Министерство образования и науки Российской Федерации
Государственное образовательное учреждение высшего профессионального образования

Дальневосточный государственный университет

Институт математики и компьютерных наук

Кафедра информатики

Зенкина Александра Олеговна

БИБЛИОТЕКА АЛГОРИТМОВ ДЛЯ ГЕНЕРАЦИИ ЗАДАЧ

ДИПЛОМНАЯ РАБОТА

«__» _____ 20__г.

Студент группы 256 _____
(подпись)

Руководитель ВКР _____
(ученое звание, должность)

_____ _____
(подпись) (и.о.ф.)

«__» _____ 20__г.

Консультант _____
(ученое звание, должность)

_____ _____
(подпись) (и.о.ф.)

«__» _____ 20__г.

«Допустить к защите»

Заведующий кафедрой _____
(ученое звание)

_____ _____
(подпись) (и.о.ф.)

«__» _____ 20__г.

Защищена в ГАК с оценкой _____

Секретарь ГАК

_____ _____
подпись И.О.Фамилия

г. Владивосток

2010

Содержание

Аннотация

Данная работа посвящена изучению метода генерации тестовых заданий на основе использования языка программирования как средства составления шаблонов. В отличие от других подходов, основанных на вызове внутри шаблона алгоритма, специально реализованного для данной задачи, этот метод позволяет создавать шаблон на основе средств выбранного языка программирования и существующей библиотеки функций, чем достигает большей гибкости при описании различных типов задач. На основании результатов использования данного метода выделен набор наиболее часто используемых функций, который может быть полезен людям, занимающимся составлением тестов и мало знакомым с программированием.

1 Введение

1.1 Глоссарий

Варианты заданий — это набор сходных по структуре заданий, направленных на проверку одного и того же знания или навыка, отличающихся друг от друга условием, вопросом или вариантами ответа.

Варианты ответа — это множество выражений, содержащее как правильные, так и неправильные ответы на вопрос задания.

Вопрос — это предложение, определяющее соответствие между условием и множеством ответов.

Генерация задания — создание тестового задания на основе шаблона при помощи случайных величин.

Генерация результата — один из этапов генерации задания, в котором происходит построение дерева ответа.

Дерево выражения — это представление выражения, имеющее иерархическую структуру.

Дистракторы — варианты ответов в заданиях с выбором, не являющиеся правильными решением, но внешне близкие к нему [28].

Длина выражения — в данной работе это число листьев в дереве выражения.

Задание — это базовая единица проверки знаний. Под заданием в данной работе будем понимать тестовое задание, состоящее из условия задания, вопроса и вариантов ответа.

Ответ — некоторое выражение, значение или графическое представление.

Результат тестирования — последовательность правильных и неправильных ответов, позволяющая сделать вывод о степени владения материалом теста, выявить наиболее трудные задания и наиболее частые ошибки, осуществить обратную связь между преподавателем и учащимся.

Тест — это стандартизованный метод диагностики уровня и структуры подготовленности [10].

Тестирование — это процесс выполнения заданий теста.

Трансформация результата — преобразование результирующего дерева выражения по некоторому правилу.

Ход решения — это последовательность преобразований, приближающая к получению ответа.

Шаблон задания — это описание задания, содержащее обычный текст, параметризованные величины, указания на алгоритмы их подстановки.

1.2 Описание предметной области

Тестирование как метод проверки знаний стало широко использоваться с начала XX века [4]. Предпосылкой к этому явилось усовершенствование технологий производства, потребовавшее большого количества квалифицированных кадров, в силу чего возросли требования к образованию. Образовательная система должна была позволять обучать много людей и при этом — обучать качественно.

Контроль знаний учащихся, служащий для осуществления обратной связи между преподавателем и студентами, является трудоемким процессом, который включает в себя составление заданий по учебному материалу и проверку результатов их выполнения, поэтому нуждается в оптимизации. Примером такой оптимизации является применение тестов.

Обоснование этого подхода приведено в работах доктора педагогических наук, профессора В. И. Аванесова [10]. В частности, он пишет: «Учебные вопросы многословны и порождают ответы, полные и неполные, правильные и неправильные, разные по форме, содержанию и по структуре, вследствие чего оценка таких ответов требует обязательного участия преподавателя и сопровождается некоторой долей субъективизма. Вопросы и ответы на них иногда бывают столь неопределенными и многословными, что для выявления их истинности требуются большие затраты интеллектуальной энергии, в то время как технологичная методика тестирования предполагает четкую

и быструю дифференцируемость ответов», что в свою очередь облегчает их проверку.

Следующий шаг в оптимизации проверки знаний был сделан с введением компьютерного тестирования, которое позволило полностью автоматизировать проверку результатов. Таким образом, работа преподавателя по контролю знаний свелась к составлению тестов. Но даже в таком виде она все еще требует значительного труда, поскольку, как правило, требуется разработать не один, а сразу несколько вариантов заданий для того, чтобы обеспечить самостоятельную работу учащихся.

Другой проблемой является то, что даже к нескольким, но не меняющимся со временем вариантам заданий студенты приспосабливаются при помощи шпаргалок и подсказок друг другу. Кроме того, из-за повторяемости вопросов происходит механическое запоминание правильного варианта ответа, в силу чего тестирование перестает отвечать задаче контроля знаний [24].

Таким образом, преподавателю нужно не только составлять многочисленные и разнообразные задания, но также со временем обновлять их. Автоматизация этой деятельности возможна с помощью генерации заданий по заранее составленному описанию (шаблону). Однако сам метод описания также должен быть предварительно разработан. Его разработка является качественно сложной задачей, в общем случае — невыполнимой.

По этой проблеме ведутся исследования, в которых приоритетными могут являться различные факторы, как-то: применимость к конкретным областям знаний, максимальная простота использования, универсальность при составлении разных типов заданий и т.п. Максимальная простота может быть достигнута с помощью описаний на основе графического интерфейса. Однако область применения таких описаний весьма ограничена. В большинстве своем описания составляют для конкретного типа задач, что позволяет ограничиться наличием в текстовом шаблоне набора параметризованных констант и вызова

решающей функции, специализированной под данную задачу.

Для описания более широкого класса задач необходимо использовать более выразительные средства. В качестве такого средства может быть выбран язык программирования. Изучению именно этого подхода посвящена моя работа.

1.3 Неформальная постановка задачи

В рамках данной работы требуется сделать следующее:

- применить существующий язык программирования либо разработать собственный для создания шаблонов;
- разработать библиотеку функций для осуществления генерации заданий;
- применить полученный таким образом метод генерации к широкому классу задач;
- сделать вывод об эффективности метода в целом и о возможности использования его наиболее полезных функций пользователями, мало знакомыми с программированием.

1.4 Обзор существующих методов решения

Прежде всего, нужно сказать, что до сих пор очень распространено составление тестов вручную. Для систем, которые в той или иной степени автоматизируют этот процесс, можно приведем их основные типы.

Во-первых, это редакторы тестов [25]. Они позволяют управлять внешним видом теста: формой ответа, наличием подсказок, добавлением графического материала, установкой сложности задания, однако содержание теста необходимо вводить вручную. Этим достигается гибкость в описании заданий, поскольку пользователь ничем не ограничен и может свободно набирать то задание, которое ему нужно.

Однако это пример неавтоматизированного составления задач, поскольку он не освобождает пользователя от необходимости составлять множество однотипных заданий.

Во-вторых, это редакторы курсов, предоставляющие возможность генерировать тесты на основе ранее введенного материала курса [15]. Как правило, они используют базу данных с понятиями и определениями из курса и базу возможных вопросов и ответов. При этом вопросы могут комбинироваться при помощи логических связок: И, ИЛИ, НЕ, благодаря чему увеличивается разнообразие заданий. Они рассчитаны больше на преподавателей, не связанных с программированием, поэтому важной чертой для них является наличие простого и удобного графического интерфейса, что накладывает ограничения на содержание задания. Так, например, генерация выражений и формул в них уже не так просто осуществима.

В-третьих, существуют различные генераторы выражений, создающие формулы и выражения на основе некоторого описания. Примером может служить библиотека MatLab для генерации случайных выражений [2] или Pinery, программа для генерации выражений на основе регулярных выражений Perl [17]. Однако такие генераторы могут быть использованы лишь как вспомогательные средства при генерации тестов, поскольку генерируют не задание в целом, а лишь некоторую его часть. При этом, чтобы их применять, пользователь должен быть знаком с программированием.

В-четвертых, есть генераторы индивидуальных домашних заданий (ИДЗ) и тестов: например, генератор ИДЗ по мат. анализу [12] и генератор тестов по информатике [24]. Эти генераторы могут отличаться друг от друга функциональностью, алгоритмами и реализацией. В частности, приведенный выше генератор ИДЗ по мат. анализу позволяет генерировать условия заданий, решение и вычислять ответ, но в нем не предусмотрена генерация нескольких вариантов ответов. Генератор тестов по информатике позволяет генерировать условия заданий и

варианты ответов, но в нем не предусмотрено построение хода решения.

Эти программы объединяет механизм составления шаблона. Как правило, шаблон содержит некоторое число параметризованных величин, для которых указана область их изменения и, возможно, алгоритм генерации. В процессе генерации величины заменяются константами из соответствующих областей и в зависимости от этих конкретных значений вводятся грамматические изменения в формулировку задачи. Кроме того, шаблон содержит вызов алгоритма решения задачи, по которому после генерации параметров вычисляется правильный ответ.

При этом невозможно однозначно оценить простоту и гибкость таких систем. Если система ориентирована на какую-то определенную область и определенный вид теста, как, например, генератор ИДЗ по мат. анализу, то шаблоны тестов могут быть жестко зашиты внутри нее без возможности их изменения, что удобно с точки зрения использования: нужно лишь выбрать нужный шаблон и сгенерировать тест. Если область применения системы расширяется, то возникает необходимость не только в написании новых шаблонов, но и, как правило, в доработке самой системы, добавлении в нее новых функций и алгоритмов. В этом случае либо пользователь должен обладать навыками программиста и самостоятельно вносить изменения в систему, либо время от времени обращаться к программисту, что не очень удобно с точки зрения использования системы. Поскольку перед автором работы поставлена задача создания гибкой системы генерации тестов, то создаваемый им проект также не освобожден от этого недостатка.

В ходе обзора автору не удалось выяснить, поддерживают ли генераторы ИДЗ и тестов возможность генерации дистракторов — вариантов ответов, которые являются результатом определенной ошибки в рассуждениях, основанной на недостаточном понимании материала. В свою очередь, такая возможность является одним из приоритетов в данной работе.

Кроме того, особенностью данной работы является использование

для составления шаблонов языка программирования, что позволяет комбинировать уже существующие функции для описания новой задачи, а не реализовывать всякий раз программно новый алгоритм. Хотя перед автором не стоит задачи построения хода решения для заданий теста, использование языка программирования имеет потенциал к выполнению также и этой задачи.

Следует также оговорить еще одну особенность создаваемой системы — это достаточно большой объем текста, в котором выражается описание задания, что является оборотной стороной гибкости подхода. Однако этот недостаток компенсируется по мере увеличения числа сгенерированных по данному описанию тестов.

Ниже приведена сводная таблица характеристик рассмотренных систем.

	Редактор тестов	Редактор курсов	Генератор выражений	Генератор ИДЗ	Создаваемая система
Генерация условий	нет	есть	есть	есть	есть
Генерация ответов	нет	есть	нет	возможна	есть
Генерация дистракторов	нет	возможна	нет	возможна	есть
Построение хода решения	нет	нет	нет	есть	возможно
Требования к пользователю	низкие	низкие	высокие	разные	высокие
Универсальность для разных видов заданий	высокая	низкая	низкая	разная	высокая
Объем текста шаблона	низкий	низкий	низкий	низкий	высокий

Таблица 1: Характеристики систем создания тестов

2 Требования к окружению

2.1 Требования к аппаратному обеспечению

Для работы системы необходима стандартная конфигурация компьютера:

- системный блок;
- монитор;
- клавиатура.

2.2 Требования к программному обеспечению

На рабочем месте пользователя должны быть установлены:

- операционная система;
- интерпретатор Perl;
- библиотека Template::Toolkit;
- оболочка командной строки;
- текстовый редактор;
- браузер.

Тестирование производилось на системе со следующей конфигурацией:

- операционная система MS Windows XP Professional;
- интерпретатор Strawberry Perl 5.10.1;
- библиотека Template::Toolkit 2.22;
- файловый менеджер Far Manager 1.70 alpha 6;
- браузер Mozilla Firefox 3.5.8.

2.3 Требования к пользователям

Пользователи должны:

- уметь работать с командной строкой;
- знать синтаксис языка Template::Toolkit.

Желательно также умение программировать на языке Perl.

3 Функциональные требования

Система должна позволять пользователю генерировать тестовые задания, главным образом, из областей математики и информатики. Более детально, система должна:

- обрабатывать текстовые файлы шаблонов с включенными в них параметрами, вызовами функций, конструкциями языка описания;
- генерировать значения параметров и дерево ответа;
- видоизменять результат для создания дистракторов;
- вычислять значения полученных вариантов ответа;
- производить вывод таким образом сгенерированного задания в файл или на терминал.

Интерфейс, использующийся при составлении шаблонов, должен включать в себя:

1. Классы:
    - генератор;
    - переменные;
    - функции;
    - результат.
2. Функции вызова конструкторов соответствующих классов.
3. Методы:
    - генерации;
    - перевода результата в строковую форму;
    - вычисления значения;
    - преобразования результата.

4. Средства языка программирования:

    - тэги;
    - операторы присваивания, вызова функций;
    - объявления переменных.

4 Требования к интерфейсу

Создание файлов шаблонов может осуществляться в любом текстовом редакторе. Запуск системы должен осуществляться при помощи командной сроки. Имя файла шаблона должно передаваться системе в качестве параметра при запуске. Получившийся в результате обработки шаблона тест система должна выводить на терминал.

5 Проект

5.1 Средства реализации

Система, осуществляющая генерацию заданий, реализована на языке программирования Perl [9]. Его преимущества заключаются в том, что он является динамическим, объектно-ориентированным, поддерживает анонимные функции и содержит мощный механизм поиска по шаблону.

После попытки разработать свой собственный язык для создания шаблонов заданий был выбран язык программирования Template::Toolkit [8]. Он реализован как набор модулей, написанных на языке Perl, и применяется, в основном, для обработки шаблонов статических и динамических Web-страниц.

Его достоинство в том, что он достаточно прост, но в то же время в нем реализованы основные полезные при составлении шаблонов возможности: арифметические и логические операции, присваивание, ветвление, циклы, объявления скалярных переменных, списков, хэшей, виртуальные методы для работы с ними и т.д.

Кроме того, код, написанный на нем, может быть встроен непосредственно в текст шаблона при помощи тегов, что значительно облегчает задачу написания и обработки шаблона.

Также этот язык реализован как библиотека Perl, благодаря чему его интерпретация осуществляется автоматически средствами самого Perl, что весьма удобно для использования его возможностей внутри создаваемой системы.

5.2 Модули и алгоритмы

5.2.1 Классы

Система классов данного проекта содержит 3 базовых класса, от которых происходит наследование основных использующихся при генерации элементов: переменных, функций, выражений, операторов. Существует также набор классов, содержащих в себе некоторые из перечисленных элементов, например: преобразования, выражения в столбик, таблица символов. Все эти классы связаны между собой при помощи генератора. Кроме того, существуют независимые от других, «библиотечные» классы. В них реализованы полезные функции для работы с данными.

Диаграмма классов приведена ниже.

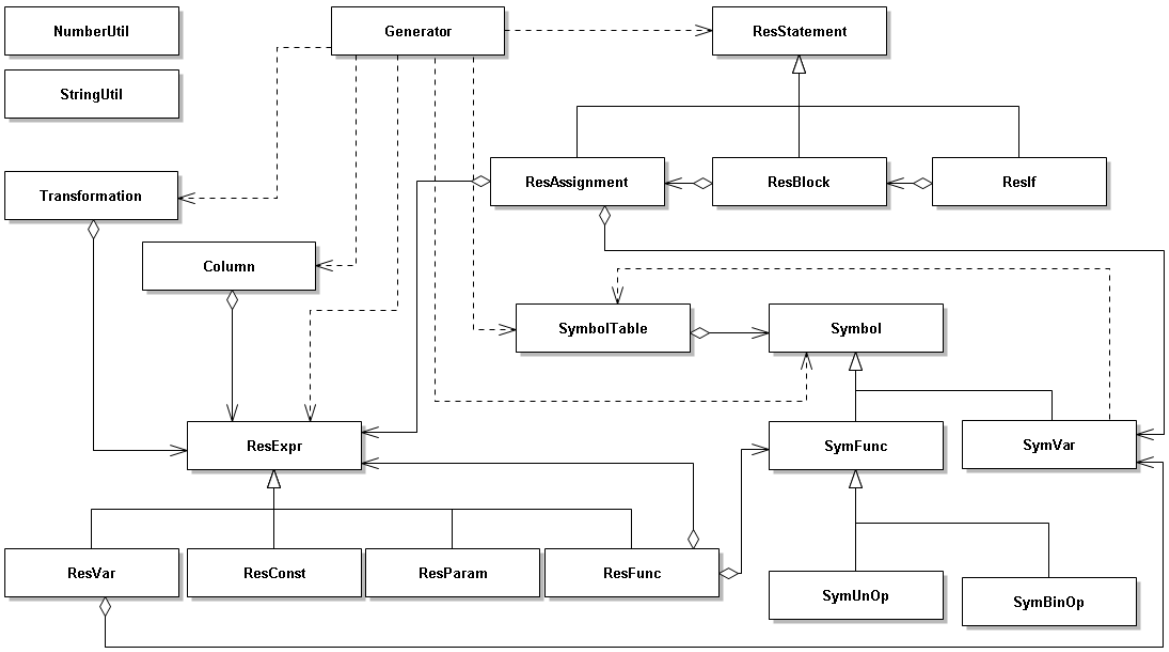

Рис. 1: Диаграмма классов

В качестве примечания уточним следующее:

- Каждый класс создается при помощи вызова конструктора `new`, поэтому не будем указывать его в описании методов.
- На данный момент типизация констант, переменных, функций, выражений и пр. не поддерживается, и поле *type* для объектов соответствующих классов содержит значение `any`.

Рассмотрим подробнее каждый класс:

1. **Symbol** — базовый класс для элементов таблицы символов: переменных, функций, операций.
2. **SymVar** — класс переменных, унаследован от **Symbol**.

   Свойства:

   - *name* — имя переменной;
   - *value* — значение переменной;
   - *type* — тип значения.

   Методы:

   - `value` — возвращает значение переменной.
3. **SymFunc** — класс функций, унаследован от **Symbol**, является базовым классом для классов **SymUnOp** и **SymBinOp**.

   Свойства:

   - *name* — имя функции;
   - *params* — список типов параметров;
   - *type* — тип возвращаемого значения;
   - *body* — тело функции. Как правило, представляет собой анонимную функцию Perl.

Методы:

- paramsCount — возвращает количество параметров;
- isUnOp — возвращает 0, функция не является унарной операцией;
- isBinOp — возвращает 0, функция не является бинарной операцией.

4. **SymUnOp** — класс унарных операций, унаследован от **SymFunc**.

   Методы:

   - isUnOp — перегруженный метод, возвращает 1.

5. **SymBinOp** — класс бинарных операций, унаследован от **SymFunc**.

   Свойства:

   - *commutative* — флаг коммутативности бинарной операции;
   - *priority* — приоритет бинарной операции.

   Методы:

   - isBinOp — перегруженный метод, возвращает 1.

6. **SymbolTable** — класс таблицы символов; представляет из себя хэш, состоящий из пар «имя объекта + объект».

   Методы:

   - getValue — возвращает значение переменной;
   - setValue — устанавливает значение переменной;
   - getSymbol — возвращает весь объект по его имени;
   - setSymbol — устанавливает объект по его имени;
   - allSymbols — возвращает все объекты таблицы в виде списка.

7. **ResExpr** — базовый класс выражений.

   Методы:

- copy — ничего не делает;
- isScalar — возвращает 0;
- isParam — возвращает 0;
- isFunc — возвращает 0;
- argsCount — возвращает -1;
- stringify — возвращает пустую строку;
- evaluate — возвращает неопределенное значение;
- st — сокращенная форма от stringify, вызывает stringify;
- ev — сокращенная форма от evaluate, вызывает evaluate;
- equal — сравнение; происходит путем установления факта равенства двух строковых форм, полученных при помощи stringify; участвует в перегрузке операции eq;
- notequal — сравнение; происходит путем установления факта неравенства двух строковых форм, полученных при помощи stringify; участвует в перегрузке операции ne;
- pathsByFunc — возвращает пустой список — терминальное значение;
- modifyByScalar — возвращает новое выражение, содержащее исходное в виде операнда некоторой функции, другим операндом которой является скаляр: константа или переменная. Функция и скаляр передаются методу в качестве параметров;
- addScalar — вызывает modifyByScalar, в качестве функции выступает операция сложения;
- subScalar — вызывает modifyByScalar, в качестве функции выступает операция вычитания;
- addOne — вызывает addScalar, в качестве скаляра выступает константа 1;

- transformAny — применяет преобразование к случайному вхождению искомого поддерева в выражение. Входной параметр: объект трансформации.
- transformAll — применяет преобразование ко всем вхождениям искомого поддерева. Входной параметр: объект трансформаци.
- applyTransformationSet — заданное число раз выбирает преобразование из списка и применяет его к выражению (transformAll). Входные параметры: список объектов трансформаций, количество применений.

8. **ResParam** — класс параметров, унаследован от **ResExpr**; используется при описании объектов трансформаций, где обозначает вхождение произвольного поддерева.

   Свойства:

   - *value* — значение. В качестве значения выступает имя параметра.

   Методы:

   - copy — перегруженный конструктор копирования;
   - isParam — перегруженный метод, возвращает 1;
   - stringify — перегруженный метод перевода в строку, возвращает имя параметра;
   - evaluate — перегруженный метод вычисления значения, возвращает имя параметра.

9. **ResConst** — класс констант, унаследован от **ResExpr**; выступает в качестве скаляра — листа дерева выражения.

   Свойства:

   - *value* — значение константы.

   Методы:

- copy — перегруженный конструктор копирования;
- isScalar — перегруженный метод, возвращает 1;
- stringify — перегруженный метод перевода в строку, возвращает значение константы;
- evaluate — перегруженный метод вычисления значения, возвращает значение константы.

10. **ResVar** — класс переменных, унаследован от **ResExpr**; выступает в качестве скаляра — листа дерева выражения.

    Свойства:

    - *value* — объект типа **SymVar**.

    Методы:

    - copy — перегруженный конструктор копирования;
    - isScalar — перегруженный метод, возвращает 1;
    - stringify — перегруженный метод перевода в строку, возвращает имя переменной;
    - evaluate — перегруженный метод вычисления значения, возвращает значение переменной.

11. **ResFunc** — класс вызова функции, унаследован от **ResExpr**, представляет собой внутренний узел дерева выражения.

    Свойства:

    - *func* — объект типа **SymFunc**;
    - *args* — список аргументов, представляющих собой объекты типа **ResExpr**.

    Методы:

    - copy — перегруженный конструктор копирования;

- isFunc — перегруженный метод, возвращает 1;
- argsCount — перегруженный метод, возвращает количество аргументов;
- stringify — перегруженный метод перевода в строку;
- evaluate — перегруженный метод вычисления значения;
- pathByFunc — перегруженный метод, возвращает список путей до данной функции, где путь — набор вершин от корня до данной функции;
- genBadFunc — возвращает копию исходного объекта, в которой одно случайным образом выбранное вхождение заданной функции заменено на другую функцию. Входные параметры: исходная функция, функция для замены.
- genBadAll — возвращает копию исходного объекта, в которой все вхождения заданной функции заменены на другую функцию. Входные параметры: исходная функция, функция для замены.

12. **Column** — класс арифметических выражений, вычисляемых в столбик.

    Свойства:

    - *result* — объект типа **ResExpr**, внутреннее представление выражения в столбик, построенное по цифрам.
    - *oper* — арифметическая операция, на данный момент поддерживается только сложение.
    - *operands* — список операндов, представленных массивами цифр.

    Методы:

    - build — производит построение внутреннего представления (*result*) на основе операции и операндов путем развертки их в дерево, включающее в себя сложение цифр и учет значений переноса.

- digits — возвращает список цифр, полученный из строкового представления операнда. Входной параметр: индекс операнда.
- operandsCount — возвращает длину списка операндов;
- operandLength — возвращает количество цифр в операнде;
- maxLength — возвращает наибольшее количество цифр в операнде по всем операндам;
- stringify — записывает выражение в столбик;
- evaluate — вычисляет результат на основе сложения цифр операндов;
- st — сокращенная форма stringify, вызывает метод stringify.
- ev — сокращенная форма evaluate, вызывает метод evaluate.

13. **Transformation** — класс преобразований, осуществляет замену одного поддерева внутри объекта **ResExpr** другим.

    Свойства:

    - *source* — образец исходного поддерева;
    - *target* — образец результирующего поддерева.

    Методы:

    - parseList — производит разбор строки, являющейся префиксной записью поддерева, и построение самого поддерева как объекта класса **ResExpr**. Входные параметры: список параметров преобразования, список функций и строка, которая может содержать названия параметров, имена функций и константы.
    - parse — вызывает parseList для строк, описывающих исходное и результирующее поддеревья;
    - match — осуществляет сравнение поддерева с образцом, заданным в преобразовании;

- paths — возвращает список всех путей в дереве до поддеревьев, удовлетворяющих образцу;
- substitute — заменяет одно поддерево другим;
- transform — осуществляет поиск и замену поддерева по заданному пути.

14. **ResStatement** — базовый класс операторов.

    Методы:

    - stringify — возвращает пустую строку;
    - evaluate — возвращает неопределенное значение;
    - st — вызывает stringify;
    - ev — вызывает evaluate.

15. **ResAssignment** — класс присваивания, унаследован от **ResStatement**.

    Свойства:

    - *lvalue* — переменная, объект класса **SymVar**;
    - *rvalue* — выражение, объект класса **ResExpr**;

    Методы:

    - stringify — перегруженный метод, выводит присваивание в виде строки кода;
    - evaluate — перегруженный метод, осуществляет установку значения переменной *lvalue* в таблице символов, объекта класса **SymbolTable**.

16. **ResBlock** — класс блока операторов, унаследован от **ResStatement**.

    Свойства:

- *list* — список операторов — объектов класса **ResStatement**.

Методы:

- Push — помещает оператор в список;
- stringify — перегруженный метод, в цикле осуществляет вывод операторов списка;
- evaluate — перегруженный метод, в цикле осуществляет оценивание операторов списка.

17. **ResIf** — класс ветвления, унаследован от **ResStatement**..

Свойства:

- *condition* — условие, объект класса **ResExpr**.
- *then_block* — блок, выполняющийся в случае истинности условия; объект типа **ResBlock**.
- *else_block* — блок, выполняющийся в случае неистинности условия; объект типа **ResBlock**.

Методы:

- stringify — перегруженный метод, переводит в строковый вид оператор ветвления;
- evaluate — перегруженный метод, осуществляет вычисление условия и выполнение одного из блоков в зависимости от полученного значения.

18. **Generator**

Методы:

- vars — возвращает список всех переменных из таблицы символов;
- funcs — возвращает список всех функций из таблицы символов;
- trans — возвращает список всех преобразований из таблицы символов;

- makeSymbol — создает объект заданного класса и устанавливает те его свойства, которые были переданы в качестве входных параметров.
- makeVar — вызывает makeSymbol для класса **SymVar**, записывает переменную в таблицу символов.
- makeFunc — вызывает makeSymbol для класса **SymFunc**, записывает функцию в таблицу символов.
- makeUnOp — вызывает makeSymbol для класса **SymUnOp**, записывает операцию в таблицу символов.
- makeBinOp — вызывает makeSymbol для класса **SymBinOp**, записывает операцию в таблицу символов.
- makeTransformation — создает объект класса **Transformation**, производит разбор строк, переданных в качестве входных параметров, для получения исходного и результирующего поддеревьев. Входные параметры: префиксная запись исходного поддерева, префиксная запись результирующего поддерева, список параметров, список функций.
- parseConsts — производит формирование списка диапазонов констант в зависимости от того, какое поле присутствует во входных параметрах: *consts* или *range*.
- genConst — оболочка для генерации констант, вызывает parseConsts и genConstant с полученным списком диапазонов.
- genVar — оболочка для генерации переменных, вызывает genVariable.
- genExpr — оболочка для генерации выражений, осуществляет разбор входных параметров, которые могут включать в себя: диапазон констант, список диапазонов констант, список переменных, список функций, список поддеревьев, вероятность генерации константы, вероятность генерации скаляра и вероятность генерации аргумента. Вызывает genExpression.

- genColumnExpr — оболочка для генерации выражений в столбик, вызывает genColumnExpression;
- genConstant — возвращает объект класса **ResConst**, значение которого сгенерировано случайным образом из заданного диапазона;
- genVariable — возвращает объект класса **ResVar**, значение которого сгенерировано случайным образом из заданного списка переменных класса **SymVar**;
- genScalar — если задан диапазон констант и список переменных, то с заданной или установленной по умолчанию вероятностью генерирует объекты классов **ResConst** и **ResVar**. Иначе — генерирует тот объект, для которого указан список генерации.
- genArgument — с заданной или установленной по умолчанию вероятностью генерирует скаляр или объект из списка поддеревьев. В случае когда не определен список поддерьев или списки переменных и констант, генерирует тот объект, для которого список указан.
- genFunctionName — из заданного списка функций выбирает те, число параметров которых меньше максимально возможной для подвыражения длины. Из полученного списка случайным образом выбирает функцию и возвращает ее.
- genExpression — с заданной вероятностью или если нет подходящих по длине функций генерирует значение из списка поддеревьев, переменных или констант. Иначе вызывает genFunctionName. По числу параметров выбранной функции делит максимально возможную длину и с этой новой длиной вызывает себя же для генерации аргументов функции.
- genColumnExpression — возвращает объект класса **Column**, полученный путем генерации последовательностей цифр каждого операнда до указанной длины и построению по

этим последовательностям и указанной операции выражения *result*.

- genAssignment — возвращает объект класса **ResAssignment** с установленными по входным параметрам свойствами *lvalue* и *rvalue*.
- genLOBlock — возвращает объект класса **ResBlock**; использует те же входные параметры, что и метод генерации выражения genExpr. Генерирует последовательность присваиваний в порядке перечисления переменных.
- genIf — возвращает объект класса **ResIf**.

19. **StringUtil**

Методы:

- Length — возвращает длину указанной строки;
- numberOf — возвращает количество вхождений символа в строку;
- numberOfSpaces — возвращает количество пробелов в строке;
- lengthWithout — возвращает длину строки без заданного символа;
- lengthWithoutSpaces — возвращает длину строки без пробелов;
- Case — в зависимости от последней цифры заданного числа и заданных окончаний возвращает указанное слово в нужном падеже; входные параметры: число, слово в именительном падеже единственного числа, окончание именительного падежа множественного числа, окончание родительного падежа множественного числа;
- RegCase — возращает слово в нужном падеже для окончаний «-а» и «-ов».
- BitCase — возращает нужный падеж слова «бит».
- BiteCase — возращает нужный падеж слова «байт».

20. **NumberUtil**

    Методы:

    - numberOfDigit — возвращает количество вхождений цифры в число;
    - toDecimal — переводит число в десятичную систему счисления; входные параметры: число и его исходная система счисления;
    - toNotation — переводит десятичное число в заданную систему счисления;
    - toHexadecimal — переводит десятичное число в шестнадцатиричную систему счисления;
    - toOctal — переводит десятичное число в восьмеричную систему счисления;
    - toBinary — переводит десятичное число в двоичную систему счисления;
    - inverseBinary — возвращает инвертированное двоичное число;
    - binaryLength — возвращает длину двоичного числа;
    - Exp — возвращает степень числа; входные параметры: основание степени, показатель степени;
    - Exp2 — возращает степень числа 2 (степень двойки).

5.2.2 Модули

1. *auxiliary.pm*

    - setFields — устанавливает значения свойств объекта в зависимости от их наличия в ссылке входных параметров;
    - setRef — устанавливает поля ссылки по ссылке входных параметров;
    - copyRef — возвращает копию исходной ссылки;

- isNoOf — проверяет отсутствие в ссылке заданного поля; входные параметры: имя поля, ссылка;
- defaultIfUndefined — если определено указанное значение, то возвращает его, иначе — значение по умолчанию; входные параметры: исходное значение, значение по умолчанию.

2. *auxrandom.pm*

   - frand — возвращает случайное вещественное число от 0 до 1.
   - irand — возвращает случайное целое число от 0 до заданного, не включая его.
   - randelem — возвращает случайный элемент списка.

3. *auxtemplate.pm*

   - templit — производит создание объекта шаблона Template::Toolkit с описанной внутри функции конфигурацией и его средствами осуществляет обработку входного текстового файла.

4. *column.pm*

   - описание класса **Column**.

5. *generator.pm*

   - описание класса **Generator**.

6. *maker.pm*

   - makeGenerator — вызывает функцию makeSymbolTable, создает объект класса **Generator** и возвращает его;
   - makeNumberUtil — создает объект класса **NumberUtil**;
   - makeStringUtil — создает объект класса **StringUtil**;

- makeSettings — создает ссылку с заданными полями и значениями, которая затем может быть использована в качестве входного параметра для функций генерации.

7. *numberutil.pm*
   - описание класса **NumberUtil**.
8. *result.pm*
   - описание классов **ResExpr**, **ResParam**, **ResConst**, **ResVar**, **ResFunc**.
9. *statement.pm*
   - описание классов **ResStatement**, **ResAssignment**, **ResBlock**, **ResIf**.
10. *stringutil.pm*
    - описание класса **StringUtil**.
11. *symbols.pm*
    - описание класса **Symbol**, **SymVar**, **SymFunc**, **SymUnOp**, **SymBinOp**.
12. *symboltable.pm*
    - описание класса **SymbolTable**.
13. *symboltableutil.pm*
    - makeSymbolTable — создает три глобальные таблицы символов: для переменных, функций и преобразований;
    - getValue — возвращает значение переменной из таблице символов;

- setValue — устанавливает значение переменной в таблице символов;
- getFunc — возвращает функцию по ее имени в таблицы символов;
- setFunc — устанавливает функцию по ее имени в таблице символов;
- getTransformation — возвращает преобразование по его имени в таблице символов;
- setTransformation — устанавливает преобразование по его имени в таблице символов;
- getAllVars — возвращает все переменные из таблицы символов;
- setAllFuncs — возвращает все функции из таблицы символов;
- getAllTransformations — возвращает все преобразования из таблицы символов.

14. *transform.pm*

    - описание класса **Transformation**.

5.2.3 Алгоритмы

Алгоритм генерации выражения

Входные параметры:

- список констант (**consts**);
- список переменных (**vars**);
- список поддеревьев (**subexprs**);
- список функций (**funcs**);
- максимальная длина (**maxlen**);
- вероятность константы (**const_pr**);

- вероятность скаляра (константы или переменной) (**scalar_pr**);
- вероятность аргумента (**arg_pr**).

Все входные параметры являются необязательными, кроме максимальной длины выражения. Однако алгоритм не будет работать, если не будет указан хотя бы один из списков констант, переменных и поддеревьев.

1. С помощью генератора случайных чисел и вероятности сгенерировать аргумент функции определить, что будет сгенерировано: функция или ее аргумент.

   1.1. Если будет сгенерирован аргумент, то если заданы все три списка: поддеревьев, переменных и констант, по вероятности сгенерировать скаляр опередделить, что будет сгенерировано: поддерево или скаляр.

      1.1.1. Если поддерево, то выбрать случайное поддерево из списка и вернуть его.

      1.1.2. Если скаляр, то по вероятности сгенерировать константу определить, что будет сгенерировано: переменная или константа.

         1.1.2.1. Если переменная, то выбрать случайную переменную из списка переменных.

         1.1.2.2. Если константа, то выбрать случайную константу из диапазона констант.

      Если какой-то из списков аргументов не задан, то соответвующие вероятности не используются при генерации, и алгоритм идет по нужной ветке автоматически.

   1.2. Если будет сгенерирована функция, то выбрать из списка функций те, у которых количество параметров не превышает максимальной длины выражения. Если таких нет, то

сгенерировать аргумент, иначе — случайным образом выбрать из сформированного списка ту функцию, которая будет использоваться.

2. Разделить максимальную длину на количество параметров этой функции (если один параметр или нет параметров, то вычесть из максимальной длины единицу). С полученной длиной, меньшей максимальной, вызвать этот же алгоритм для генерации аргументов.

Алгоритм преобразования дерева выражения

Преобразование является классом, содержащим два поля: образец исходного и образец результирующего поддеревьев. В качестве листьев образца дерева, кроме констант и переменных, могут выступать параметры, которые обозначают место вхождения произвольных поддеревьев.

Входные параметры: дерево выражения и путь к поддереву, которое нужно заменить.

Определение списка всех путей, соответствующих виду исходного поддерева в преобразовании, осуществляется отдельным алгоритмом, который идет от корня и сравнивает каждый узел с образцом. Если узел удовлетворяет сравнению, то алгоритм добавляет пустой список в список путей. Затем он продолжает поиск среди аргументов данного узла. К путям, полученных от аргументов, алгоритм добавляет порядковые номера аргументов и помещает их в общий список путей (который может содержать уже пустой список) и возвращает полученный общий список. Любой путь из этого списка может использоваться в качестве параметра для алгоритма преобразования.

Использующийся в поиске путей и преобразовании поддерева алгоритм сравнения поддерева с образцом на вход получает поддерево, образец поддерева и таблицу соответствий, являющуюся ассоциативным массивом, в котором в качестве ключей выступают имена параметров,

а в качестве значений — закрепленные за ними поддеревья. Алгоритм производит следующую проверку:

1. Если образец является параметром и не содержит значения в таблице соответствий, то он помещает текущее поддерево в таблицу соответствий и возвращает истину. Если же он содержит значение, то результат алгоритма зависит от того, равно ли текущее поддерево этому значению. Факт равенства устанавливается при помощи сравнения строковых форм соответствующих поддеревьев.

2. Если образец является константой, то алгоритм возвращает истину только в том случае, когда поддерево также является константой и значения констант образца и поддерева равны.

3. Если образец является функцией, то алгоритм возвращает ложь, если поддерево не является функцией или является функцией, отличной от функции образца, или количество аргументов в этих функциях не совпадает.

4. Если образец является коммутативной операцией, то если оба аргумента поддерева в любом порядке удовлетворяют аргументам образца, то алгоритм возвращает истину, иначе — ложь.

5. Если образец является произвольной функцией, то алгоритм возвращает истину только в том случае, когда все аргументы поддерева соответствуют всем аргументам образца.

Далее, для преобразования выражения необходимо рекурсивно спуститься по указанному пути до нужного поддерева, заменить его на копию результирующего поддерева из объекта преобразований и подставить вместо параметров поддеревья из таблицы соответствий. При этом, если выражение успело измениться и по заданному пути нужного поддерева не оказалось, алгоритм завершает работу.

Алгоритм генерации линейного упорядоченного блока

Алгоритму передаются все те же параметры, что и для алгоритма генерации выражения. Единственное отличие в том, что порядок переменных в списке имеет значение, поскольку именно в этом порядке будут генерироваться операторы присваивания соответствующих переменных. Полученный в результате блок называется линейным упорядоченным, поскольку содержит только линейные операторы присваивания, упорядоченные в порядке следования переменных.

Генерация блока происходит путем последовательной генерации присваиваний переменным в переданном списке. Для этого создается список переменных, изменяющийся от присваивания к присваиванию. При генерации первого присваивания он является пустым, при генерации второго — содержит первую переменную, при генерации третьего присваивания — первые две переменные и т.д.

Генерация присваивания состоит из сохранения переменной в *l-value* и генерации выражения и сохранении его в *r-value*.

Генерация выражений при создании блока происходит с одними и теми же параметрами, указанными при вызове алгоритма. Однако внутри алгоритма изменяются значения параметров настроек: вероятностей генерации аргумента, скаляра и константы. При генерации первого присваивания они все равны единице: таким образом, правая часть первого присваивания неизбежно содержит константу. При последующих присваиваниях эти вероятности начинают убывать по экспоненциальному закону и выражения в них содержат уже больше переменных, чем констант.

6 Реализация и тестирование

Текущая версия системы обладает следующими характеристиками:

- объем кода на языке Perl: 2000 строк (32 КБайт);
- объем кода на языке Template::Toolkit: 780 строк (14 Кбайт);
- число модулей: 15;
- число тестов: 20.

Тестирование производилось вручную по методу белого ящика.

Ниже приведены основные этапы реализации системы, примеры тестов и результаты тестирования.

Базовая функциональность

Вначале необходимо было создать элементы, которые бы легли в основу генерации. Такими базовыми элементами стали: диапазоны констант, переменные, функции, операции и таблица символов.

Первоначально планировалось создавать константы на основе перечислений, диапазонов, ограничений и последовательностей. Средствами Template::Toolkit оказалось возможным объединить перечисления и диапазоны, однако ограничения и последовательности остались нереализованными.

Из переменных должны были поддерживаться скалярные переменные и массивы, но массивы остались нереализованными. Для переменных нужно было указывать два поля: имя и тип. Со временем к этим полям было добавлено третье — значение, поскольку изначально в выражениях участвовали не сами переменные, а их имена. Значение же и прочие детали планировалось брать из таблицы символов, хранящейся в генераторе. Однако, чтобы не возникало циклической зависимости между модулями: генератора, создающего дерево выражения, и выражения,

использующего таблицу из генератора для вычисления значения, было принято решение хранить в дереве весь объект переменной вместе со значением.

Когда работа коснулась генерации операторов и, в частности, присваивания, появилась необходимость вернуться к таблице символов, которая бы хранила эталон переменной. Такая таблица была реализована отдельно, независимо от обоих модулей: генератора и выражения.

Для функции решено было указывать ее стандартное описание: имя, список типов параметров, тип результата и тело. Оно также должно было храниться в таблице символов с использованием в качестве ключа прототипа, построенного на основе описания. Так можно было бы добиться использования перегруженных функций с различными списками параметров. Тогда доступ к функции осуществлялся бы по ее имени и списку типов либо по прототипу. Но, чтобы не заострять внимания на деталях, было решено использовать только одну функцию с одним именем.

Описание тела функции после попыток выполнить его в чистом Template::Toolkit выполняется в блоке кода Perl, заключенного между тегами PERL и END, при помощи вызова метода set специальной переменной *$stash* библиотеки Template::Toolkit.

Оказалось удобным создавать операции отдельно от функций. Такое решение позволило описывать для операций дополнительные поля, такие, как коммутативность и приоритет, и по умолчанию устанавливать количество аргументов, а для стандартных операций — не указывать тело функции.

Ниже на иллюстрациях показаны примеры создания переменных и операций с помощью методов генератора: makeVar, makeUnOp, makeBinOp. Не отраженным на примере остался метод makeFunc и функция makeSettings, предназначенная для создания списка параметров в виде ссылки. Изначально перечисленные методы были реализованы вне генератора, но позднее были в него перенесены.

Генерация выражений

Реализация генерации выражений происходила от простого к сложному. Сперва была осуществлена генерация констант, затем — переменных и в конце — выражений, содержащих функции. Использовавшийся для этого синтаксис первоначально был реализован в виде функций, затем был осуществлен переход на объектно-ориентированную модель, и функции генерации были преобразованы в методы объекта генератора.

При генерации константы параметром является диапазон констант, при генерации переменной — список переменных. В общем случае — при генерации выражения — эти списки комбинируются и дополняются списком функций, а также — максимальной длиной выражения, которая представляет из себя количество листьев дерева выражения.

В выражениях, содержащих переменные и константы, для того, чтобы константы выглядели свернутыми, т.е. чтобы аргументы функций или операций не состояли исключительно из констант, был введен параметр вероятности константы (**const_pr**), по умолчанию равный 0.2. Эта мера позволила сократить число нежелательных выражений, но не исключить их полностью.

Первоначально реализованный алгоритм генерировал вызовы функций до тех пор, пока позволяла длина, только после этого он переходил к генерации скаляров, в результате чего все сгенерированные выражения были одинаковой длины и имели схожую структуру. Для того, чтобы их разнообразить, был введен параметр вероятности скаляра (**scalar_pr**), равный по умолчанию 0.2. В результате этого на каждом витке генерации стало приниматься решение, будет ли данное подвыражение скаляром (переменной или константой) или функцией. Побочным эффектом такой реализации явилось то, что скаляр может быть сгенерирован уже на первом шаге генерации, что обычно не соответствует ее целям.

По мере тестирования для описаний, в частности, различных тригонометрических выражений, появилась необходимость в поэтапной генерации выражений, позволяющей использовать для генерации нового этапа те выражения, которые были получены на предыдущем этапе. Для этого в список параметров генерации был включен параметр списка аргументов — уже готовых деревьев выражений. Соответственно, в качестве вероятности генерации скаляра стала использоваться вероятность генерации аргумента (**arg_pr**), по умолчанию равная 0.2, а вероятность генерации скаляра (**scalar_pr**) была опущена на уровень ниже и стала использоваться для выбора между генерацией поддерева и генерацией скаляра.

Сам результат генерации — дерево выражения — изначально представлял из себя один класс, использовавшийся для всех типов выражений: констант, переменных и функций. Позднее была создана иерархическая система классов, которая разделила эти типы, а также отделила от них появившийся с реализацией класса преобразований тип выражения-параметра.

На рисунке приведен пример генерации выражения с использованием констант, переменных и функций, максимальной длиной, составляющей 3 операнда, и вероятностью генерации константы, равной 0.4.

```
[%-
  g = makeGenerator;

  v1 = g.genConst(range => [10..99]);
  v2 = g.genConst(range => [10..99]);

  x = g.makeVar(name => 'x', value => v1.ev);
  y = g.makeVar(name => 'y', value => v2.ev);

  f = g.makeBinOp(name => '+', commutative => 1);

  e = g.genExpr(range => [10..99], vars => [x, y],
    funcs => [f], maxlen => 3, const_pr => 0.4);

-%]
Пусть x = [% x.value %], y = [% y.value %].
Чему равно значение выражения?

[%- e.st -%];

Ответ: [% e.ev %].
```

Результат:

Пусть x = 95, y = 75.
Чему равно значение выражения?
x + 37;
Ответ: 132.

Пусть x = 41, y = 94.
Чему равно значение выражения?
y;
Ответ: 94.

Пусть x = 49, y = 96.
Чему равно значение выражения?
y + 48;
Ответ: 144.

Пусть x = 41, y = 75.
Чему равно значение выражения?
x + y;
Ответ: 116.

Рис. 2: Генерация выражений

Генерация сложения в столбик

Примеры на вычисление в столбик должны были стать одной из наиболее простых областей применения системы. Однако для того, чтобы они могли поддерживать преобразования, влияющие на алгоритм вычислений, встала необходимость реализации их как отдельного класса и построения в этом классе дерева вычислений, пример которого, переведенный в строковую форму, можно видеть на рисунке.

Преобразование выражений

Простейшие преобразования выражений, такие, как добавление к выражению скаляра или изменение функции на другую, были реализованы как методы класса выражений. По мере движения к более сложным преобразованиям был реализован отдельный класс, осуществляющий преобразования поддеревьев. Однако для сохранения

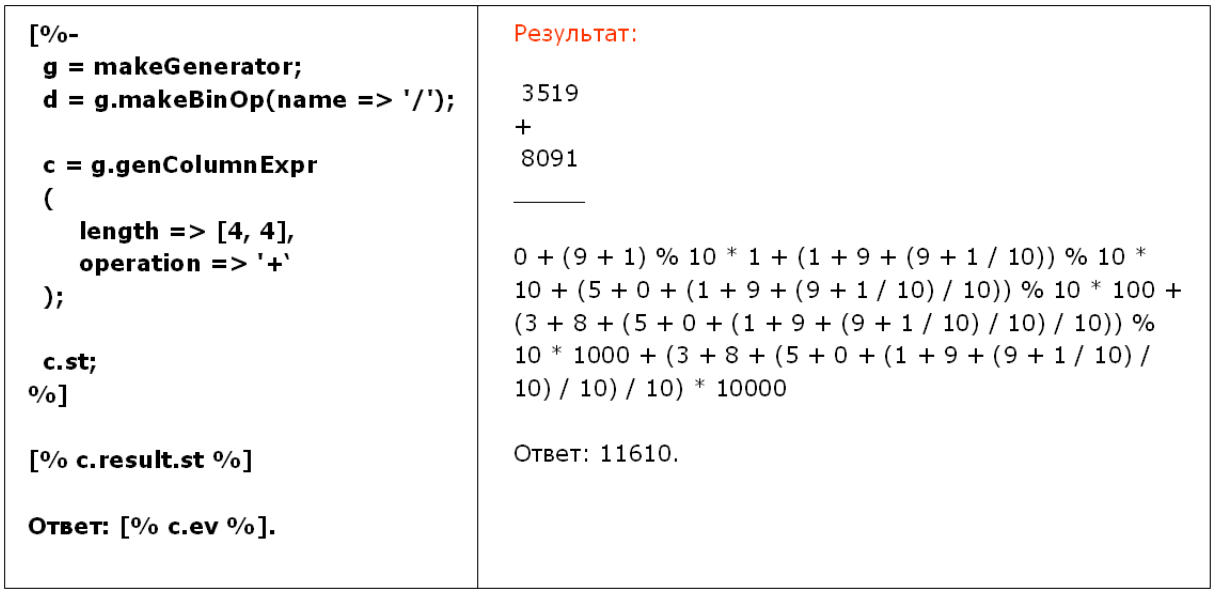

Рис. 3: Генерация сложения в столбик

единообразия вызов преобразования с использованием объектов этого класса происходит из объекта выражения.

Объекты преобразований создаются при помощи метода генератора makeTransformation и сохранются в таблице символов. При их описании используется префиксная запись образцов исходного и результирующего поддеревьев, список имен параметров и список функций. Для того, чтобы не дублировать объявления, преобразования по коммутативным операциям реализованы автоматически вне зависимости от порядка следования операндов в выражении.

Подобные преобразования осуществляют только один проход по выражению. Для того, чтобы изменить выражение несколько раз одним или несколькими преобразованиями, был реализован соответствующий метод, выбирающий случайным образом одно преобразование из указанного списка, применяющий его к выражению и повторяющий эту процедуру указанное число раз.

В результате при достаточно большом числе применений происходит существенное изменение выражения, но в зависимости от списка

преобразований оно может быть реализовано как в сторону его упрощения, так и в сторону усложнения. Для контроля такого поведения удобно использовать оптимизационный алгоритм, однако его реализация не вошла в данную работу.

На рисунке приведен пример генерации дистрактора для логического выражения, реализованный путем применения к выражению правильного и неправильного преобразований на основании правила де Моргана.

```
[%-
  g = makeGenerator();
  neg = makeUnOp(name => '!');
  con = makeBinOp(name => `&&`, commutative => 1);
  dis = makeBinOp(name => `||`, commutative => 1);

  t1 = makeTransformation(
    source => `&& ! a ! b', target => `! || a b',
    funcs  => g.funcs);
  t2 = makeTransformation(
    source => '&& ! a ! b', target => '! && a b',
    funcs  => g.funcs);

  a = makeVar(name => 'a', value => 1);
  b = makeVar(name => `b', value => 1);
  e = g.genExpr(funcs => [neg, con],
    vars => [a, b], maxlen => 10);

  x = e.transformAny(t1);
  y = e.transformAny(t2);
-%]
Исходное:         [% e.st %]
Эквивалентное:    [% x.st %]
Неэквивалентное:  [% y.st %]
```

```
Результат:
Исходное:       !(a) && !(a) && !(!(a))
Эквивалентное:   !(a || a) && !(!(a))
Неэквивалентное: !(a && a) && !(!(a))

Исходное:       !(!(b)) && !(!(a && b))
Эквивалентное:   !(!(b) || !(a && b))
Неэквивалентное: !(!(b) && !(a && b))

Исходное:       b && b && !(a) && !(b)
Эквивалентное:   b && b && !(a || b)
Неэквивалентное: b && b && !(a && b)
```

Рис. 4: Генерация дистрактора

Генерация операторов

Следующим этапом после генерации выражений стала генерация операторов и, прежде всего, — оператора присваивания. Для этого в употребление была введена таблица символов, хранящая значения переменных.

После реализации присваивания стала возможна генерация блока присваиваний. На данный момент блок генерируется в порядке

следования переменных. На рисунке приведен пример генерации такого блока.

Кроме того, возможна реализация генерации линейных блоков с произвольным порядком переменных, реализация ветвлений, циклов и произвольных блоков, однако на данный момент она не осуществлена.

```
Чему равно значение z?
[%-
  g = makeGenerator;

  x = g.makeVar(name => 'x');
  y = g.makeVar(name => 'y');
  z = g.makeVar(name => 'z');

  sum = g.makeBinOp(name => '+',
    commutative => 1, priority => 1);
  sub = g.makeBinOp(name => '-',
    commutative => 0, priority => 1);
  mul = g.makeBinOp(name => '*',
    commutative => 1, priority => 2);

  b = g.genLOBlock(vars => g.vars, range => [1..15],
    funcs => g.funcs, maxlen => 10);
  b.ev;
-%]
[% b.st %]
Ответ: [% z.value %].
```

Результат:

Чему равно значение z?
x = 2;
y = 12;
z = y - x - (4 + 10);
Ответ: -4.

Чему равно значение z?
x = 7;
y = x + x - 3;
z = y;
Ответ: 11.

Чему равно значение z?
x = 12;
y = x;
z = (x - 9) * (x - y);
Ответ: 0.

Рис. 5: Генерация блока

Проверка работы системы на заданиях ЕГЭ по информатике

Причины

Решение проверить работу системы на тесте из ЕГЭ по информатике было принято по следующим причинам:

1. Эта область является актуальной для применения, поскольку варианты тестов ЕГЭ по информатике до сих пор составляются вручную (примеры тестов можно найти в [13], [16], [26], [33], [34]).

2. Условия и вопросы заданий по-разному сформулированы, в связи с чем можно проверить, насколько система позволяет их вопроизвести внутри шаблона.

3. Набор заданий из теста по информатике захватывает различные области знаний [21], такие, как:

   - системы счисления;
   - дискретная математика;
   - программирование;
   - кодирование информации;
   - электронные таблицы и т.п.

   Вследствие этого интересно выделить специфические и общие возможности системы, которые требуются при составлении шаблона для каждой из этих областей.

4. Задания единого государственного экзамена составлены так, чтобы наиболее адекватно и полно проверить знания учащихся по данной дисциплине. В связи с этим, если метод, реализованный в системе, позволяет генерировать задания ЕГЭ по информатике, то есть вероятность того, что он также может быть применен и для генерации ЕГЭ по другим дисциплинам.

На рисунке приведен пример генерации одного из простых заданий ЕГЭ с использованием справочника констант.

```
A15 [%
  g = makeGenerator;
  colors =
  {
    'Красный'    => "FF0000"
    'Желтый'     => "FFFF00"
    'Синий'      => "0000FF"
    'Зеленый'    => "00FF00"
    'Белый'      => "FFFFFF"
    'Черный'     => "000000"
    'Фиолетовый' => "FF00FF"
  };
  color = g.genConst(range => colors.keys).ev;
  code = colors.item(color);
  colors.delete(color);
%]
Какой цвет будет у страницы, заданной тегом
    <body bgcolor=[% code %]>?
[%- FOREACH i IN [1..3] -%]
[%i%])[% c = g.genConst(range => colors.keys).ev;
    colors.delete(c); c -%];
[%- END -%]
4)[%- color # правильный ответ -%]
```

Результат:

A15
Какой цвет будет у страницы, заданной тегом <body bgcolor=FFFFFF>?

1) Желтый; 2) Красный;
3) Черный; 4) Белый

A15
Какой цвет будет у страницы, заданной тегом <body bgcolor=0000FF>?

1) Фиолетовый; 2) Зеленый;
3) Красный; 4) Синий

A15
Какой цвет будет у страницы, заданной тегом <body bgcolor=FFFF00>?

1) Черный; 2) Синий;
3) Фиолетовый; 4) Желтый

Рис. 6: Генерация задания из ЕГЭ по информатике

Для каждого типа задания из 10 демонстрационных вариантов ЕГЭ по информатике за 2010 год [33] был проведен анализ сложности его описания средствами данной системы. Результат анализа приведен ниже.

Описание заданий, шаблоны и дистракторы

A1 Влияние выбора кодировки (однобайтной или двухбайтной) на информационный объем сообщения (т.е. на количество байт в нем)

Шаблоны условия:

1. Вычислить информационный объем строки, если строка записана в указанной кодировке.

2. Вычислить изначальный информационный объем строки, если при изменении кодировки он изменился на указанное число байт.

Дистракторы:

- длина строки без пробелов;
- неверное число бит в байте;
- замена понятия бита байтом и наоборот;
- наличие исходных значений в ответе.

A2 Минимальное количество бит, необходимое для того, чтобы закодировать некоторое конечное множество значений

Примеры условия (всего — 6 шаблонов):

1. В некоторой базе данных хранятся записи, содержащие информацию о некоторых датах. Каждая запись содержит 3 поля: год (от 1 до 2100), месяц (от 1 до 12) и день (от 1 до 31). Каждое поле записывается отдельно от других полей с использованием минимально возможного количества бит. Каково минимальное количество бит, необходимое для кодирования N записей.
2. Сколько бит содержит N килобайт?

Дистракторы:

- замена понятия бита байтом и наоборот;
- наличие исходных значений в ответе;
- цифра или символ закодированы с помощью 8 бит;
- цифра или символ закодированы с помощью 1 бита;
- степени 10 для кило-, мега- и гигабайт вместо степеней 2.

A3 Системы счисления

Шаблоны условия:

1. Даны 2 числа в различных системах счисления. Определить, какое число лежит в помежутке между ними.
2. Определить количество вхождений заданной цифры (0 или 1) в двоичную запись некоторого числа.
3. Перевести число из одной системы счисления в другую.

Дистракторы:

- правильный ответ с неправильной системой счисления;
- инвертирование для двоичных чисел;
- неверные буквенные обозначения для шестнадцатиричных чисел;
- случайные числа.

A4 Сложение чисел в различных системах счисления

Шаблон условия:

1. Чему равна сумма двух чисел, заданных в различных системах счисления.

Дистракторы аналогичны описанным в предыдущем задании.

A5 Блоки кода, состоящие из последовательности операторов присваивания

Шаблон условия:

1. Определить значение переменной после выполнения фрагмента программы.

Шаблон фрагмента программы:

1. Последовательность присваиваний вещественным переменным выражений, содержащих только арифметические операции.

2. Последовательность присваиваний целочисленным переменным выражений, содержащих арифметические операции и операции DIV и MOD.

Дистракторы:

- значение другой переменной;
- использование исходного значения переменной, несмотря на последующие ей присваивания;
- использование MOD вместо DIV и DIV вместо MOD;
- случайные изменения в выражениях;
- изменение правильного ответа на единицу.

Обобщение:

- генерация линейного упорядоченного блока;
- генерация блока заданной структуры (с описанием операций, которые должны использоваться в каждом присваивании);
- преобразования над блоками;
- перевод блока в строковую форму с использованием синтаксисов различных языков программирования.

Обобщение этого задания позволяет уменьшить объем текста шаблона, однако требует значительных усилий в реализации функциональной части.

A6 Массивы и циклы FOR

Составные части шаблона задания:

1. Исходные данные об имеющихся массивах:
   - количество массивов (1 или 2);
   - размерности массивов (одномерные или двумерные);
   - диапазон индексов.

2. Блок кода, в котором происходит инициализация и обработка исходных массивов. Он может включать в себя:
   - циклы FOR с увеличением переменной цикла;
   - циклы FOR с уменьшением переменной цикла;
   - вложенные циклы FOR;
   - ветвления;
   - изменение внешних переменных: счетчиков, индикаторов, сумм;
   - присваивания индексов;
   - присваивания элементам массива выражений, зависящих от переменной цикла и внешних переменных;
   - присваивания элементам массива выражений, содержащих элементы другого массива;
   - использование элементов массивов с индексами, сдвинутыми относительно переменной цикла.
3. Вопрос, в котором, как правило, требуется следующее:
   - объяснить, что делает приведенный блок кода;
   - определить, чему после обработки будет равно содержимое массива;
   - применить некоторую агрегирующую функцию к элементам массива, например:
     * найти наибольший или наименьший элемент;
     * посчитать количество положительных (отрицательных, нулевых, четных, нечетных) элементов;
     * посчитать их сумму (произведение, среднее арифметическое).
   - определить значение некоторой внешней переменной, которая изменялась в цикле вместе с массивом или которой было присвоено значение после цикла.

4. Дистракторы. В зависимости от условия и вопроса могут содержать:
   - значения на границах массива;
   - значения, полученные при выполнении действий, противоположных требуемым (например, вместо наибольшего значения найдено наименьшее, вместо количества четных элементов — количество нечетных и т.п.);
   - значения, вычисление которых содержит сходные операции (например, для операции «больше» ответами могут служить поиск наибольшего значения и определение числа положительных элементов; для сложения — вычисление суммы и среднего арифметического).
   - индекс элемента вместо его значения (ср. «поиск наименьшего элемента» и «поиск индекса наименьшего элемента»).

A7 Определение значения, удовлетворяющего логическому выражению

Шаблоны условия:

1. Для какого из имен (названий животных, символьных наборов) истинно (или ложно) высказывание: (приведено высказывание, содержащее любые логические операции над утверждениями типа «Всего N букв», «K-я буква — гласная», «L-я буква — согласная» и т.п.).
2. Для какого числа Y истинно высказывание: (приведено высказывание, содержащее любые логические операции над сравнениями с использованием числа Y).

Дистракторы:

- значение, для которого выполняются все условия, кроме одного (для конъюнкции);

– любое значение из справочника, для которого условие не выполняется;

– неэквивалентное преобразование условия;

– изменение утверждения в высказывании (например, «Четвертая буква — согласная» заменить на «Третья буква — согласная»).

A8 Преобразования логических выражений, содержащих операции И, ИЛИ, НЕ

Шаблон условия:

1. Какое логическое выражение равносильно данному.

Дистракторы:

– выражение, полученное при помощи неэквивалентного преобразования;

– случайно сгенерированное выражение с таблицей истинности, отличной от таблицы истинности исходного выражения.

A9 Поиск логического выражения по его таблице истинности

Шаблон задания:

1. Дан фрагмент таблицы истинности выражения F. Какое выражение из приведенных соответствует F?

Дистракторы аналогичны описанным в предыдущем задании.

Генерацию задания удобно осуществлять в следующей последовательности:

1. Сгенерировать выражение, которое будет правильным ответом.
2. Сгенерировать дистрактор к нему.
3. На основе различий в таблице истинности между правильным ответом и дистрактором выделить фрагмент талицы истинности правильного ответа, который будет указан в условии задания.

A10 Отображение таблично заданного графа, поиск кратчайшего пути

Шаблоны условия:

1. Между четырьмя местными аэропортами ежедневно выполняются авиарейсы. Приведен фргамент расписания перелетов между ними. Путешественник оказался в аэропорту N в некоторое время hh:mm. Определите самое раннее время, когда он может попасть в аэропорт M.
2. В таблице приведена стоимость перевозок между соседними железнодорожными станциями. Укажите схему, соответствующую таблице.

Дистракторы для первого шаблона:

– время прямого перелета из N в M;
– время любого другого маршрута, не являющееся самым ранним;
– случайное время;
– случайное время, являющееся самым ранним из приведенных.

Дистракторы для второго шаблона:

– схема с лишними ребрами между станциями;
– схема, в которой недостает ребер между станциями;
– схема с измененными весами;
– случайная схема для данных железнодорожных станций.

A11 Запись букв двоичным кодом, перевод сообщения, составленного из этих букв, в некоторую систему счисления

Шаблон условия:

1. Для кодирования букв А, Б, В, Г используются указанные двоичные числа. Если таким образом закодировать сообщение (сообщение состоит из букв А, Б, В, Г) и записать результат в заданной системе счисления, какое получится число.

Дистракторы:

– неправильное сообщение;

– использование систем счисления, отличных от заданной;

– неправильное соответствие между буквами и кодами;

– изменение самих кодов;

– суммирование кодов вместо их последовательной записи в сообщении.

A12 Поиск размещения из N по K элементов, удовлетворяющего заданному правилу

Шаблоны условия:

1. Цепочка из N бусин, помеченных латинскими буквами, формируется по следующему правилу: (например: в середине цепочки стоит одна из бусин A, D, E. На третьем месте — одна из бусин A, B, C, E, которой нет на втором месте. На первом месте — одна из бусин A, B, D, не стоящая на третьем месте). Какая из перечисленных цепочек создана по этому правилу?
2. M пригласил своего друга N в гости, но не сказал ему код от цифрового замка своего подъезда, а послал следующее сообщение: (например: «в последовательности 4, 1, 8, 2, 6 все числа больше 3 разделить на 2, а затем удалить из полученной последовательности все четные числа»). Выполнив указанные действия, N получил следующий код для цифрового замка.

Дистракторы для первого шаблона:

– неправильный номер позиции в правиле;

– наличие бусины, не соответствующей данной позиции.

Дистракторы для второго шаблона:

– неправильный порядок чисел в исходной последовательности;

– пропуск, добавление или изменение чисел в исходной последовательности;

– невыполнение одного из правил (например, не делить все числа больше 3 на 2);

– изменение правила (например, делить на 2 все числа больше 2).

A13 Маски файлов, создание подкаталога, перемещение по дереву каталогов

Шаблоны условия:

1. Определить, какое из указанных имен файлов удовлетворяет заданной маске (маска имени файла включает в себя квантификаторы '*' — любая, в том числе пустая, последовательность символов, '?' — некоторый символ).
2. В некотором каталоге хранится заданный файл. После того, как в этом каталоге создали новый подкаталог и переместили файл в созданный каталог, полное имя файла стало (приведено полное имя в виде: путь к файлу + имя файла). Какое было полное имя данного файла до перемещения.
3. Перемещаясь из одного каталога в другой, пользователь последовательно посетил каталоги (приведен список каталогов, включающий корневой). Каково полное имя каталога, из которого начал перемещение пользователь.

Дистракторы для первого шаблона:

– изменение положения квантификаторов;

– вставка «пустого» символа на место квантификатора '?';

– вставка последовательности символов между символами, не разделенными квантификатором.

Дистракторы для второго шаблона:

- путь к каталогу без имени файла;
- относительный путь к файлу (без корневого каталога);
- полное имя файла после перемещения файла;
- полное имя, включающее в себя случайный каталог.

Дистракторы для третьего шаблона:

- неполный путь к каталогу;
- неправильный порядок каталогов;
- полное имя каталога, в который переместился пользователь;
- полное имя, включающее в себя случайный каталог.

A14 Поиск по таблице

Шаблон условия:

1. Результаты тестирования представлены в таблице: (приведена таблица, содержащая фамилии учеников, пол, количество баллов по нескольким предметам, например: физике, математике, химии, биологии). Сколько записей в ней удовлетворяют условию: (приведено логическое выражение над значениями полей таблицы, например: «Пол = 'ж' И Физика < Биология»).

Дистрактор:

- неэквивалентное преобразование логического выражения и поиск по нему.

A15 Цветовая модель RGB

Шаблон условия:

1. Для кодирования цвета фона страницы Интернет используется атрибут bgcolor="#XXXXXX где в кавычках задаются шестнадцатиричные значения интенсивности цветовых компонент в 24-битной RGB-модели. Какой цвет будет у страницы, заданной тегом <bgcolor="#XXXXXX».

Дистракторы:

– цвет, полученный в результате изменения порядка компонент;
– любой другой цвет, сгенерированный на основе справочника.

A16 Работа с функциями электронной таблицы СУММ и СРЗНАЧ

Шаблон условия:

1. В электронной таблицы заданы три формулы с использованием функций СУММ и СРЗНАЧ — для всего заданного диапазона ячеек и двух непересекающихся его частей. Если известны значения двух формул, найти значение третьей формулы.

Дистракторы:

– замена функций СУММ и СРЗНАЧ друг другом;
– для формул, содержащих СРЗНАЧ, — изменение границ диапазона;
– наличие исходных данных в ответе.

A17 Столбиковые и круговые диаграммы

Шаблон условия:

1. На столбиковой диаграмме показано количество участников тестирования по предметам в разных регионах России. Какая из круговых диаграмм правильно отражает соотношение количества всех (либо по какому-либо отдельному региону или предмету) участников на тестировании по некоторому предмету (в некотором регионе)?

Дистракторы:

- неправильная расцветка диаграммы;
- диаграмма для региона (предмета), отличного от заданного, или для всех регионов (предметов);
- диаграмма по предмету (региону), отличному от заданного;
- случайное деление круговой диаграммы на нужное число частей.

A18 Исполнение алгоритма, поиск всех начальных данных, приводящих к нужному результату

Шаблон условия:

1. Система команд исполнителя РОБОТ, «живущего» в прямоугольном лабиринте на клетчатой плоскости: вверх, вниз, влево, вправо. При выполнении этих команд РОБОТ перемещается на одну клетку соответственно: вверх, вниз, влево, вправо. Четыре команды проверяют истинность условия отсутствия стены у той клетки, гле находится РОБОТ: сверху свободно, снизу свободно, слева свободно, справа свободно. Цикл «ПОКА <условие> команда» выполняется, пока условие истинно, иначе происходит переход на следующую строку. Если РОБОТ начнет движение в сторону стены, то он разрушится и программа прервется. Сколько клеток приведенного лабиринта соответствует требованию, что, выполнив предложенную ниже программу, РОБОТ уцелеет и остановится в той же клетке, с которой начал движение? (Приведена программа, состоящаяиз последовательности циклов «ПОКА <условие> команда»).

Дистракторы:

- случайное число.

Сводные таблицы

Обозначение	Название библиотеки	Количество обращений	Сложность алгоритма	Трудоемкость реализации
gcon	Константы	10	низкая	низкая
gexp	Выражения	3	низкая	низкая
gtr	Преобразования выражений	5	средняя	низкая
glb	Линейные блоки кода	1	средняя	низкая
gif	Ветвления	1	средняя	низкая
gfor	Циклы FOR	1	высокая	средняя
gcol	Выражения в столбик	0	низкая	средняя
gmask	Маски	1	низкая	низкая
gtab	Таблицы и запросы	2	средняя	высокая
grule	Правила	1	высокая	средняя
gam	Матрицы смежности	1	средняя	высокая
lgr	Графы	1	низкая	высокая
ldia	Диаграммы	1	низкая	высокая
llab	Лабиринт	1	высокая	высокая
ltt	Таблицы истинности	1	низкая	средняя
lbit	Биты	1	низкая	низкая
lstr	Строки	1	низкая	низкая
lnum	Числа	1	низкая	низкая
lnot	Системы счисления	3	низкая	низкая

Таблица 2: Описания библиотек

№	Справочники констант	Используемые библиотеки
A1	нет	gcon, lstr
A2	да	gcon, lbit, lnum
A3	нет	gcon, lnot
A4	нет	gcon, lnot
A5	нет	glb, gtr
A6	да	gif, gfor, gtr
A7	да	gcon, gexp, gtr
A8	нет	gexp, gtr
A9	нет	gexp, gtr, ltt
A10	да	gam, lgr
A11	нет	gcon, lnot
A12	да	gcon, grule
A13	да	gcon, gmask
A14	да	gtab
A15	да	gcon
A16	да	gcon
A17	да	gtab, ldia
A18	нет	llab

Таблица 3: Элементы генерации заданий

№	Работа с русским языком	Возможна необходимость повторной генерации	Знание алгоритма облегчает решение	Параметр определяет алгоритм решения
A1	нет	нет	да	да
A2	да	нет	да	нет
A3	нет	нет	нет	нет
A4	нет	нет	нет	нет
A5	нет	да	нет	да
A6	нет	да	нет	да
A7	нет	да	нет	нет
A8	нет	да	нет	нет
A9	нет	да	нет	нет
A10	нет	да	да	да
A11	нет	нет	нет	нет
A12	да	да	нет	да
A13	нет	нет	да	нет
A14	нет	да	нет	нет
A15	нет	нет	нет	нет
A16	нет	нет	нет	да
A17	нет	нет	нет	нет
A18	нет	да	да	нет

Таблица 4: Алгоритмические особенности генерации

Результаты

На основе приведенных данных был сделан вывод о том, что наиболее употребительными являются библиотеки генерации констант, выражений, преобразований, таблиц и запросов, функций по работе с системами счисления.

К другим библиотекам обращение происходит только из одного задания, что свидетельствует об их специализированности в рамках данного теста. Однако, поскольку охват областей знаний при составлении ЕГЭ является очень широким, каждая из этих мало используемых бибилотек описывает целую область знаний и может быть многократно востребована при составлении более узких тестов.

Были рассмотрены предположительные трудоемкость и сложность реализации каждой библиотеки. При этом наиболее трудоемкими были признаны библиотеки, в которых происходит работа с графическим материалом: таблицами, диаграммами, графами, — а наиболее сложными — библиотеки, в которых генерация алгоритмов тесно связана с генерацией их описаний, как, например, в заданиях с циклами, правилами и лабиринтом.

При создании шаблонов широко используются возможности языка программирования. Наиболее распространенными являются следующие случаи:

- использование переменных для хранения промежуточных и конечных результатов генерации;
- использование списков для генерации случайных элементов из них;
- использование ассоциативных массивов для хранения и генерации взаимосвязанных значений;
- использование ветвлений для выбора шаблона, если для данного задания шаблонов несколько;

- использование арифметических операций для детерминированных преобразований ответов.

Однако пришлось столкнуться также и с ограничением языка описания. Например, он не поддерживает диапазоны, границы которых являются выражениями.

Кроме того, была выявлена проблема производительности: генерация одного задания занимает 0,5 секунды. Эта проблема связана с неоптимальностью алгоритмов и затратами при использовании встроенных средств библиотеки Template::Toolkit.

В ходе анализа также были обнаружены следующие особенности заданий:

1. Литературность задания влияет на объем работ по грамматическому изменению формулировки после генерации параметров. Это изменение состоит в подборе правильного рода, числа и падежа для слов, грамматически зависимых от сгенерированных значений.

2. Для сложных алгоритмов генерации, настраиваемых при помощи большого числа параметров, существует альтернатива в виде многократной генерации по более простым алгоритмам до получения желаемого результата.

3. Для некоторых дистракторов справедливо утверждение о том, что знание алгоритмов их генерации облегчает процесс выбора правильного ответа. Например, если известно, что в задании используется дистрактор, увеличивающий на единицу правильный ответ, то из двух вариантов, отличающихся на единицу, учащийся выберет меньший.

4. Для заданий с высоким уровнем абстракции генерация параметра влияет на выбор алгоритма генерации всего задания в целом.

7 Заключение

В ходе выполнения дипломной работы были изучены:

- примеры тестов по математике, информатике и программированию;
- системы автоматизации тестирования;
- объектно-ориентированное программирование на языке Perl;
- язык создания шаблонов Template::Toolkit.

Была разработана система, осуществляющая генерацию заданий на основе шаблонов с использованием библиотеки классов и функций, а также возможностей языка программирования Template::Toolkit.

Основными достижениями являются:

- генерация выражений и вычисление их значений;
- описание эквивалентных и неэквивалентных преобразований;
- генерация примеров сложения в столбик и дистрактора для них;
- генерация присваиваний и линейных упорядоченных блоков кода;
- генерация заданий из ЕГЭ по информатике при помощи комбинаций функций из библиотеки и средств языка Template::Toolkit
- анализ генерации заданий ЕГЭ по информатике.

В процессе работы были выявлены следующие проблемы, для решения которых требуются дальнейшие исследования:

- проблема выбора алгоритма для отсечения нежелательных выражений;
- проблема наращивания функциональности по мере расширения области применения;

- проблема ограниченности языка программирования;
- проблема производительности.

Дальнейшее усовершенствование и развитие системы может проходить по следующим направлениям:

- типизация выражений и операторов;
- использование оптимизационного алгоритма для набора преобразований;
- генерация сложных частей программного кода, содержащих ветвления и циклы;
- расширение библиотек вспомогательных функций.

На настоящий момент система включает в себя все основные возможности других подобных систем, такие, как генерация специализированных тестов и генерация тестов на основе набора соответствий. Кроме того, она достигла достаточной универсальности в описании заданий и при соответствующем расширении функциональности может быть использована для генерации тестовых заданий ЕГЭ по информатике и в других областях.

Список литературы

[1] Dagand Pierre-Evariste. Filet-O-Fish Tutorial: The Fugu Error Definition Language. — Режим доступа: http://perso.eleves.bretagne.ens-cachan.fr/ dagand/fof/Fugu.pdf

[2] Klimke Andreas. Random Expression Generator, 2003. — Режим доступа: http://matlabdb.mathematik.uni-stuttgart.de/download.jsp?MC_ID=9&MP_ID=118

[3] Koji Yokokawa. Script Synthesis Tool for Non-Experienced Programmers. C5 2006: 218-221. — Режим доступа: http://www.informatik.uni-trier.de/ ley/db/indices/a-tree/y/Yokokawa:Koji.html

[4] Mathews Jay. Just Whose Idea Was All This Testing? //Washington Post. — Tuesday, November 14, 2006 — Режим доступа: http://www.washingtonpost.com/wp-dyn/content/article/2006/11/13/AR2006111301007_2.html

[5] NP Проект. Генетическое программирование, 2008 г. — Режим доступа: http://np-soft.ru/npproject/research/ga/gp_alg.htm

[6] Ravitch Diane. Education: A Brief History of Testing and Accountability // Hoover Digest. — 2002, №4. — Режим доступа: http://www.hoover.org/publications/digest/4495866.html

[7] Rugina Ana-Elena. System Dependability Evaluation using AADL (Architecture Analysis and Design Language). — Режим доступа: http://rjcitr05.loria.fr/papiers/papier_final/AnaRugina_RJCITR.pdf

[8] Template::Toolkit Home Page. Режим доступа: http://template-toolkit.org/index.html.

[9] The Perl Programming Language. Режим доступа: http://www.perl.org/

[10] Аванесов В.И. Теория и методика педагогических измерений (материалы публикаций в открытых источниках Интернет). — ЦТ и МКО УГТУ-УПИ, 2005 г. Режим доступа: old.ustu.ru/Аванесов%20В.С.pdf?id=2421&jf=yes

[11] Генерация булевского выражения, 2009 г. — Режим доступа: http://comp-science.narod.ru/matem/matem.html

[12] Грушевский С.П. Методика конструирования систем генерации индивидуальных домашних заданий по математическому анализу с применением пакетов прикладных программ. — Режим доступа: http://window.edu.ru/window_catalog/files/r24265/2000_3-4_032.pdf

[13] Демонстрационный тест ЕГЭ по информатике для 11 класса, 2009 г. — Режим доступа: http://www.ucheba.ru/ege-article/8356.html

[14] Джонстон Г. Учитесь программировать / Г.В.Сенин. — М.: Финансы и статистика, 1989 г.

[15] Дьяченко Е.П. Редактор курсов. — Дипломная работа, ИМКН ДВГУ, кафедра информатики, 2010 г.

[16] ЕГЭ по информатике и ИКТ, 2010 г. — Режим доступа: http://www.alleng.ru/edu/comp2.htm

[17] Институт системного программирования Российской академии наук. Pinery: Генерация на основе регулярных выражений Perl, 2006 г. — Режим доступа: http://www.unitesk.ru/pinery/pinery-regexp.php

[18] Информационно-экономический лицей г. Новосибирска. Тесты: Программирование на языке Pascal, 2008 г. — Режим доступа: http://l_infoeko.edu54.ru/p65aa1.html

[19] История тестирования // Журнал «Работа + Карьера» — Режим доступа: http://www.parta.com.ua/articles/interesting_edu/86/

[20] Керниган Б., Ритчи Д., Фьюэр А. Язык программирования Си. Задачи по языку Си. — М.: Финансы и статистика, 1985 г.

[21] Кодификатор элементов содержания и требований к уровню подготовки выпускников общеобразовательных учреждений для единого государственного экзамена 2010 года по информатике и ИКТ. // подготовлен Федеральным государственным научным учреждением «Федеральный институт педагогических измерений». — Режим доступа: http://rish.ru/upload/iblock/366/inform_kodif_2010.pdf

[22] Кручинин В.В. Методы генерации тестовых заданий по информатике. — Иформатика и образование №2, 2005 г., с. 87-93. — Режим доступа: http://www.tcde.ru/articles/docs/e1d2cf013e2e51f97e105597fe048ad1.pdf

[23] Кручинин В.В. Использование деревьев И/ИЛИ для генерации вопросов и задач. — Режим доступа: http://sun.tsu.ru/mminfo/000063105/284/image/284_182-186.pdf

[24] Кручинин В.В., Морозова Ю.А. Модели и алгоритмы генерации задач в компьютерном тестировании. — Известия Томского политехнического университета, Т.307, №5, 2004 г., с. 127-131. — Режим доступа: http://www.duskyrobin.com/tpu/2004-05-00030.pdf

[25] Мирсанов Р. И. Описание программного комплекса SuperTest, 2002 г. — Режим доступа: http://ipg.h1.ru/tests/mirsanov.files/super_test.html

[26] Молодцов В.А. Рыжикова Н.Б. Информатика. Тесты, задания, лучшие методики. — Ростов-на-Дону: Феникс, 2008 г.

[27] Пойя Д. Как решать задачу (пособие для учителей). — М.: Государственное учебно-педагогическое издательство министерства просвещения РСФСР, 1959 г.

[28] Пси-шпаргалка психологический образовательный сайт. Режим доступа: http://psylist.net/slovar/5a73.htm

[29] Себеста Роберт У. Основные концепции языков программирования. — М.: Издательский дом «Вильямс», 2001 г.

[30] Центр тестирования Chopin. Тесты по информатике и информационным технологиям, 2008 г. — Режим доступа: http://altnet.ru/ mcsmall/cat_inf.htm

[31] Шестаков А.П. Генерация дидактических материалов по математике, 2000 г. — Режим доступа: http://comp-science.narod.ru/matem/matem.html

[32] Шестаков А.П., Широких А.А. Контрольный тест (1999-2000 учебный год), II курс (факультет информатики и экономики, математический факультет ПГПУ, дисциплина "Языки и методы программирования"), 2000 г. — Режим доступа: http://comp-science.narod.ru/KR/2000_II_III.html

[33] Якушкин П.А. Лещинер В.Р. Кириенко Д.П. Информатика. Типовые тестовые задания. — М.: Издательство «Экзамен», 2010 г.

[34] Ярцева О.В. Цикина В.Р. Кириенко Е.Н. Информатика: ЕГЭ-2009: Самые новые задания. — М.: АСТ: Астрель, 2009 г.