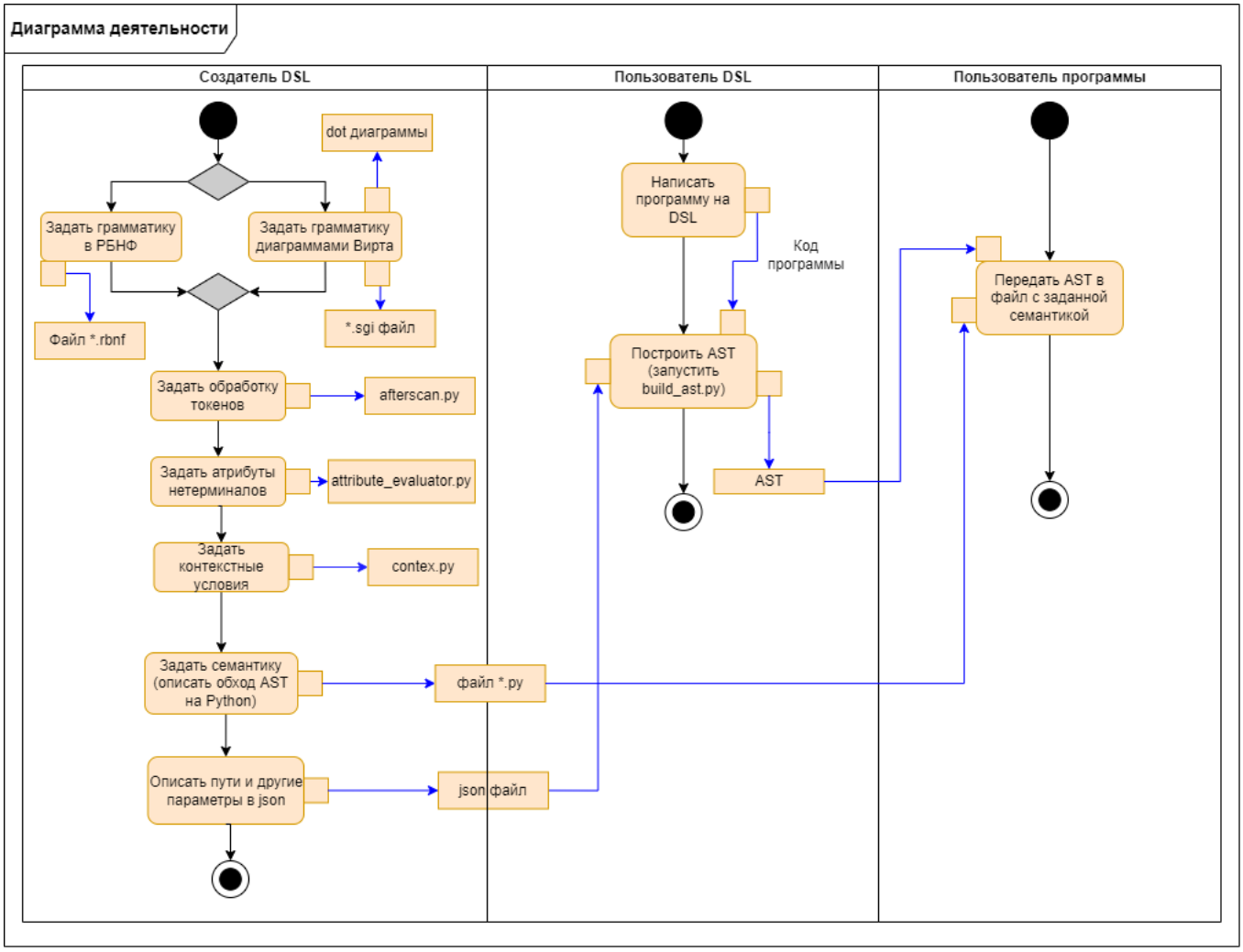

Рис. 17. Диаграмма деятельности инструмента

5.2. 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]}.
```

5.3. Описание регулярной формы Бэкуса-Наура синтаксическими диаграммами Вирта

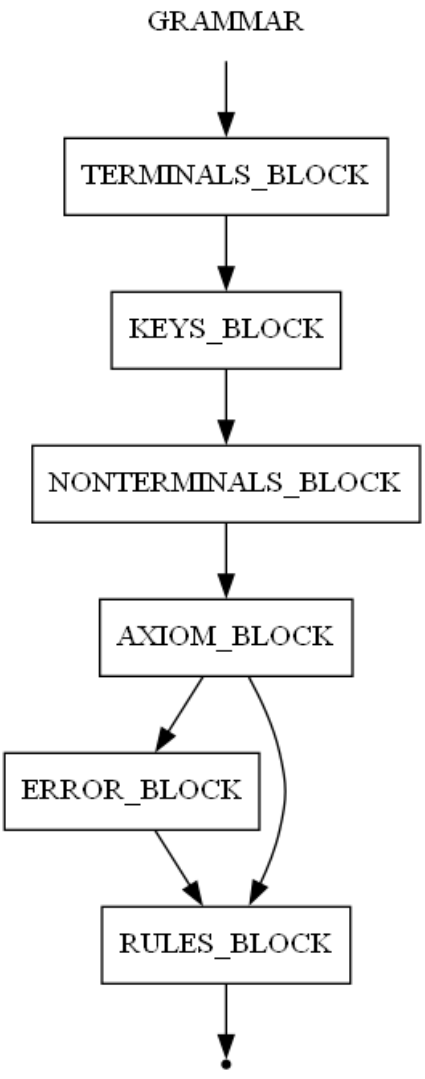

Рис. 18. Аксиома описания РБНФ

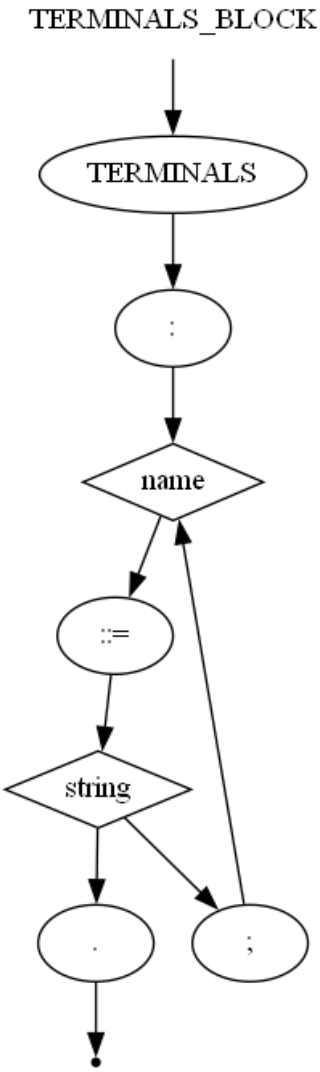

Рис. 19. Блок терминалов РБНФ

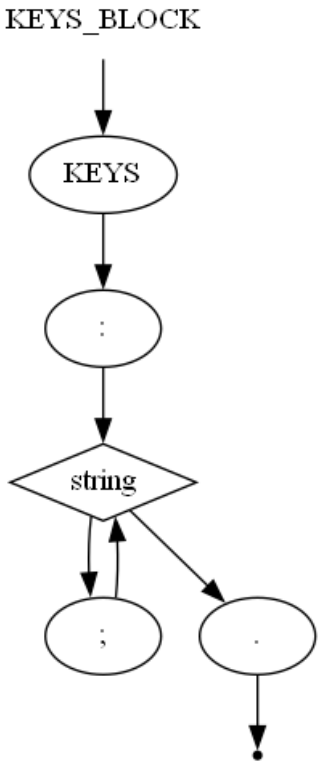

Рис. 20. Блок ключей РБНФ

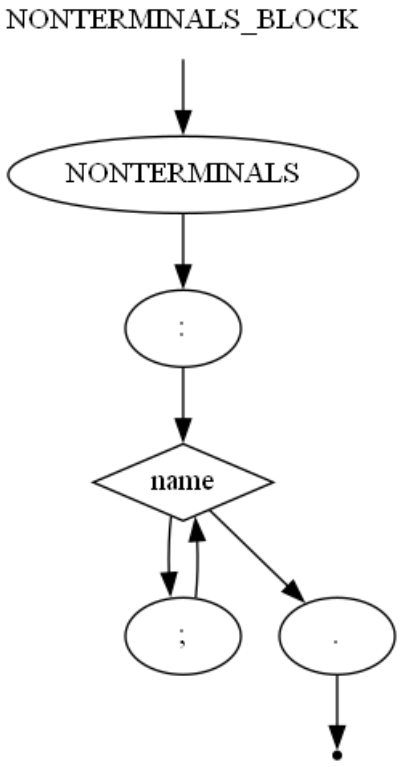

Рис. 21. Блок нетерминалов РБНФ

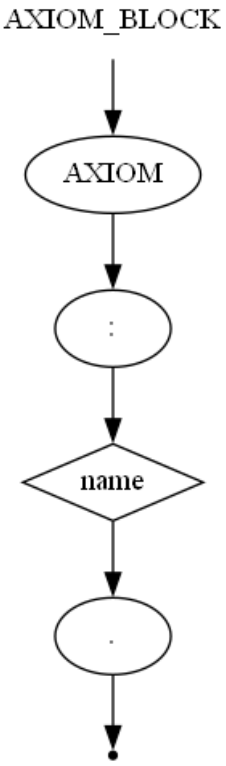

Рис. 22. Блок аксиом РБНФ

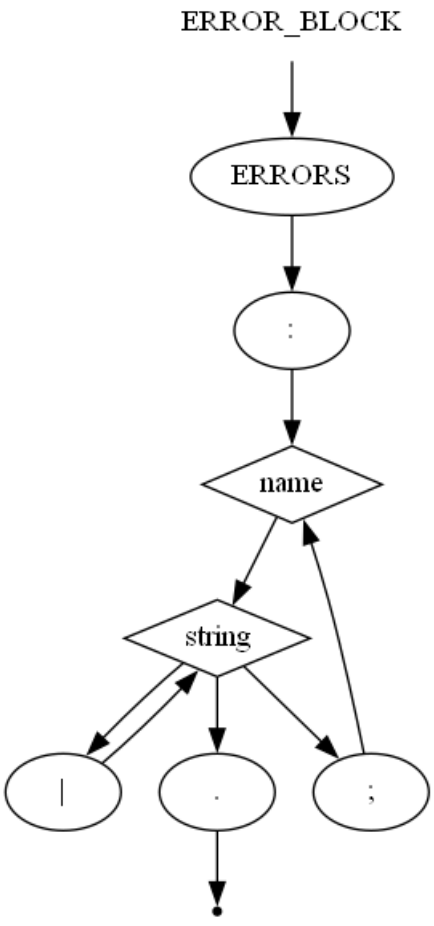

Рис. 23. Блок ошибок РБНФ

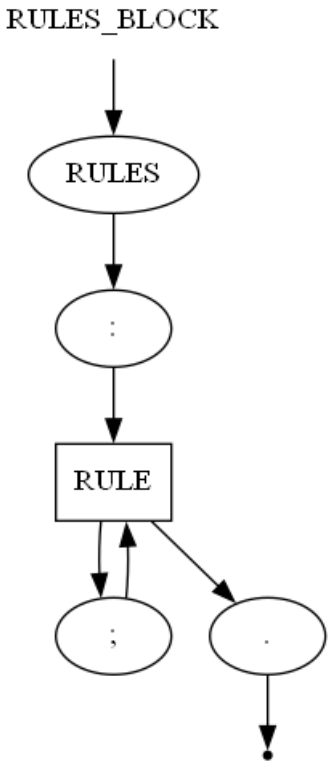

Рис. 24. Блок правил РБНФ

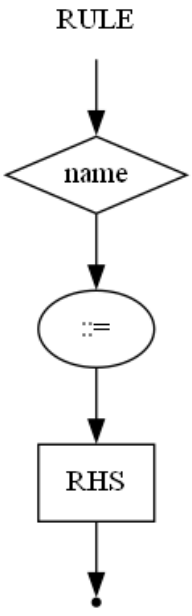

Рис. 25. Правило РБНФ

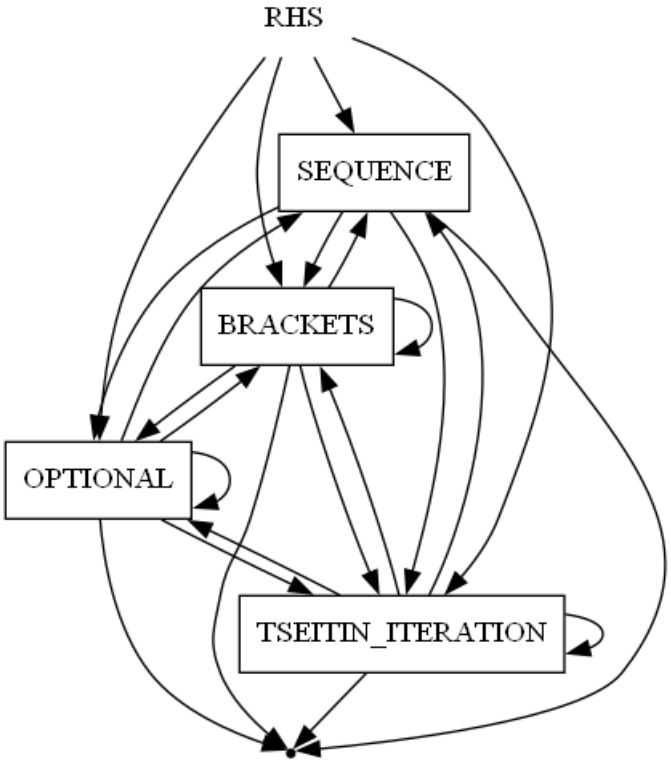

Рис. 26. Правая часть правил РБНФ

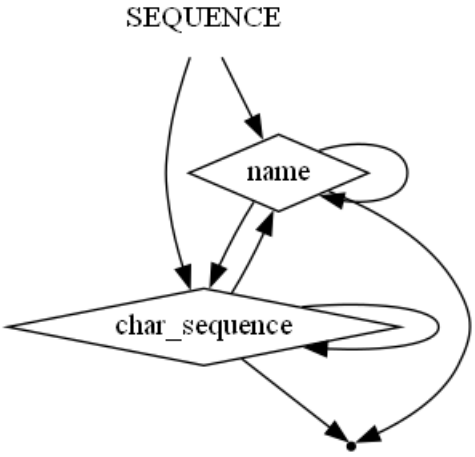

Рис. 27. Последовательность значений РБНФ

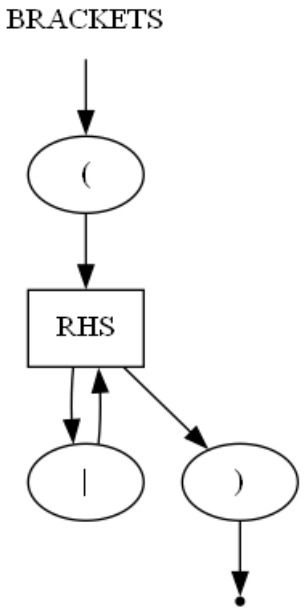

Рис. 28. Скобки в правилах РБНФ

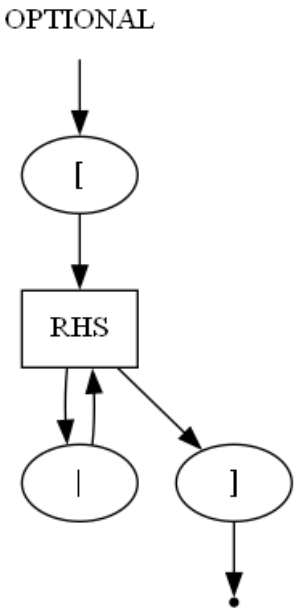

Рис. 29. Квадратные скобки в правилах РБНФ

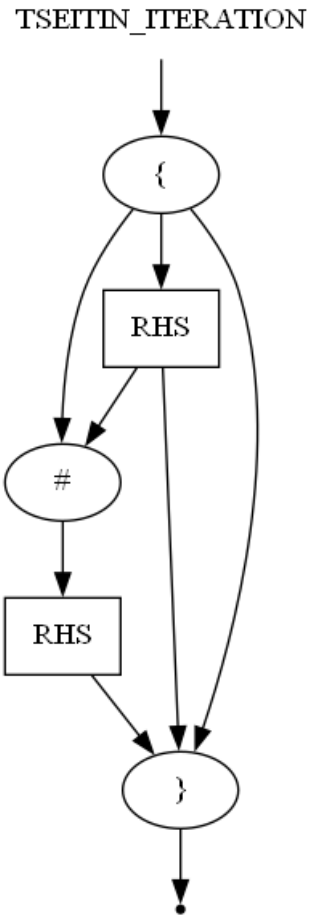

Рис. 30. Итерация Цейтина в правилах РБНФ

5.4. Ссылка на репозиторий

Исходные коды программы на GitHub, URL: `https://github.com/aVorotnikov/dsl_generator`.