САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра Системного Программирования

Озерных Игорь Станиславович

Декларативное форматирование в режиме онлайн

Курсовая работа

Научный руководитель:
асп. Подкопаев А. В.

Санкт-Петербург
2015

Оглавление

Введение

Этап поддержки и сопровождения в жизненном цикле программного обеспечения может занимать до 90% времени существования программного продукта. На этом этапе особенно важно понимание программного текста поддерживаемой системы, что зачастую бывает сложной задачей. Необходимым условием для лучшего понимания программного текста является его аккуратное оформление, отражающее структуру-программы. Существуют различные стандарты оформления кода (стили кодирования) – наборы правил и соглашений, используемые при написании исходного кода на некотором языке программирования. Рассмотрим их на примере стандарта кодирования, принятого в компании Google[1], и стандарта, используемого в проекте GNU[2] (см. рис 1) для языка C++.

```
int ClassName::Foo(int k) {
  if (k <= 0) {
    Bar();
  } else {
    switch (k)
    {
      case 1: {
        k += 2;
        break;
      }
      case 2: {
        k *= 2
        break;
      }
    }
  }
  return k;
}
```

а) Стиль кодирования Google

```
int ClassName::Foo(int k)
{
    if (k <= 0)
      {
        Bar();
      }
    else
      {
        switch (k)
          {
            case 1:
                k += 2;
                break;
            case 2:
                k *= 2
                break;
          }
      }
    return k;
}
```

б) Стиль кодирования GNU

Рис. 1: Различные стили кодирования для языка C++

Когда программист работает над проектом, он придерживается некоторого стиля кодирования. Если файлы проекта с одним стилем кодирования присоединяется к уже существующему проекту с другим стилем кодирования, то их форматирование нужно привести к тому же виду. Для решения этой задачи можно воспользоваться уже существующими средствами, например, форматтерами (программами, которые форматируют исходный текст) в IDE таких, как Eclipse, IntelliJ IDEA, Visual Studio

[1]https://google-styleguide.googlecode.com/svn/trunk/cppguide.html
[2]http://www.gnu.org/prep/standards/standards.html

и др. Однако в этом случае необходимо вручную задавать настройки форматирования. Кроме того, количество принципиально разных стилей форматирования, которые можно задать с помощью данных настроек, невелико.

Еще одним примером может послужить принтер-плагин для IntelliJ IDEA[3], который позволяет форматировать исходный код проекта по образцу. В качестве образца используется код из некоторого репозитория. Из него выделяются шаблоны форматирования для структур языка, которые применяются к структурам целевого кода. Под *шаблоном* понимаются данные, сопоставление которых с элементом синтаксического дерева дает текстовое представление этого элемента (и его потомков). Например, на рис. 2 представлено дерево разбора для оператора ветвления.

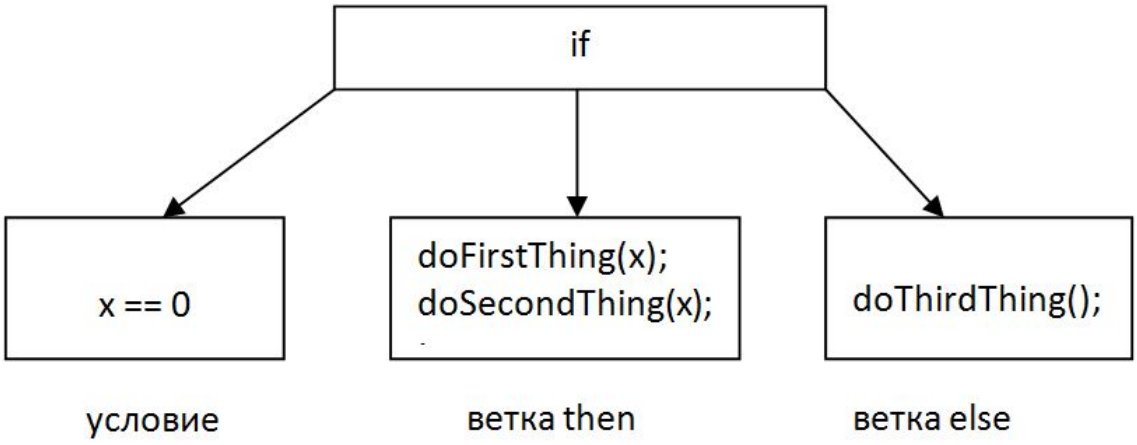

Рис. 2: Представление оператора ветвления в виде дерева разбора

На рис. 3а изображен шаблон форматирования, который может быть применен к нему (несколько подвыражений, соответствующих выполнению условия, и одно подвыражение, соответствующее невыполненению условия). На рис. 3б представлен результат применения этого шаблона к дереву разбора для оператора ветвления.

```
if (@condition) {
    @exprM
} else {
    @exprS
}
```

а) Шаблон для оператора ветвления

```
if (x == 0) {
    doFirstThing(x);
    doSecondThing(x);
} else {
    dothirdThing();
}
```

б) Текст, полученный при применении шаблона к дереву разбора

Рис. 3: Оператор ветвления и шаблон для него

Однако не всегда этот репозиторий существует. Кроме того, необходимо наличие заданного форматирования для всех структур языка.

Шаблоны можно извлекать из уже существующего кода и применять их к другим элементам того же типа, но отформатированных иным способом. Например, пользователь меняет форматирование некоторой структуры языка (в частности, оператор ветвления), программа извлекает из полученного участка кода новый шаблон и применяет его к структурам того же типа, содержащим старое форматирование. Назовем такой способ задания шаблонов – форматирование в режиме онлайн.

Целью данной работы является расширение функциональности описанного принтер-плагина путем добавления возможности задания шаблонов в режиме онлайн.

1. Обзор существующих решений

Существуют различные способы задания шаблонов для форматирования. Далее рассматриваются некоторые из них, а так же приводится описание структуры проекта, в рамках которого будет реализовываться дополнительный функционал.

1.1. Средства форматирования в IDE

Интегрированные среды разработки предоставляют средства для форматирования программного кода. Рассмотрим их на примере IDE IntelliJ IDEA[3]. Необходимый стиль кодирования (СК) задается с помощью различных настроек форматтера. Среди них:

- настройка размера и типа отступа (табуляция или пробелы);
- принудительная расстановка фигурных скобок;
- расположение простых блоков на одной или нескольких строках.

На рис. 4 приведен диалог задания параметров форматирования в IntelliJ IDEA.

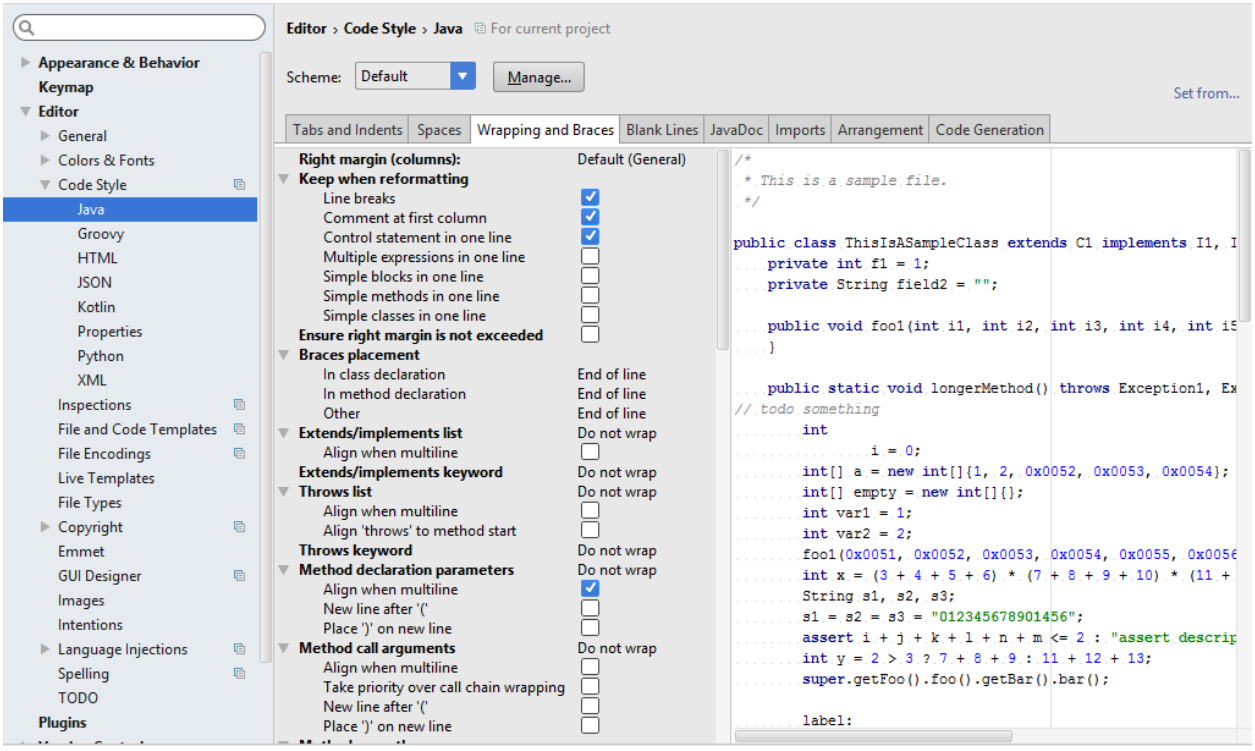

Рис. 4: Окно настройки форматтера IntelliJ IDEA

Однако у встроенных форматтеров есть недостатки. Во-первых, с помощью упомянутых настроек нельзя выразить нестандартный СК (см. рис. 5). Во-вторых, стиль

[3]http://jetbrains.com/idea

кодирования, указанный в настройках, будет применен ко всем блокам некоторого типа, независимо от их форматирования. Иногда имеет смысл поддерживать различные варианты форматирования для блоков одного типа в рамках проекта.

```
for (i = 0; i < 10; i++) {
    if (i % 2 == 0) {
        doSomething(i); }
    else {
        doSomethingElse(i); } }
```

Рис. 5: Нестандартный СК для языка C

1.2. Stratego/XT

Stratego/XT[4] – язык и набор инструментов для преобразования программ. Язык предоставляет правила переписывания, описывающие базовые шаги преобразования программы[2]. Одно из этих правил изображено на рис. 6.

```
desugar : While(e, stm) -> If(e, DoWhile(stm, e))
```

Рис. 6: Правило для развертки блока While

Преобразование программы производится путем одного или нескольких применений этих правил к абстрактному синтаксическому дереву (АСТ) (рис. 7), а затем "красивой" печати этого дерева согласно некоторым правилам форматирования. Одно из правил преобразования АСТ в текст изображено на рис. 8.

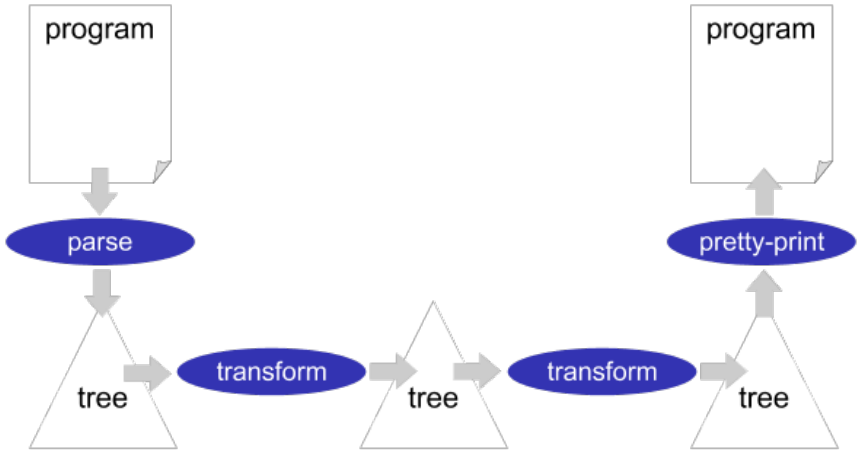

Рис. 7: Преобразование программ в Stratego/XT[5]

[4]http://strategoxt.org
[5]http://releases.strategoxt.org/

```
V [ H [ "if" _1 "then" ] _2 "fi" ]
```

Рис. 8: Правило для преобразование ACT в текст в Stratego/XT

Недостатками этого подхода являются, во-первых, необходимость владения синтаксисом языка Stratego, во-вторых, нужна отдельная среда разработки (к примеру, IntelliJ IDEA и Visual Studio не имеют плагина, позволяющий работать со Stratego/XT), в-третьих, данный подход имеет единые правила форматирования стукртур одного типа, а значит, не может поддерживать различные стили форматирования для них.

1.3. Принтер-плагин для IntelliJ IDEA

Другой метод задания шаблонов представляет собой принтер-плагин для IntelliJ IDEA, позволяющий форматировать исходный текст по образцу. Образцом является некоторый репозиторий, содержащий код с желаемым форматированием. Пользователь выбирает файлы проекта-образца и целевой файл для форматирования. Каждый файл представляется в виде дерева разбора, состоящего из элементов классов типа PsiElement[1], например, PsiIfStatement (оператор ветвления), PsiExpressionStatement (выражение). Из этих элементов извлекаются шаблоны форматирования с помощью метода getAndSaveTemplate() класса PsiElementComponent. Эти шаблоны будут использоваться принтером (Printer) для изменения форматирования целевого файла.

Затем принтер получает на вход целевой файл и к каждому элементу дерева применяет функцию, которая строит FormatSet, набор, состоящий из различных вариантов форматирования, построенных исходя из извлеченных шаблонов. Из этого набора выбирается лишь один вариант форматирования, и происходит перепечатывание этого элемента дерева разбора с учетом нового форматирования. Взаимодействие классов изображено на рис. 9.

Рассмотренные способы задания шаблонов не могут поддерживать различные стили форматирования для элементов одного типа, а так же не имеют возможности задания шаблонов в интерактивном режиме.

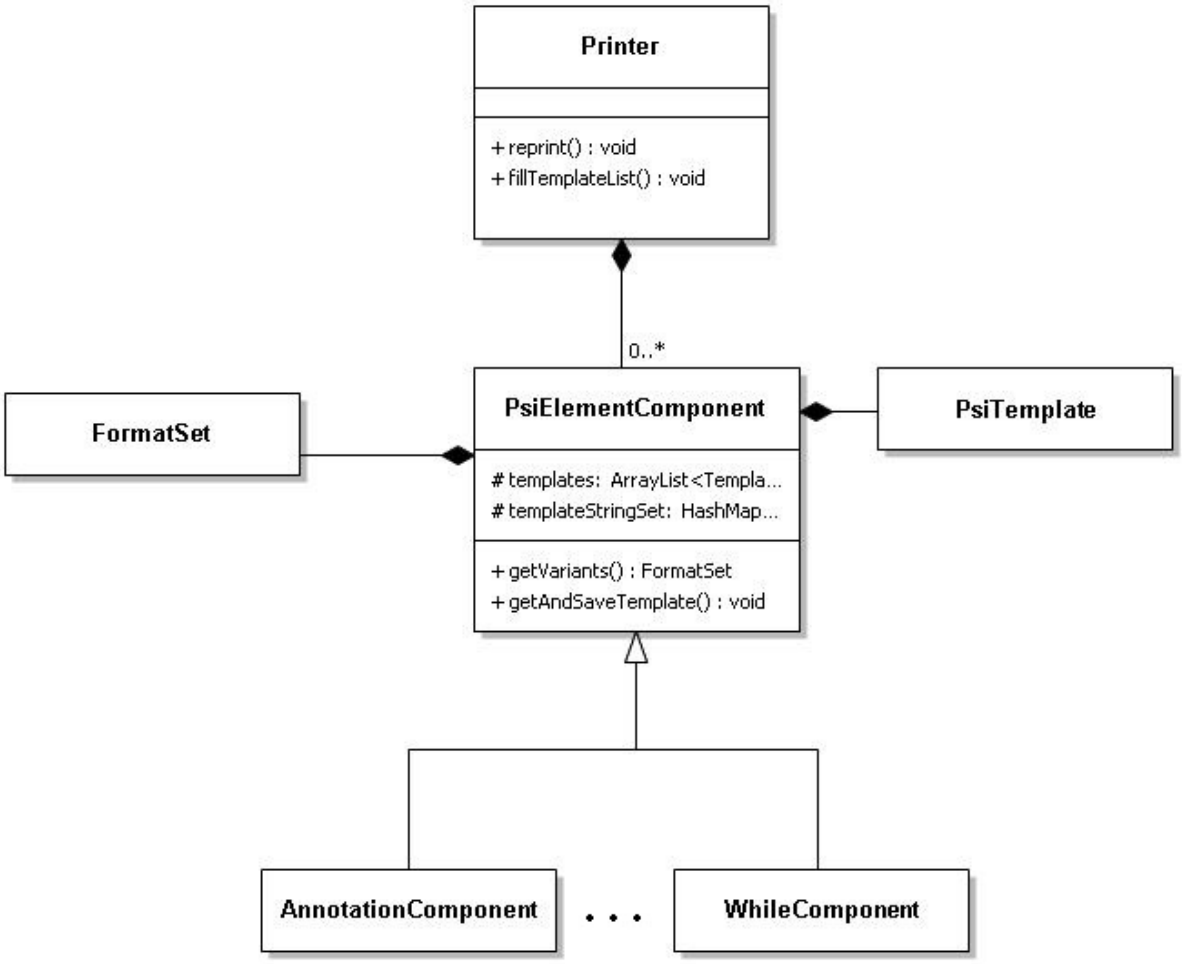

Рис. 9: Основные классы принтер-плагина

2. Реализация

В процессе работы над проектом было выделено две основных задачи: реализация взаимодействия с пользователем и расширение функциональности для переформатирования программного текста с фильтрацией шаблонов. Предполагаемый процесс взаимодействия с пользователем: пользователь выделяет участок программного кода, в котором будут производиться изменение, и вызывает действие (Action), которое извлекает шаблон из выделенного участка кода. Затем пользователь меняет форматирование этого же участка, выделяет его снова и вызывает действие, которое извлекает новый шаблон форматирования и сохраняет его.

После этого происходит перепечатывание элементов (PsiElement), имеющих старое форматирование, в файле, в котором пользователь менял программный текст.

2.1. Переформатирование

2.1.1. Извлечение элемента для форматирования

В первую очередь необходимо получить элемент, форматирование которого пользователь хочет изменить. Весь этот процесс изображен на рис. 10.

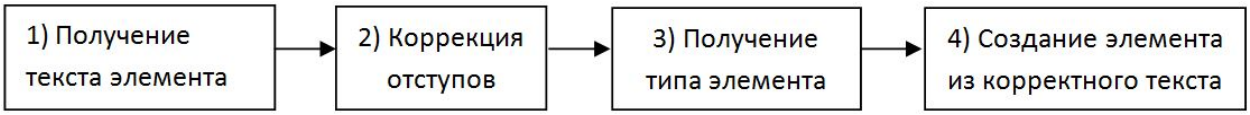

Рис. 10: Диаграмма этапов извлечения элемента из текста

На первом этапе с помощью функций, предоставляемых IntelliJ IDEA API, можно получить текст элемента. Однако этот текст не всегда корректный, так как может содержать в себе лишние отступы. Например, на рис. 11 изображены методы doSomething() и doAnotherThing(). Они имеют отступ относительно оператора ветвления, равный четырем пробелам. Кроме того, они имеют отступ относительно метода foo(), равный восьми пробелам. Чтобы шаблон был корректным, на втором этапе убираются лишние пробелы (в данном случае четыре). Далее получаем из текста сам элемент. Это необходимо, чтобы узнать его тип. Затем (на четвертом этапе) из корректного текста и полученного элемента создается новый элемент того же типа, но с корректными отступами.

Все это необходимо проделать, так как IntelliJ IDEA API не предоставляет функционала по изменению текста элемента явным образом.

Аналогичные действия необходимо повторить для получения элемента из измененного пользователем текста. Из этого элемента извлекается шаблон и сохраняется принтером для дальнейшего использования.

```
public void foo() {
    int someValue = getSomeValue();
    if (someValue != 0) {
        doSomething();
        doAnotherThing();
    }
}
```

Рис. 11: Общий отступ элементов

2.1.2. Сравнение с шаблоном

Как уже было отмечено в обзоре, при перепечатывании принтер обходит все элементы (PsiElement) дерева разбора и применяет к ним новые шаблоны. В рамках реализуемого подхода происходит перепечатывание лишь тех элементов, которые имели старое форматирование. Это необходимо, если хочется поддерживать два и более различных варианта форматирования элементов некоторого типа. На рис. 12 изображены три блока оператора ветвления. Без фильтрации шаблонов все три блока будут изменены и получат одинаковое форматирование. Однако первые два блока и так легко читаемы и понятны. Третий блок, будучи однострочным, уже не будет восприниматься так же хорошо. Кроме того, существуют ограничения на ширину вывода, то есть в одной строке не можеть быть символов больше некоторой наперед заданной величины.

```
if (someValue == 0) { return; }
if (someObject == null) { return; }

if (veryLongUsefulFunctionName())
{
    doSomethingSpecial();
}
```

а) До переформатирования

```
if (someValue == 0) {
    return;
}
if (someObject == null) {
    return;
}

if (veryLongUsefulFunctionName()) {
    doSomethingSpecial();
}
```

б) После переформатирования

Рис. 12: Переформатирование элементов одного типа без фильтрации шаблонов

Чтобы сделать настройку форматирования более гибкой и избавиться от подобного рода проблем, необходимо производить сравнение шаблонов текущего элемента дерева разбора и элемента со старым форматированием. Переформатировать элемент – только в случае совпадения шаблонов. Шаблоны совпадают, если совпадет их текстовое представление: шаблон записывается в виде строки, которая содержит необходимые элементы: ключевые слова, пробелы, переносы строк, скобки, а так же

метки для поддеревьев, в которых указывается дополнительная информация, например, количество выражений в поддереве (одно или несколько). В случае оператора ветвления (рис. 13а) его поддеревьями являются: условие (condition), ветка, соответствующая выполнению условия (then block) и содержащая несколько подвыражений, ветка, соответствующая невыполнению условия (else block) и содержащая одно подвыражение. Соответствующий ему шаблон изображен на рисунке 13б.

```
if (someValue == 0) {
    doFirstThing();
    doSecondThing();
}
else {
    doThirdThing();
}
```

а) Оператор ветвления

```
if (condition#0 OneLineB#) {
    then block#-1 MultiB#
}
else {
    else block#-1 OneLineB#
}
```

б) Шаблон для оператора ветвления

Рис. 13: Оператор ветвления и соответствующий ему шаблон

2.2. Оценка производительности

Описанный выше способ давал очень низкую производительность. В таблице 1 приведено время, затрачиваемое принтером на переформатирование файлов различной длины. Измерения производились на примере двух блоков: оператора ветвления и метода.

	Формат. операторов ветвления		Форматирование методов	
Размер файла (строк)	Кол-во	Время (сек.)	Кол-во	Время (сек.)
>7000	520	35	660	58
1000..2000	145	2.7	90	4.4
500..1000	54	0.73	72	1.34
0..500	35	0.2	36	0.67

Таблица 1: Время переформатирования файлов I

Причина столь низкой производительности кроется в том, что для переформатирования элементов используется тот же способ, что и при форматировании по образцу: принтер обходит все дерево разбора и сравнивает шаблоны. Однако нет необходимости сравнивать шаблон, полученный из элемента со старым форматированием, с шаблоном, полученным из каждого элемента дерева разбора. Количество вариантов для переформатирования можно сузить лишь для элементов того же типа.

Для уменьшения затрат времени при обходе дерева элементов создается список кандидатов на форматирование (элементы того же типа, что и элемент со старым

форматированием). После этого из элементов созданного списка извлекаются шаблоны. Если полученный шаблон совпадает с шаблоном элемента со старым форматированием, то происходит перепечатывание элемента и замена его в дереве разбора. Время, затрачиваемое в этом случае на переформатирование, приведено в таблице 2.

	Формат. операторов ветвления		Форматирование методов	
Размер файла (строк)	Кол-во	Время (сек.)	Кол-во	Время (сек.)
>7000	520	6.6	660	5.8
1000..2000	145	0.72	90	1
500..1000	54	0.225	72	0.67
0..500	35	0.145	36	0.325

Таблица 2: Время переформатирования файлов II

Заключение

В рамках данной работы была реализована возможность задания шаблонов в режиме онлайн для форматирования текста, а так же были достигнуты приемлемые результаты производительности.

Список литературы

[1] JetBrains. Документация // IntelliJ IDEA. — 2015. — URL: https://confluence.jetbrains.com/display/IDEADEV/PluginDevelopment (дата обращения: 26.05.2015).

[2] Stratego/XT. Документация // Stratego/XT. — 2015. — URL: http://strategoxt.org/Stratego/StrategoDocumentation (дата обращения: 26.05.2015).

[3] Подкопаев А.В. Полиномиальной сложности оптимальные принтер-комбинаторы с выбором // Дипломная работа, кафедра системного программирования, математико-механический факультет СПбГУ. — 2014. — URL: http://se.math.spbu.ru/SE/diploma/2014/s/Podkopaev_Diploma.pdf (дата обращения: 26.05.2015).