Язык описания иллюстраций

Басалаев Даниил, Кромачёв Максим,
Сажин Даниил, Корпусова Софья, Асанов Дамир

15.04.2025

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Эта документация познакомит Вас с языком описания иллюстраций, созданным командой «Графы» студентов Физико-Механического института Санкт-Петербургского политехнического университета Петра Великого. DSL предназначен для лёгкого и быстрого создания схем, графов и облаков точек.

В документации Вы найдёте информацию об установке и запуске, грамматике и семантике языка, а также подробные примеры, которые помогут в создании Ваших собственных иллюстраций.

ГЛАВА 1
УСТАНОВКА И ЗАПУСК

1.1 Установка

Исходный код проекта расположен на платформе GitHub и доступен по адресу:
`https://github.com/11AgReS1SoR11/Graph.git`
Более подробную информацию по развёртыванию проекта можно найти в README.md.
Чтобы получить последнюю версию программмы, перейдите по ссылке
`https://github.com/11AgReS1SoR11/Graph/releases`
и скачайте архив, соответствующий Вашей операционной системе.

1.2 Запуск

Для трансляции кода в .json формат используется флаг `--translate`. За ним должен следовать флаг `--codeFilePath`, после которого пишется путь к файлу с кодом. Для того, чтобы включить режим ретрансляции, нужно указать флаг `--retranslate` и `--figuresJsonPath`, после чего написать путь к файлу с фигурами. Эти 4 флага являются обязательными. Есть также опциональные флаги, которые имеют значения по умолчанию:

- `--outputFilePath` - путь к файлу, в который нужно вывести результат (по умолчанию в текущей директориии figures.json для трансляции и code.graph для ретрансляции)
- `--logFilePath` - путь к файлу, в который будут выводиться логи (по умолчанию log.log текущей директории)

Примеры запуска:

```
./Graph --translate --codeFilePath code.txt --outputFilePath figures.json --
   logFilePath log.log
```

```
./Graph --retranslate --figresJsonFilePath figures.json --outputFilePath
   code.txt --logFilePath log.log
```

ГЛАВА 2

ГРАММАТИКА DSL

2.1 Создание иллюстрации

Грамматика языка имеет следующий вид:

```
program: '@startgraph ' statement* '@endgraph';

statement: object_decl | relation | note | graph | dot_cloud;

object_decl: SHAPE ID ('{' (property ';')* '}')?;

SHAPE: 'circle' | 'rectangle' | 'diamond';

relation: ID ARROW ID ('{' (property ';')* '}')?;

note: 'note' ID ('{' (property ';')* '}')?;

graph: 'graph' ID ('(' (property ';')* ')')? '{' (object_decl | relation)* '
    }'

dot_cloud: 'dot_cloud' ID ('(' (property ';')* ')')? '{' ('{' (property ';')
    * '}')* '}'

property: PROPERTY_KEY '=' (TEXT | NUMBER);

PROPERTY_KEY: 'color' | 'text' | 'border' | 'x' | 'y' | 'size_text' | '
    size_A' | 'size_B'; 'angle'; 'radius'; 'grid';

ARROW: '->' | '-->' | '<->' | '<-->' | '--' | '-';

ID: [a-zA-Z][a-zA-Z0-9_]*;
TEXT: [a-zA-Z0-9,.!? -]+;
NUMBER: [0-9]+;
WS: [ \t\r\n]+ -> skip;
```

Разберём её по шагам. Чтобы создать иллюстрацию, нужно заключить все объявления объектов и связей (другими словами, все `statement`) между двумя ключевыми словами:

```
@startgraph

# Здесь будут Ваши statements

@endgraph
```

Между этими ключевыми словами можно объявить:

1. Объект (`object_decl`)
   Это может быть

   - Круг (`circle`)
   - Прямоугольник (`rectangle`)
   - Ромб (`diamond`)

   При этом сначала пишется ключевое слово, обозначающее форму объекта, затем указывается его ID (другими словами, название объекта).

```
@startgraph

circle Start
rectangle Process
diamond End

@endgraph
```

   В фигурных скобках, следующих за ID, указываются свойства (`property`) объекта. Свойства DSL разделяются на общие и специфические. Следующие свойства могут пренадлежать любым объявлённым объектам.

   - `color` - цвет объекта. Может принимать значения из такого набора: `NONE`, `RED`, `GREEN`, `BLUE`, `BLACK`, `WHITE`, `YELLOW`, `PURPLE`.
   - `text` - текст, указанный на объекте.
   - `border` - ширина границы объекта.
   - `x`, `y` - координаты центра объекта.
   - `size_text` - размер текста на объекте.

   Кроме того, некоторые свойства могут пренадлежать только определённым фигурам. Среди них:

   - Круг (`circle`):
     - `radius` - радиус круга.
   - Прямоугольник (`rectangle`):
     - `size_A` - длина первой стороны.
     - `size_B` - длина второй стороны.
   - Ромб (`diamond`):
     - `size_A` - длина первой диагонали.
     - `size_B` - длина второй диагонали.
     - `angle` - угол между сторонами ромба.
   - Заметка (`note`):
     - `size_A` - длина первой стороны.
     - `size_B` - длина второй стороны.

Свойства отделяются друг от друга точкой с запятой. Если указано всего одно свойство, после него так же ставится точка с запятой.

```
@startgraph

circle Start {color = GREEN; radius = 10;}
rectangle Process {size_A = 5; border = 14;}
diamond End {angle = 40;}

@endgraph
```

2. Отношение между объектами (`relation`)
   Поддерживаются следующие виды отношений:

   - -> (тире, знак больше) - сплошная линия с наконечником-стрелкой на конце
   - --> (2 тире, знак больше) - пунктирная линия с наконечником-стрелкой на конце
   - <-> (знак меньше, тире, знак больше) - сплошная линия с наконечником-стрелкой на обоих концах
   - -- (2 тире) - пунктирная линия
   - - (одно тире) - сплошная линия

   Чтобы задать отношение между объектами, нужно указать имя объекта, от которого идёт стрелка/линия, затем указать саму стрелку/линию и затем написать имя объекта, к которому эта стрелка/линия ведёт. Строка, задающая отношение, завершается фигурными скобками, в которых можно указать свойства. Отношения могут быть проведены между любыми объектами, кроме графа и облака точек. Графы и облака точек не могут участвовать в отношениях.

```
@startgraph

circle Start {color = GREEN; radius = 10;}
rectangle Process {size_A = 5; border = 14;}
diamond End {angle = 40;}

Start -> Process {size_text = 11; text = "Действия";}
Process --> End {color = BLUE; text = "Последствия"}

@endgraph
```

3. Заметку (`note`)
   Заметка задаётся аналогично фигурам. ID заметки - это ID объекта, к которому относится заметка.

```
@startgraph

circle Start {color = GREEN; radius = 10;}
rectangle Process {size_A = 5; border = 14;}
diamond End {angle = 40;}

Start -> Process {size_text = 11; text = "Действия";}
Process --> End {color = BLUE; text = "Последствия";}

note Process {text = "Любое действие имеет последствия"; size_A = 10;
   size_B = 15;}
```

```
@endgraph
```

4. Граф (`graph`)
   Граф также объявляется аналогично предыдущим объектам, однако его отличает то, что он открывает свою область видимости. То есть все объекты, объявлённые в фигурных скобках графа, существуют только в его области видимости. Из области видимости графа нельзя обратиться ко внешним объектам и наоборот, из внешней области видимости нельзя обратиться к объектам внутри области видимости графа. Граф не имеет своих специальных свойств, но может обладать всеми общими свойствами, которые указываются в круглых скобках и распространяются на объекты, объявленные внутри графа. Если же для каких-то из этих объектов будут объявлены свои свойства, то для этих объектов будут применены указанные свойства, а не свойства графа.

```
@startgraph

graph Graph (color = YELLOW) {
    circle A {text = "A";}
    circle B {text = "B"; color = PURPLE;}
    circle C {text = "C";}
    Rectangle D {text = "D";}

    A -> B {}
    A -> C {}
    A -> D {}
    B -> D {}
}

@endgraph
```

   В таком графе все вершины будут желтого цвета, а вершина B будет фиолетовой.

5. Облако точек (`dot_cloud`)
   Облако точек объявляет свою область видимости подобно графу. Так как все объекты в нём точки, их не нужно отдельно объявлять. В фигурных скобках облака точек указываются только координаты точек. Они являются обязательными свойствами. Облако точек имеет специфическое свойство: можно включить для него сетку (`grid = true`). Свойства облака точек указываются в круглых скобках.

```
@startgraph

dot_cloud(grid = true){
    {x = 15; y = 20; color = red;}
    {x = 40; y = 14;}
}

@endgraph
```

2.2 Правила лексики, синтаксиса и семантики

В этом разделе описаны ограничения, накладываемые на свойства и идентификаторы объектов, а так же оговорены проверки, проводимые семантическим анализатором.

1. Все ключевые слова
   `circle; rectangle; diamond; color; text; border; x; y; graph; dot_cloud; size_text; size_A; size_B; note; @startgraph; @endgraph; angle; radius; grid`
   указываются строго в нижнем регистре.

2. Идентификаторы объектов могут быть написаны в любом регистре. При этом `circle Process` и `circle process` будут считаться разными объектами.

3. Свойства должны иметь следующие типы:

   - `color` - `string`. Допустимые значения: `NONE, RED, GREEN, BLUE, BLACK, WHITE, YELLOW, PURPLE`. Могут быть указаны стрго в верхнем или строго в нижнем регистре.
   - `text` - `string`. Текст может быть написан в любом регистре.
   - `border` - целое число.
   - `x` - целое число.
   - `y` - целое число.
   - `size_text` - целое число, большее нуля.
   - `radius` - число, большее нуля.
   - `size_A` - число, большее нуля.
   - `size_B` - число, большее нуля.
   - `angle` - число, большее нуля.
   - `grid` - `boolean`.

   Все свойства должны иметь соответствующие типы и удовлетворять соответствующим ограничениям.

4. Для каждой точки в облаке точек обязательно должны быть указаны обе координаты.

5. Идентификатор объекта (`ID`) должен быть уникальным в пределах своей области видимости. Области видимости создаются для:

   - Основной программы
   - Каждого графа (`graph`)
   - Каждого облака точек (`dot_cloud`)

6. Каждое свойство объекта должно быть допустимым для его типа (см. общие свойства и специфические свойства).

7. Оба объекта, участвующие в отношении, должны быть объявлены. Связь не может быть создана с графом (`graph`) или облаком точек (`dot_cloud`).

8. Внутри графа (`graph`) не должно быть двух связей между одними и теми же объектами.

9. Граф не может содержать другие графы или облака точек.

10. При обнаружении ошибки пользователю выдаётся сообщение с информацией о ней.

Кроме того, фигуры на холсте и внутри графа по умолчанию располагаются таким образом, чтобы они не накладывались друг на друга и связи между объектами имели как можно меньше число пересечений.