Санкт–Петербургский государственный университет
Факультет математики и компьютерных наук

Никита Алексеевич Босов

Выпускная квалификационная работа

Тема работы: Расширяемый генератор синтаксически корректных программ для обучения программированию

Уровень образования: бакалавриат
Направление 01.03.02 «Прикладная математика и информатика»
Основная образовательная программа СВ.5005.2018 «Прикладная математика, фундаментальная информатика и программирование»
Профиль «Современное программирование»

Научный руководитель:
д.ф.-м.н, профессор СПбГУ А.С. Куликов

Рецензент:
ассистент кафедры МОЭВМ ЛЭТИ
Н. В. Шевская

Санкт-Петербург
2022 г.

Содержание

Введение

В настоящее время знание языка программирования является необходимым для специалиста в отрасли информационных технологий, а обучение им - крайне востребованным. На сегодняшний день программы по обучению языкам программирования есть не только в университетах, но и на различных образовательных платформах в интернете. В связи с ростом числа учащихся подобных курсов и ослабления контакта между студентом и преподавателем острее встает проблема создания учебных материалов, в частности практических заданий. Требуется создавать их в большем объеме и в то же время делать их разнообразными во избежание списывания. Специфически для курсов по изучению языков программирования возникает необходимость создания множества примеров программ на определенную тему или по конкретному шаблону. Создание подобных примеров вручную в нескольких вариантах (в идеале по отдельности для каждого ученика) затруднительно. Таким образом, создание удобного программного инструмента, позволяющего автоматически генерировать примеры кода на различных языках программирования для учебных задач представляет собой актуальную проблему.

Постановка задачи

Целью данной выпускной работы является создать расширяемый генератор случайных программ для учебных задач, используемых в курсах по обучению языкам программирования.

Основные задачи которые необходимо сделать:

a. Изучить существующие системы генерации случайных программ на предмет возможности их настройки и применимости результатов их работы в учебных целях.

b. Создать систему генерации программ с возможностью настройки параметров для одного языка программирования (Python)

c. Адаптировать систему к возможности поддержки других языков программирования.

Объектом моего исследования являются инструменты генерации программного кода, а **предметом** исследования — применимость инструментов генерации кода для создания учебных задач.

Данная работа является развитием идеи, изложенной в статье [1], в сторону расширяемости и поддержки разных языков программирования.

Глава 1. Обзор и сравнение существующих генераторов программного кода

1.1 Понятие генерации программного кода

Генерация программного кода — это автоматическое создание программного кода специальным приложением, при котором по заданным условиям полностью или частично формируется исходный код программы. Такое специальное приложение называется **генератором кода**. Получается, что это программа, создающая программный код.

Основной сферой применения генераторов программного кода является автоматическое тестирования компиляторов. С помощью них можно обнаружить незаметные ошибки которые могут влиять на работу скомпилированного этими компиляторами программного обеспечения. В сравнение были включены несколько инструментов для тестирования компиляторов.

Также для поиска существующих аналогов был произведен поиск в поисковых системах “Google” и “Google Scholar” по следующим ключевым словам:

- “C++ program generator”
- “program generator”
- “random program generator”
- "java program generator"
- "python program generator"

В обзор не включены различные генераторы привязок к SQL таблицам (Spring Data JPA, jOOQ и подобные), шаблоны для языков разметки (jinja, Django Template Engine) в виду их узкой специализации. Были получены следующие результаты, соответствующие теме дипломной работы:

1.2 Automated C++ Program Generator using English Language Interface

В статье [2] описана программа, генерирующая код на C++ с помощью описания на английском языке. Из описания выделяются ключевые слова и параметры, которым сопоставляются один из множества поддерживаемых шаблонов и алгоритмов, в которые передаются параметры. Поддерживаются арифметические операции, числовые алгоритмы, строковые алгоритмы, алгоритмы над последовательностями и операции ввода-вывода.

1.3 Automatic code generation for C and C++ programming

В статье [3] описана программа, генерирующая код на C++ с помощью описания в виде блок-схем. Элементами блок-схемы является ввод-вывод, условные ветвления и циклы. Следствием этого является ограниченный набор поддерживаемых операций, но в то же время за счет низкоуровневого интерфейса данная программа может генерировать более сложные программы.

1.4 Csmith

Csmith — инструмент для генерации случайных программ на языке программирования C в соответствии со стандартом C99. Используется в тестировании компиляторов, благодаря нему получилось найти более 400 ошибок в компиляторах языка C которые не были известны до этого. Также поддерживает генерацию кода на C++. [4]

1.5 Liveness-Driven Random Program Generation (ldrgen)

Проект, основанный на идеях Csmith, также созданный для тестирования компиляторов. Основная идея - уменьшение количества “мертвого кода” при генерации, что позволяет добиться большего количества инструкций на строку кода и, соответственно, генерировать более компактные программы для тестирования. [5]

1.6 Yarpgen

Инструмент для генерации случайных программ на языке C для тестирования компиляторов. Для тестирования вычисляется хэш всех значений глобальных переменных программы после ее запуска. По сравнению с Csmith код, сгенерированный Yarpgen, более похож на написанный человеком, так как в некоторых случаях сначала генерируется более высокоуровневая модель, которая затем наполняется случайными данными. Также гарантируется отсутствие неопределенного поведения у сгенерированных программ. [6]

1.7 Deepsmith

Инструмент для генерации программ для библиотеки OpenCL на основе машинного обучения. Сгенерированный код похож на написанный человеком так как модель обучена на open-source коде с github. [7]

1.8 SL Random Program Generator

Инструмент для генерации случайных программ на языке Python. Можно настраивать количество инструкций и используемые в выражениях операторы. Имеется веб-версия, гле также можно найти исходны код и грамматику. [8]

1.9 Pyfuzz

Инструмент для генерации случайных программ на языке Python. Используется для тестирования инструментов компиляции и JIT-интерпретации Python-кода. [9]

Имеется веб-версия [10]

1.10 Сравнительный анализ найденных инструментов и статей

Сравнение аналогов будет проведено по следующим критериям:

- “Читаемость кода”, то есть похож ли сгенерированный код на написанный человеком

- Возможность расширения на разные языки программирования (расширяемость)
- Наличие интерфейса для взаимодействия
- Возможность настройки параметров генерации
- Поддержка рандомизации, в частности, возможность настроить начальное значение для генератора случайных чисел

Для статей ответы критерии будут проверяться из описания, так как код реализации отсутствует в открытом доступе.

Сравнение по данным критериям представлено в Таблице 1.

Инструмент	Читаемость	Расширяемость	Интерфейс	Настройка параметров	Рандомизация
Automated C++ Program Generator using English Language Interface	+	+	Natural language	+	-
Automatic code generation for C and C++ programming	+	+	Block-scheme	+	-
Csmith	-	-	CLI[1]	+	+
ldrgen	-	-	CLI	+	+
Yarpgen	-	-	CLI	+	+
Deepsmith	+	-	CLI	+	-
SL Random Program Generator	+	-	Web	+	+/-[2]
Pyfuzz	-	-	Web and CLI	+[3]	+[3]

Таблица 1: Сравнение аналогов.

[1] CLI = Command Line Interface (интерфейс командной строки)

[2] Отсутствует возможность задания seed для генератора случайных значений.

[3] Только в CLI

1.10.1 Результаты сравнения

Инструменты, описанные в статьях [2] и [3], имеют разный интерфейс, но оба имеют ограниченную параметризацию и генерируют читаемый код, однако не имеют поддержки рандомизации.

Csmith, ldrgen, и Yarpgen имеют схожий функционал и недостатки, однако среди них csmith имеет наиболее широкую степень параметризации, Yarpgen и ldrgen имеют меньшую возможность кастомизации.

Deepsmith благодаря машинному обучению генерирует код, максимально схожий с написанным человеком, однако, по этой же причине, обладает небольшой возможностью кастомизации и не поддерживает какую-либо рандомизацию.

SL Random Program Generator имеет удобный веб-интерфейс, но ограниченную возможность настройки и рандомизацию, так же очень ограничено количество поддерживаемых языковых конструкций языка Python.

Pyfuzz так же имеет веб интерфейс, однако в нем совсем отсутствует возможность настройки и рандомизации. В CLI такая возможность присутствует, однако получившиеся программы все же используются для тестирования компиляторов и интерпретаторов, поэтому код получется плохо читаемым для человека.

Глава 2. Разработка инструмента генерации программ

Сравнительный анализ существующих решений для генерации программ показал, что инструменты, используемые на практике, не подходят для учебных целей. Эти инструменты в основном заточены под конкретный язык программирования, у многих слабая кастомизация и нет удобного способа описания шаблона программы. Поэтому было принято решение разработать собственную систему генерации программ для учебных задач.

2.1 Требования к системе генерации

Разрабатываемый инструмент должен обладать следующими возможностями:

- Возможность генерации базовых элементов языка программирования;
- Поддержка генерации кода на разных языках, чтобы иметь возможность использовать данный инструмент в разных обучающих курсах;
- Расширяемость, что включает в себя:
  - Гибкую систему создания шаблонов для генерации задач;
  - Возможность настройки параметров генерации;
  - Высокую вариативность задач, поддержку рандомизации отдельных элементов кода

2.2 Схема генерации программ

На схеме 1 предоставлена схема генерации кода программы.

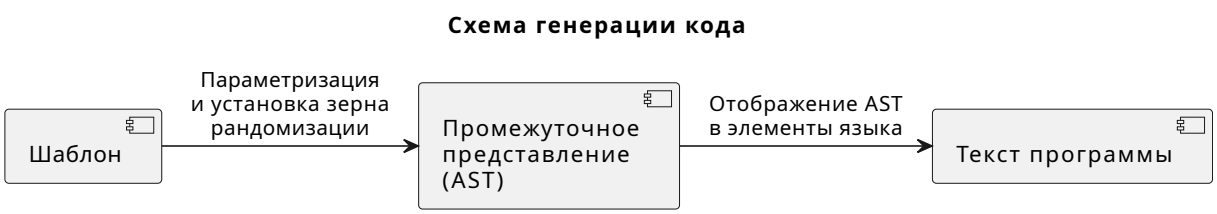

Рис. 1: Схема генерации кода программы

Разберём ее подробнее. На первом шаге выбирается шаблон, из которого будет генерироваться программа. Подробнее о шаблонах можно прочитать в разделе 2.4, на текущем этапе нам достаточно знать, что шаблон — это некая структура, в которую можно подставить начальное значение для генератора псевдослучайных чисел (`seed`) и словарь атрибутов, и получить промежуточное представление, о котором речь пойдет ниже. `seed` представляет собой целое 64-битное число, словарь атрибутов сопоставляет строковому ключу список строковых значений, некоторые из которых могут быть преобразованы в числа. На последнем этапе происходит преобразование промежуточного представления в код. Во время преобразования элементам промежуточного предоставление сопоставляются элементы синтаксиса конкретного языка программирования.

2.3 Промежуточное представление

При генерации кода на разных языках можно выделить конструкции, которые имеют схожую семантику в разных языках, но, при этом, могут отличаться синтаксически. Такими конструкциями являются к примеру:

- Арифметические выражения
- Условные ветвления (`if...else`)
- Множественный выбор/сопоставление с образцом (`switch...case, match, when, case...of`)
- Циклы (`for, while`)
- Объявление и инициализация переменных
- Объявление, определение и вызов функций
- Блоки кода
- Инструкции подключения модулей/библиотек (`import, include`)
- Строковые и числовые литералы

- Объявление классов и структур
- ...

Из данных сущностей можно составить структуру данных, которой можно сопоставить код на разных языках программирования. Такая структура и будет промежуточным представлением. Опишем ее подробнее.

Промежуточное представление является по сути расширением абстрактного синтаксического дерева (AST)

«Дерево абстрактного синтаксиса (ДАС) — в информатике конечное помеченное ориентированное дерево, в котором внутренние вершины сопоставлены (помечены) с операторами языка программирования, а листья — с соответствующими операндами. Таким образом, листья являются пустыми операторами и представляют только переменные и константы.» [11]

В отличие от чистого AST промежуточное представление, используемое в проекте, хранит некоторую информацию о синтаксисе языка (или группы языков), в которые оно в дальнейшем будет интерпретировано. К примеру, в данном дереве могут содержаться скобки в арифметических выражениях, также промежуточное представление хранит метку (тэг), соответствующую группе языков.

2.4 Шаблоны программ

Заметим, что промежуточное представление соответствует конкретной программе на выбранном языке программирования. Однако, для обучающих целей предпочтительнее иметь возможность зафиксировать общую логику программы, но иметь возможность настраивать какие-то параметры для генерации индивидуального варианта программы под каждого студента (аналог вариантов для письменных проверочных работ).

Для этих целей необходимо создать структуру данных («шаблон»), которая бы поддерживала параметризацию и при подстановке параметров преобразовывалась в промежуточное представление (1). Структура шаблона похожа на внутреннее представление, за исключением того, что помимо вершин, соответствующих элементам синтаксиса, в нем также могут присутствовать

специальные вершины, обозначающие случайное значение, какое-либо преобразование поддерева или значение какого-либо атрибута.

2.5 Архитектура системы

Для поддержки одновременной работы с несколькими пользователями и возможности добавления шаблонов и валидации ответов студентов разработанная система имеет клиент-серверную архитектуру с отдельными микросервисами для некоторых компонент. Благодаря этому удалось добиться конфиденциальности правильных ответов и контроля над управлением шаблонов.

Клиентская часть представляет собой веб-сайт с помощью которого пользователь может делать запросы на получение текста или картинки кода по шаблону (задаче). Также имеется возможность делать запросы напрямую к API в формате JSON [12].

Серверная часть состоит из нескольких компонент: непосредственно веб-сервер, система генерации кода и система проверки ответов студентов. В отдельный микросервис выделена система хранения текстов и изображений программ, состоящая из базы данных, находящейся в изолированном окружении.

Схема архитектуры показана на изображении 2, ниже подробнее описаны отдельные компоненты.

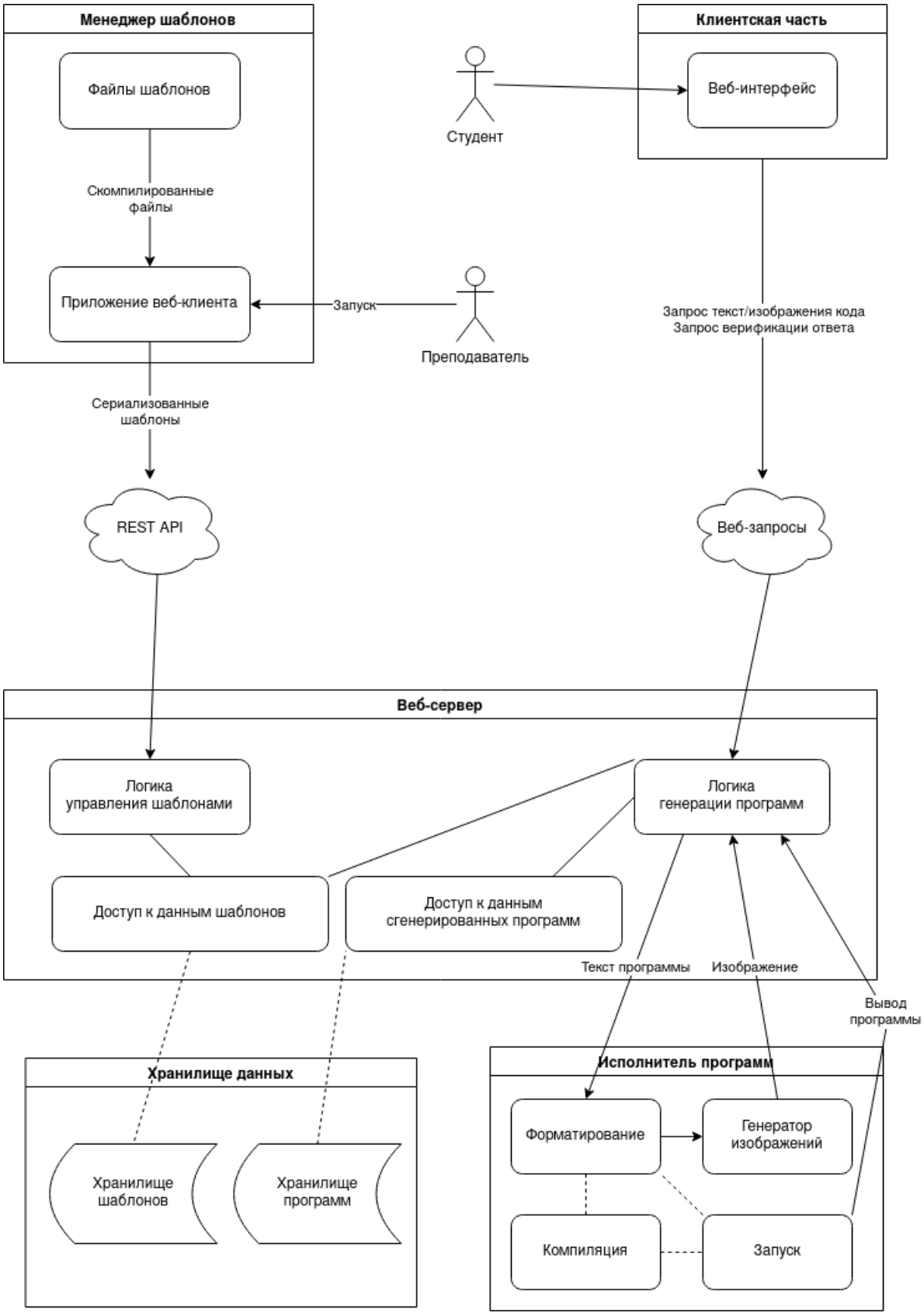

Рис. 2: Архитектура системы генерации программ

2.6 Компоненты системы

2.6.1 Веб-сервер

Веб-сервер разделен на две логические части — управление шаблонами, доступная только преподавателям или администраторам, и генерация программ. Первая отвечает за добавление и удаление шаблонов программ, вторая — за создание и получение изображения и текста программы, а так же за верификацию ответов на задачу.

2.6.2 Исполнитель программ

Исполнитель программ отвечает за обработку сгенерированного кода. Он содержит в себе инструмент форматирования кода, инструмент генерации изображения кода, а также окружение (компилятор и/или интерпретатор), необходимое для выполнения программы. С помощью данного компонента осуществляется генерация изображения и получение вывода программы при ее запуске.

Исполнитель вынесен в отдельный компонент так как ему необходимо специальное окружение (установленные компиляторы, интерпретаторы, средства форматирования).

2.6.3 База данных

Для поддержки работы с несколькими пользователями необходимо хранить шаблоны и данные о сгенерированных программах в базе данных.

В базе сгенерированных программ хранятся параметры генерации, которые, вместе с идентификатором задачи, выступают ключом. Также в ней хранятся текст программы, изображение и вывод программы при запуске.

В базе шаблонов хранятся шаблоны программ, тэг, соответствущий языку или группе языков, для которых написан этот шаблон, и имя шаблона (задачи), которое является ключом.

2.6.4 Сторона клиента

На клиентской стороне пользователь может как взаимодействовать напрямую с сервером, делая запросы через строку браузера и получая ответ в формате HTML, так и через специальные образовательные платформы по типу Moodle [13].

2.6.5 Менеджер шаблонов

Для создания и управления шаблонами программ на стороне клиента используется отельная программа — менеджер шаблонов (Template Manager на схеме). С помощью нее при добавлении текст шаблона преобразуется в машиночитаемый вид *(TODO: вставить ссылку на детали реализации)* и затем, вместе с прочей необходимой информацией (названием, тэгом) отправляется на сервер для сохранения в базу шаблонов (см. 2.6.3).

Глава 3. Реализация инструмента генерации программ

3.1 Используемые технологии

Для разработки инструмента был выбран язык программирования Kotlin [14]. Это язык программирования, разработанный компанией JetBrains в 2010 году. Основными преимуществами этого языка программирования являются:

- Кроссплатформенность
- Возможность интеграции с различными популярными системами сборки (Maven, Gradle, etc.)
- Возможность интеграции с java без переписывания имеющегося кода
- Возможность создания DSL.

Для автоматизации развертывания системы генерации, поддерживания окружения для хранилища и системы проверки используются технологии Docker [15] и docker-compose [16]. Благодаря Docker можно создать изолированное окружение (контейнер) для компонента, а docker-compose позволяет объединять контейнеры в единую локальную сеть. В данном проекте созданы отдельные контейнеры для базы данных и веб-сервера.

3.1.1 Менеджер шаблонов

Для хранения и пересылки шаблонов была добавлена поддержка конвертации шаблона в формат JSON — один из самых распространенных форматов обмена данных, который также может использоваться и для хранения информации (см 3.1.3). Из-за сложной структуры классов, в виде экземпляров которых представляется шаблон (3.2.1), для сериализации шаблонов в JSON была выбрана библиотека `Kotlinx.serialization` [17], которая поддерживает сериализацию объектов языка программирования Kotlin и умеет работать с наследованием и полиморфизмом.

Для отправки запросов на создание и удаление через Интернет был использован Http -клиент фреймворка Ktor [18], так как данный клинт поддерживает сериализацию тела запроса (и ответа) с помощью `Kotlinx.serialization`.

3.1.2 Веб-сервер

Для реализации веб-сервера в связи с использованием для сериализации `Kotlinx.serialization` было принято решение использовать Http -сервер фреймворка Ktor [18], который поддерживает данный способ сериализации запросов, а также имеет простой и элегантный API и возможность расширения с помощью плагинов.

3.1.3 База данных

Так как шаблоны программ конвертируются и передаются в формате JSON, то было принято решение переиспользовать имеющуюся логику и хранить их также в этом формате. В данном проекте используется `MongoDB` [19] — база данных, которая работает именно с форматом JSON.

Для работы с `MongoDB` из серверного кода была выбрана библиотека `kmongo`. Данная библиотека является самой популярной библиотекой для работы с `MongoDB` из языка `Kotlin`, также она поддерживает конвертацию в JSON и обратно с помощью `Kotlinx.serialization`, что также упрощает работу с шаблонами.

Поля, которые хранятся в соответствующих коллекциях, описаны в 2.6.3. Однако в данной библиотеке отсутствует поддержка хранение сырых байтов в базе данных, поэтому для хранения изображения кода в базе байты изображения, с помощью алгоритма Base64[20], преобразуются в строку и в таком виде сохраняются в базу.

3.1.4 Исполнитель программ

Для компиляции (при необходимости), выполнения, форматирования кода и генерации изображений используются отдельные программы, запускающиеся во внешнем окружении. По соображениям безопасности и для ограничения нагрузки на инфраструктуру, на которой работает сервер, одновременно может работать только один исполнитель, для этого в веб-сервере код, выполняющий программы во внешнем окружении, запускается под блокировкой мьютекса. Для файлов, создаваемых во время выполнения, выделен

отдельный каталог.

Для генерации изображений по коду используется библиотека `python-pygments`[21] и поставляемая с ней программа `pygmentize`. Данная библиотека написана на языке программирования Python[22] и поддерживает большинство популярных языков программирования[23]. Изображения генерируются в формате png.

Для остальных задач из списка используются утилиты, специфичные для конкретного языка программирования. Пример таких утилит для языка программирования Python[22]:

- Утилита для форматирования: `autopep8`[24]
- Компилятор: компиляция не требуется
- Утилита для выполнения кода: стандартный интерпретатор `python` версии 3.10

Для форматирования, генерации изображения и компиляции установлено ограничение по времени в 10 секунд, для выполнения кода — 30 секунд.

3.2 Шаблоны программ

3.2.1 Внутреннее представление шаблона

Объект шаблона представляет собой экземпляр класса `ProgramTemplate`, параметризованного тэгом, соответствующим языку или группе языков программирования с общими свойствами. Все упомянутые далее классы также параметризованы тэгом.

Сам тэг предоставляет собой сущность `object` (представляющую собой singleton [25]) языка программирования Kotlin, наследованный от класса `ProgramLanguageTag` и набора интерфейсов, соответствующих особенностям языка. Примером такого интерфейса является `DynamicTyping`, обозначающий, что данный язык программирования имеет динамическую типизацию [26]. С точки зрения синтаксиса это, к примеру, означает, что не обязательно указывать типы аргументов при объявлении функции.

Внутри данного объекта содержится объект класса `ProgramGlobalTemplate`. Этот и парный к нему класс `ProgramLocalTemplate` сопоставляются глобальной и локальной области видимости. Внутри объектов данных классов содержится список элементов программы, находящихся в соответствующей области видимости.

Примеры элементов программы:

- `VariableTemplate` — ссылка на переменную.
- `AssigmentTeamplate`, `VariableDefinitionTemplate` — присвоение значения в переменную и объявление переменной.
- `FunctionDefinitionTemplate`, `FunctionCallTemplate` — объявление и вызов функции.
- `IfElseInstuctionTemplate`, `WhileTemplate` — условные ветвления и циклы.
- `ProgramConstantTemplate` — общий класс для разных типов констант. Примеры наследников данного класса:
  - `NumConstantTemplate`, `StringConstantTemplate` — обычные числовые и строковые константы.
  - `RandomNumConstantTemplate`, `RandomStringConstantTemplate`, `RandomChoiceConstant`, `RandomChoiceStringConstant` — случайные числовые и строковые значения, которые генерируются и транслируются в обычные константы во время преобразования шаблона в промежуточное предоставление. Первые два класса генерируют значения согласно переданным параметрам, последние два — выбирая из фиксированного списка значений.
  - `NumAttributeRef`, `StringAttributeRef` — значения атрибутов, передаваемых при конвертации в промежуточное представление.
- `ProgramConstantListTemplate` — общий класс для списка констант, аналогично `ProgramConstantTemplate`.

- `ProgramExpressionTemplate` — общий класс для выражений. Является, в частности, предком классов `ProgramConstantTemplate`, `ProgramConstantListTemplate` и `VariableTemplate`. Также его наследником является `BinOpExpressionTemplate` — класс, соответствующий бинарным операциям.

- `RepeatLocalScopeTemplate`, `RepeatGlobalScopeTemplate` — служебные классы, которые при генерации промежуточного представления повторяют переданный `scope` заданное число раз. При генерации внутреннего `scope` также можно получить доступ к номеру текущей итерации. Также есть классы `ForeachRepeatLocalScopeTemplate` и `ForeachRepeatGlobalScopeTemplate`, которые перебирают и передают в функцию-построитель элементы списка. Подробнее об этом в 3.2.2.

3.2.2 Конструирование шаблонов

Для создания шаблонов программ в дипломном проекте используется язык программирования Kotlin [14]. Шаблон представляет собой объект языка программирования Kotlin, созданный с помощью специальных функций-построителей. Каждая такая функция соответствует классу из 3.2.1 либо строит его часть.

Функции, соответствующие инструкциям, добавляют созданный объект в `scope`, а соответствующие выражениям — возвращают сгенерированный объект. Данная особенность хорошо видна на примере функций `funcCall` и `addFuncCall`. Так как вызов функции может быть как и самостоятельной инструкцией, так и частью выражения, то первая функция возвращает объект и предназначена для использования в выражениях, а вторая — добавляет вызов в список элементов `scope`.

Пример кода, конструирующего шаблон:

```
val task = ProgramTemplate<PythonTag> {
    val xVar = variable("x")
    val stringLen = randomNumConstant(10, 20)
    val randomString = randomStringConstant(stringLen)
```

```
        xVar setTo randomString
        `if`(funcCall("len", xVar) lt constant(15)) {
            addFuncCall("print", constant("small"))
        }.`else` {
            addFuncCall("print", constant("long"))
        }
}
```

Внутреннее представление результата конструирования:

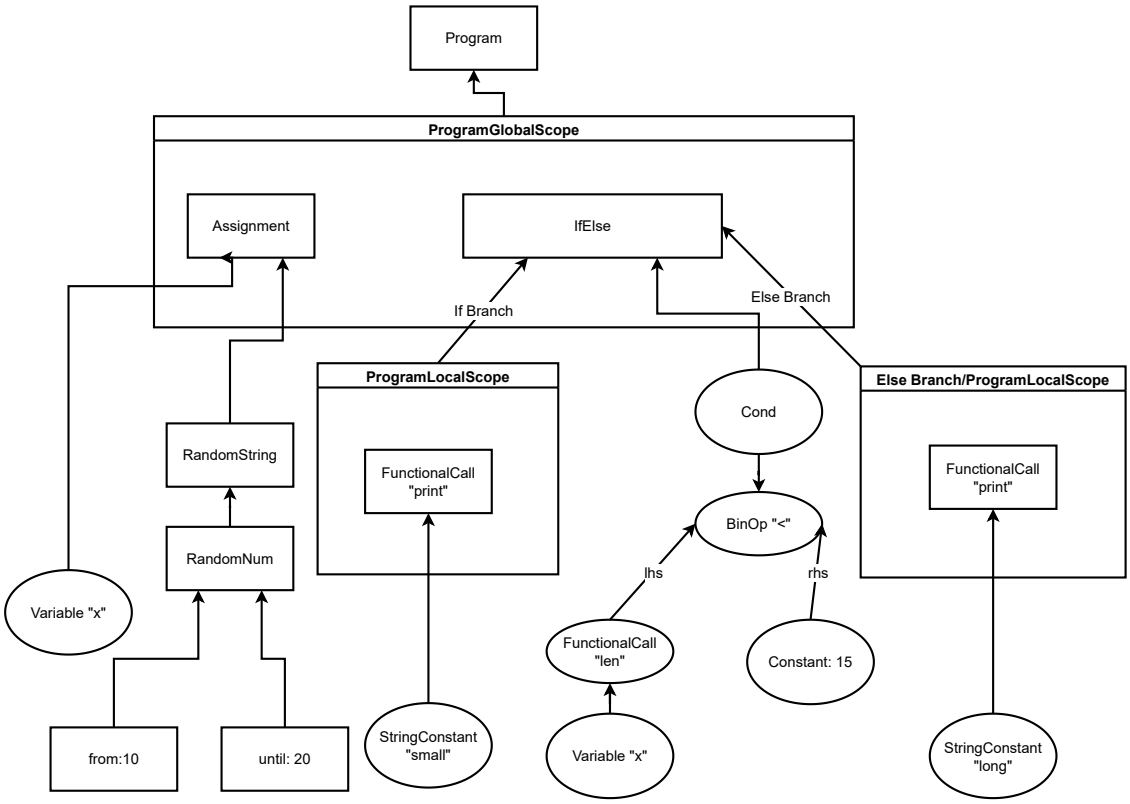

Рис. 3: Внутреннее представление шаблона

Благодаря возможностям языка Kotlin [27], можно воспроизводить в шаблоне древовидную структуру, где параметры функции — это параметры в вершине AST, а последний аргумент (ламбда-функция, переданная за пределами скобок) описывает поддерево. Это используется в функциях, которые создают новый `scope`, к примеру функции ‘`if`‘, ‘`elif`‘, ‘`else`‘, ‘`while`‘, `addFunctionalDef`. Также это используется в функциях, которые конструи-

руют служебные классы. Примером такого класса и соответствующей ему функции может являться `RepeatLocalScopeTemplate` и соответствующая ему функция `repeat`, которая позволяет повторить несколько раз генерацию некоторого куска шаблона. К примеру шаблон

```
repeat(2) { i ->
    addFuncCall("print", i)
}
```

сгенерирует следующий код (для языка Python):

```
print(0)
print(1)
```

Также в построении используется такая особенность языка Kotlin, как инфиксная нотация [28], позволяющая вызывать функции с двумя аргументами как бинарные операторы, то есть размещать имя функции между аргументами. С помощью этого можно конструировать присваивания и инициализацию переменной (функция `setTo` в примере выше) и бинарные операторы в арифметических выражениях (`lt` в примере выше).

3.3 Промежуточное представление (AST)

Внутреннее предоставление AST практически не отличается от предоставления шаблона, за исключением того, что в AST отсутствуют служебные вершины, отвечающие за генерацию случайных данных и обращение к атрибутам.

3.4 Генерация текста программы

Для генерации кода на конкретном языке программирования необходимо реализовать интерфейс `CodeMapper`, который выглядит следующим образом:

```
interface CodeMapper<LanguageTag : ProgramLanguageTag> {
    fun generateCode(program: Program<in LanguageTag>): String
}
```

По данному интерфейсу видно, что тип `Program` (а, соответственно, и все классы в промежуточном предоставлении) является *контравариантным* [29], то есть в данный `CodeMapper` можно передать как представление программы на языке Python, так и более общие программы, предназначенные для трансляции в несколько языков.

3.5 API

Для взаимодействия с сервером используется Web API.

Для генерации и получения изображений и текстов программ доступны следующие методы API:

- `/get_source` — получить отформатированный текст программы. Возвращает текст программы при наличии задачи (см. передаваемые параметры).
- `/get_image` — получить изображение программы. Возвращает `html` с изображением.
- `/get_image_bytes_png` — получить сырые байты изображения. Используется в ссылке в `html` с изображением.

Вышеописанные методы принимают следующие параметры в строке URL:

- `task` — имя задачи (шаблона), из которого будет сгенерирована программа.
- `seed` — значение, используемое как зерно рандомизации при генерации программы.
- Прочие параметры передаются как атрибуты генерации программы из шаблона.

Для проверки ответа студента используется метод `/check_answer`, принимающий те же параметры, что и методы генерации кода, а также параметр

`answer`, соответствующий ответу студента. Подробнее о проверке ответов в 3.6.

Для управления (добавления и удаления) шаблонами программ доступны следующие методы:

- `/add_task` — добавить задачу. Принимает в теле запроса параметры, описанные в 2.6.3: имя задачи, тэг и шаблон в формате JSON. Данный метод просто сохраняет переданные данные в базу
- `/delete_task` — удалить задачу. Принимает в теле запроса структуру, содержащую имя удаляемой задачи, в формате JSON. Данный метод удаляет задачу (шаблон) из базы.

3.6 Проверка ответов

Для улучшения интеграции с образовательными платформами в инструмент была добавлена поддержка проверки ответов студентов. Ответом студента для описываемых задач является вывод программы при ее выполнении. Для проверки ответа студента генерируется (или запрашивается из базы данных) реальный вывод программы (см 2.6.2). Далее реальный вывод построчно сравнивается с ответом студента, в качестве результата проверки возвращается процент совпавших строк.

Заключение

Целью и основной задачей дипломной работы было создание генератора программ для обучения программированию. В ходе выполнения работы были получены следующие результаты:

1. Было проведено исследование существующих решений по генерации программ на предмет их применимости в обучающих целях. В ходе исследования были выявлены достоинства и недостатки имеющихся решений.

2. Были сформулированы требования к генератору обучающих программ, учитывающие используемые модели в аналогах, их достоинства и недостатки. Из основных недостатков были выявлены: отсутствие возможности задать основную логику программы; малофункциональные модели задания шаблона кода при их наличии; отсутствие веб-интерфейса; отсутствие возможности хранения шаблонов и сгенерированных программ; отсутствие возможности генерации изображений.

3. Создан предметно-ориентированный язык для описания задач (шаблонов кода), на основе которых генерируются примеры программ.

4. Был реализован инструмент генерации программ, поддерживающий управление шаблонами кода, имеющий потенциал к расширению в сторону поддержки новых языков программирования, в настоящий момент поддерживающий генерацию кода на языке Python с помощью основных синтаксических конструкций.

Таким образом, цель данной работы достигнута в полном объеме.

Список литературы

[1] А Хафизова и М Заславский. «Генератор случайных программ как инструмент обучения программированию». В: *СБОРНИК ДОКЛАДОВ СТУДЕНТОВ И АСПИРАНТОВ НА КОНФЕРЕНЦИИ ПРОФЕССОРСКО-ПРЕПОДАВАТЕЛЬСКОГО СОСТАВА*. 2019, с. 191.

[2] Ambuj Kumar и Saroj Kaushik. «Automated C++ Program Generator using English Language Interface». В: ().

[3] S. Patade и др. «AUTOMATIC CODE GENERATION FOR C AND C++ PROGRAMMING». В: *IRJET* 08 (05 2021), с. 4732—4736.

[4] Xuejun Yang и др. *Csmith*. URL: `https://embed.cs.utah.edu/csmith/` (дата обр. 10.05.2022).

[5] Gergö Barany. «Liveness-driven random program generation». В: *International Symposium on Logic-Based Program Synthesis and Transformation*. Springer. 2017, с. 112—127.

[6] Intel. *Yarpgen*. URL: `https://github.com/intel/yarpgen` (дата обр. 10.05.2022).

[7] Chris Cummins и др. «Compiler fuzzing through deep learning». В: *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2018, с. 95—105.

[8] Ariel Baruch. *SL Random Program Generator*. URL: `https://www.cs.bgu.ac.il/~arielbar/sl/#/code` (дата обр. 10.05.2022).

[9] Steven Myint. *pyfuzz*. URL: `https://github.com/myint/pyfuzz` (дата обр. 10.05.2022).

[10] Steven Myint. *Random Python Program Generator*. URL: `https://www.4geeks.de/cgi-bin/webgen.py` (дата обр. 10.05.2022).

[11] *AST*. URL: `https://en.wikipedia.org/wiki/Abstract_syntax_tree` (дата обр. 15.05.2022).

[12] ECMA International. *Standard ECMA-404. The JSON Data Interchange Format*. 2017.

[13] *Moodle*. URL: https://moodle.org/?lang=ru (дата обр. 15.05.2022).

[14] Jemerov D. и Isakova S. *Kotlin in action*. Manning Publications Company, 2017.

[15] *Docker*. URL: https://www.docker.com/ (дата обр. 10.05.2022).

[16] *docker-compose*. URL: https://docs.docker.com/compose/ (дата обр. 10.05.2022).

[17] JetBrains. *Kotlinx.serialization*. URL: https://github.com/Kotlin/kotlinx.serialization (дата обр. 18.05.2022).

[18] JetBrains. *Ktor*. URL: https://ktor.io/ (дата обр. 10.05.2022).

[19] MongoDB. *MongoDB*. URL: https://www.mongodb.com/try/download/community (дата обр. 18.05.2022).

[20] Simon Josefsson. *The Base16, Base32, and Base64 Data Encodings*. RFC 4648. Окт. 2006. DOI: 10.17487/RFC4648. URL: https://www.rfc-editor.org/info/rfc4648.

[21] Georg Brandl и Matthäus Chajdas. *Pygments*. URL: https://pygments.org (дата обр. 18.05.2022).

[22] *Python*. URL: https://www.python.org/ (дата обр. 18.05.2022).

[23] Georg Brandl и Matthäus Chajdas. *Pygments: supported languages*. URL: https://pygments.org/languages (дата обр. 18.05.2022).

[24] Hideo Hattori и Steven Myint. *autopep8*. URL: https://pypi.org/project/autopep8/ (дата обр. 18.05.2022).

[25] *Singleton*. URL: https://ru.wikipedia.org/wiki/%D0%9E%D0%B4%D0%B8%D0%BD%D0%BE%D1%87%D0%BA%D0%B0_(%D1%88%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) (дата обр. 29.05.2022).

[26] *Динамическая типизация*. URL: https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) (дата обр. 29.05.2022).

[27] Jemerov D. и Isakova S. «Kotlin in action.» В: Manning Publications Company, 2017. Гл. 11. Конструирование DSL, с. 346—380.

[28] *Kotlin infix notation*. URL: https://kotlinlang.org/docs/functions.html#infix-notation (дата обр. 29.05.2022).

[29] Jemerov D. и Isakova S. «Kotlin in action.» В: Manning Publications Company, 2017. Гл. 9.3. Вариантность: обобщенные типы и подтипы, с. 300—310.