Посов Илья Александрович

ПРОГРАММИРОВАНИЕ ГЕНЕРАТОРОВ ЗАДАЧ

Аннотация

Многовариантная задача – это задача, имеющая несколько вариантов условия. Условия задачи могут создаваться автоматически с помощью программ, называемых генераторами. В статье обсуждаются особенности программирования генераторов и разработанный для создания генераторов язык программирования, основанный на JavaScript. В стандартную библиотеку языка входят функции генерации случайных чисел, формирования текстов, а главное, связи с системой компьютерной алгебры Maxima.

Ключевые слова: многовариантные задачи, генерация задач, JavaScript, Maxima.

ВВЕДЕНИЕ

При проведении контрольных работ по математике преподаватель должен предпринять все усилия, чтобы ученики не списывали друг у друга решения задач. Обычно на контрольной каждый ученик получает один из двух вариантов условий, чтобы ему было сложнее списать решение у соседа. Чтобы еще больше усложнить списывание, можно выдать каждому ученику свое условие задачи. В таком случае контрольную можно выдавать на дом, и это часто практикуется в вузах. Распространению такого метода борьбы со списыванием мешает то, что готовить несколько вариантов условия одной задачи сложно. Без дополнительных инструментов работа превращается в нетворческую и скучную рутину. Неудивительно, что многие преподаватели независимо приходят к одной и той же мысли – написать программу, которая будет автоматически создавать варианты условия задачи с ответами. Например, на своем сайте Шестаков А.П. делится опытом написания подобных программ [1] для школьного курса алгебры.

Будем называть *генераторами* программы, которые создают условия задач. Генераторы можно программировать без использования каких-либо технологий, но удачная технология в некоторых случаях позволяет создавать их даже преподавателям, не знакомым с программированием. По технологии создания генераторы можно разделить на четыре типа: программируемые, параметрические, предметно-ориентированные и выбирающие. Для создания программируемого генератора необходимо написать программу. Это базовый и самый очевидный способ, а в некоторых ситуациях он и единственный возможный. Например, для изготовления генератора задачи на какой-либо алгоритм дискретной математики необходимо этот алгоритм запрограммировать.

Параметрические генераторы не требуют полноценной программы. В них задается текст условия, в него вставляются параметры, которые будут заменены на

конкретные числа при получении окончательных условий. Кроме этого указывается диапазон возможных значений параметров и описываются условия на параметры, чтобы задачи имели смысл и одинаковую сложность. Метод хорошо подходит для простых генераторов, но, если требуется сложная разветвленная логика проверки условий и задания значений параметрам, приходится возвращаться к программированию. Параметрическая генерация используется, например, в [2]. Параметрическую генерацию также можно использовать и без компьютера, например, учащиеся могут самостоятельно выбирать и подставлять значения параметров для своей задачи в зависимости от букв своего имени или даты рождения.

Предметно-ориентированные генераторы могут быть созданы без опыта программирования, но при этом класс получаемых задач очень узок. Преподаватель запускает созданную заранее для него программу, задает несколько настроек нового генератора и после нажатия кнопки получает генератор. Например, может быть создан редактор генераторов на задачи курса математической логики по переводу формул между несколькими стандартными представлениями (КНФ, ДНФ, многочлены Жигалкина). При этом преподаватель настраивает направление перевода и сложность формул.

Последний тип генераторов – это выбирающие генераторы. Такой генератор не создает условие сам, а выбирает одно условие из заранее заданных. Ввести несколько условий может практически любой пользователь компьютера, поэтому этот метод создания генераторов доступен наибольшему количеству преподавателей. Такая технология, в отличие от предыдущих, не помогает создать несколько условий задач. Напротив, создавать их надо вручную. Тем не менее, выбирающие генераторы имеют ряд достоинств, например, они платформонезависимы, то есть могут запускаться на любом подготовленном для них компьютере, что иногда делает их незаменимыми, но обсуждение этого уже выходит за рамки статьи.

В этой статье мы остановимся на самой мощной технологии – программируемых генераторах. Работу программиста можно значительно упростить, если предоставить ему полезные при создании генераторов библиотеки, а также избавить от повторяющихся во всех генераторах действий, таких как инициализация генераторов случайных чисел, открытие, закрытие файлов для вывода условий и т. п. Мы обсудим язык создания генераторов, реализованный в инструменте создания генераторов для Интернет-портала «База генерируемых задач» [3]. Предполагается, что преподаватели, использующие портал, способны программировать генераторы на несложном для первоначального изучения языке, но при этом достаточно выразительном, чтобы можно было создать генератор любой сложности.

ПРОГРАММИРОВАНИЕ ГЕНЕРАТОРОВ

Парадигма языко-ориентированного программирования [4] предлагает метод создания сложных систем, при котором вначале создаются специализированные предметно-ориентированные языки, которые более удобно использовать для создания системы, чем исходный язык общего назначения, а после этого разработка ведется уже на них. Особенно хорошо такие языки помогают, если они оказываются понятны специалистам предметной области, для которых разрабатывается система, и часть разработки они могут взять на себя. При этом специалисты реализуют только логику и не разбираются в других аспектах работы системы. Парадигма языко-ориентированного программирования и предметно-ориентированных языков хорошо работает в случае проблемы генерации задач. Преподаватели, то есть специалисты в предметной области, во время программирования должны полностью абстрагироваться от всех особенностей платформы, не задумываться, кто, как запускает генератор, как потом

форматирует созданные условия для вывода на печать. Преподаватель должен сконцентрировать усилия только на создании текста условия и ответа задачи.

Система метапрограммирования MPS (Meta Programming System) [4] позволяет создавать предметно-ориентированные языки, причем сразу вместе с редактором для них, а точнее, с полноценной средой разработки. Автор языка может легко добавить в редактор для своего языка такие современные возможности сред разработки, как автодополнение, рефакторинги и др. К сожалению, MPS пока тяжеловесна, требует установки для использования созданных в ней языков и не всегда очевидна в интерфейсе. В будущем к ней обязательно стоит присмотреться и создать с ее помощью язык программирования генераторов, а пока было принято решение обойтись без нее.

Создавать свой язык программирования без такого инструмента, как MPS, – сложно. В первую очередь, потому, что без среды разработки им будет трудно пользоваться, а создание удобной среды разработки занимает значительное время. Для инструмента создания генераторов, следовательно, нужен уже существующий язык. Перед тем как мы выберем существующий язык, давайте попытаемся представить, каким был бы самый удобный язык программирования генераторов.

Выберем задачу, на которой можно продемонстрировать ряд аспектов программирования генераторов. Пусть это будет следующая задача: восстановить квадратный трехчлен по значениям в двух точках и по значению производной в одной из этих точек. Гипотетический код программы приведен в листинге 1.

В начале программы генератора перечисляются условия и ответ к задаче. Условие и ответ записаны в формате TeX, поэтому формулы выделяются символом `$`. TeX является системой компьютерной верстки, которая легко сочетает текст и математические формулы любой сложности. Качество вывода формул в TeX общепринято считается наилучшим, поэтому использование TeX для форматирования условий и ответов математических задач является закономерным.

Символ `%` помечает параметры в условии и ответе. Символы, которые находятся после него, задают имя параметра, он будет заменен на конкретное значение перед выводом задачи на печать. Для оригинального TeX символ `%` означает комментарий.

В конце программы перечисляются условия на параметры. Первое и второе условия говорят, что все параметры являются целыми числами и находятся в диапазоне от –9 до 9. Это нужно, чтобы получаемые условия задачи имели одинаковую сложность. В следующем условии требуется, чтобы коэффициенты квадратного трехчлена не были равны нулю, что тоже нужно, чтобы избежать упрощения зада-

Листинг 1

```
Условие: Найдите квадратный трехчлен, если известно, что его значение
в точке $%x1$ равно $%y1$, производная в $%x1$ равна $%z1$, а значение
в $%x2$ равно $%y2$.
Ответ: $%p(x)$
Параметры:
     x1, x2, a, b, c, y1, y2, z1 ← Целое
     -10 < x1, x2, a, b, c, y1, y2, z1 < 10
     a, b, c, x1, x2 != 0
     x1 != x2
     p(x) = a x^2 + b x + c
     y1 = p(x1)
     y2 = p(x2)
     z1 = p′(x1)
```

чи. Далее требуется, чтобы точки из условия были различны. Если это так, то задачу можно решить при любых исходных данных.

Далее описывается квадратный трехчлен *p*. Заметим, что в ответе задачи этот квадратный трехчлен так напрямую и записан, интерпретатор должен самостоятельно отформатировать многочлен для вывода на печать и избавить от этой работы программиста. Без встроенного форматирования вывода математических объектов, таких как многочлены, код генератора значительно увеличивается. Чтобы убедиться в этом, попробуйте написать программу, которая по трем коэффициентам квадратного трехчлена создаст строку вида «x^2 – 2x + 1», учитывая все возможные ситуации с единичными, нулевыми и отрицательными коэффициентами.

Далее перечисляются условия, которые определяют параметры $y1$, $y2$, $z1$. Здесь заметим, что определение значений параметров написано «непринужденно». Вместо выражения $y1 = a*x*x+b*x+c$ используется $p(x)$, которое уже определено как функция. Более того, далее вместо возможного $2*a*x+b$ написано $p'(x)$, что значительно более наглядно и читаемо.

Программа генератора – декларативна. Она описывает не то, *как* получить условие задачи, а то, *каким* должно быть условие задачи. К сожалению, для полностью декларативных генераторов трудно создать эффективный интерпретатор. Интерпретатор должен подобрать значения параметров, чтобы они подошли ко всем условиям. Подобрать параметры может быть нелегко, потому что в общем случае для этого требуется решать задачи удовлетворения ограничений, которые сложны и изучаются в области искусственного интеллекта. В приведенном примере генератора используются только простые условия равенства и неравенства, но в реальном генераторе могут встречаться и более сложные, например, условия на взаимную простоту чисел. Условия могут быть и такими, что их можно только запрограммировать, например, условие, что число не является суммой степеней двойки и тройки.

Ясно, что задачи удовлетворения ограничениям легко избежать, если немного отойти от декларативного программирования и все-таки заставить программиста самостоятельно указать последовательность действий по подбору параметров. Разобьем последний раздел программы на два (см. листинг 2).

В разделе «Параметры» последовательно задаются значения всех параметров. Некоторые параметры свободны, некоторые – вычисляются на основе уже заданных. Написанную программу можно понимать двумя способами. Либо как указание, как перебирать свободные параметры, чтобы получить все возможные условия задачи. В данном случае перебирать надо $x1$, $x2$, a, b, c, изменяя их последовательно с –9 до 9. Второй способ понять эту программу, более применимый на практике, состоит в том, что значения параметров нужно выбирать случайным образом из заданного множества. То есть все свободные параметры выбираются случайно из диапазона от –9 до 9.

Листинг 2

```
Параметры:
     x1, x2, a, b, c ← Целое в диапазоне(-9, 9)
     p(x) = a x^2 + b x + c
     y1 = p(x1)
     y2 = p(x2)
     z1 = p'(x1)
Условия на параметры:
     -10 < y1, y2, z1 < 10
     a, b, c, x1, x2 != 0
     x1 != x2
```

Программист должен аккуратно работать со случайным выбором параметров. Может оказаться, что разные условия задач генерируются с разной вероятностью. Для примера рассмотрим простой генератор задачи «сложите числа *a* и *b*», причем оба слагаемых и сумма должны быть цифрами в диапазоне от 1 до 9. Возможных условий задач получается 36. (8 условий для 1+, 7 условий для 2+ и т. д.) Два варианта программы генератора могут быть устроены так:

```
Параметры:
    a ← Целое в диапазоне(1, 9)
    b ← Целое в диапазоне(1, 9)
Условия на параметры:
    a+b <= 9
```

или

```
Параметры:
    a ← Целое в диапазоне(1, 8)
    b ← Целое в диапазоне(1, 9-a)
```

Во втором случае мы избежали раздела программы «условия на параметры», потому что получаемые значения параметров сразу же задают корректное условие задачи. Но легко понять, что условие 8 + 1 будет получаться чаще всех других, потому что вероятность этого события равна вероятности выбора *a* как 8, то есть равна 1/8 вместо необходимых 1/36 для равновероятных условий. Избежать разной вероятности можно, либо пользуясь первым способом задания генератора, либо сделав выбор параметра *a* неравномерно распределенным.

В общем случае программу не удается разделить на два раздела «Параметры» и «Условия на параметры». В целях эффективности требуется проверять условия не после того, как подобраны все параметры, а в течение процесса подбора параметров. Например, выбрав два параметра, можно сразу проверить, что они, допустим, взаимно просты, и только после этого продолжать выбор параметров. Кроме того, в сложных генераторах логика подбора параметров разветвлена и нелинейна. Поэтому в общем случае программа генератора содержит только один раздел, в котором операции выбора параметров чередуются с проверками условий.

Итак, мы обсудили общий вид программы создания генераторов. Она состоит из описания шаблона условия, ответа, последовательности выбора значений параметров и проверки условий на параметры. Создаваемые в генераторе условия обрабатываются интерпретатором, поэтому генератор не должен самостоятельно, например, сохранять условия в файл или формировать страницу контрольной.

Из других особенностей программирования генератора мы увидели активную работу со случайными числами, доступ к возможностям систем компьютерной алгебры (в примере: символьное вычисление производной) и форматирование текста и математических объектов для вывода в условие (в примере: форматирование многочлена). Форматирование математических объектов тоже поддерживается системами компьютерной алгебры, а вот форматирование текста должно быть поддержано отдельно. Следовательно, в стандартной библиотеке языка, то есть в библиотеке, которая доступна из всех программ по умолчанию, должны быть функции генерации случайных чисел, функции доступа к системе компьютерной алгебры и расширенные возможности работы со строками.

Некоторые другие тонкости программирования генераторов описаны в [5].

ВЫБОР ЯЗЫКА

По своему смыслу язык создания генераторов – это скриптовый язык, потому что он способен исполняться только в рамках системы, в которую внедрен. Код генератора не должен иметь доступ к окружающему миру, например, читать, писать с диска, открывать сетевые соединения, рисовать на экране. Его целью является только создание текста условия и ответа. Из скриптовых языков самыми известными являются Python, JavaScript, Lua. Именно они максимально приспособлены для расширения и использования в рам-

ках предметно-ориентированных систем. Для инструмента создания генераторов был выбран язык JavaScript. (Более правильно было бы назвать этот язык ECMAScript, потому что JavaScript это его диалект, используемый для придания интерактивности веб-страницам. Другой известный диалект ECMAScript – это ActionScript, используемый во Flash приложениях). Одним из ключевых критериев выбора являлось то, что JavaScript способен исполняться внутри браузера без дополнительной установки каких-либо компиляторов и интерпретаторов. То есть инструмент создания генераторов потенциально может быть реализован как веб-приложение. Помимо прочего, JavaScript общепризнанно прост в изучении, например, некоторые пользователи JavaScript хорошо справляются со сценариями на веб-страницах, но при этом никогда не учились программировать. Утверждается, что одной из целей разработчиков JavaScript как раз было сделать его удобным для непрограммистов. Для программистов JavaScript синтаксически похож на другие популярные языки программирования C++ и Java, но при более глубоком изучении он оказывается непривычным. В частности, трудности могут вызывать области видимости переменных, а вместо привычных для ООП классов, придется переучиваться на прототипы. Тем не менее, минимального знания JavaScript вполне достаточно для создания генераторов.

Вспомним, что одной из особенностей программирования генераторов является использование систем компьютерной алгебры. Системы компьютерной алгебры сами имеют встроенные языки, на которых можно писать полноценные программы, причем при использовании системы хотя бы минимальное знание этих языков необходимо. Для примера посмотрим на одну и ту же операцию раскрытия скобок в двух системах компьютерной алгебры. В Mathematica: `Expand[(x+1)(x+2)]`, в Maxima: `expand((x+1)*(x+2))`. В первом случае аргумент функции записывается в квадратных скобках, во втором – в круглых. Между множителями Mathematica разрешает не ставить знак умножения, Maxima – требует знак обязательно. Устройство циклов, условных операторов в этих системах различаются принципиально. Возникает вопрос, а нельзя ли программировать генераторы сразу на языке системы компьютерной алгебры, чтобы учить только ее и не учить JavaScript? Можно. Системы имеют доступ к диску, позволяют писать файлы, то есть на них возможно реализовать среду разработки генератора. Но, во-первых, эти языки не имеют удобных средств разработки. Даже Mathematica при написании кода позволяет только следить за правильностью расстановки скобок и при подсветке синтаксиса обозначает лишь определенные ранее символы. Во-вторых, эти языки не так легки для изучения непрограммистами. В основе своей они являются функциональными и используют много понятий функционального программирования. Это связано с тем, что языки активно работают с вычислениями и должны управлять процессом вычислений. Помимо этого одним из основных типов данных являются списки, так как результаты вычислений могут быть неоднозначны. Например, при поиске корней многочленов возвращаются списки найденных корней. Работа со списками может происходить императивно с помощью циклов, которые в языках систем компьютерной алгебры тоже есть, но в функциональных языках принято использовать ряд специализированных функций обработки списков. Самой известной из таких функций является функция **`map`** (применяет заданную функцию к каждому элементу списка и составляет новый список из полученных значений).

Для программирования генераторов была выбрана система компьютерной алгебры Maxima, ее основное достоинство – это бесплатность (лицензия GPL). К сожалению, по сравнению с платными, у нее есть ряд недостатков. Основной из них – это трудность общения с внешними программами. Так как генератор работает вне системы, ему нужно к ней обращаться.

Mathematica имеет хорошо документированный интерфейс для внешних программ MathLink, Maple имеет OpenMath, а Maxima для общения с внешним миром использует только базовые методы наподобие пакетной обработки команд из файла или соединение через TCP сокет. При всех способах соединения с Maxima требуется самостоятельно анализировать ее вывод, который является неформатированным текстом, предназначенным, в первую очередь, для восприятия человеком. Трудность возникает даже при определении, произошла ли ошибка. Более того, разработчики сами признают, что Maxima еще недостаточно готова для использования во внешних программах. Для целей генерации было выбрано общение через TCP сокет, потому что оно не требует запуска Maxima каждый раз, когда необходимо исполнить одну команду. Интересной особенностью Maxima при общении через сокет является то, что она выступает не сервером, обрабатывающим запросы, а клиентом. То есть логика ее работы в том, что она обращается к внешней программе-серверу с вопросом, какое вычисление требуется произвести.

ОПИСАНИЕ ЯЗЫКА СОЗДАНИЯ ГЕНЕРАТОРОВ

Перейдем к возможностям языка программирования генераторов. Как уже было сказано, за основу его выбран JavaScript. Несмотря на то, что программа на JavaScript может исполняться внутри браузера, инструмент создания генераторов был написан как отдельная программа. Для исполнения JavaScript кода был подключен движок V8 от компании Google, который они используют в своем браузере Google Chrome. Использование движка позволяет значительно расширить возможности JavaScript, но все расширения необходимо делать таким образом, чтобы их можно было повторить с помощью JavaScript внутри браузера. Сложнее всего при этом разобраться со связью с системой компьютерной алгебры. Отдельная программа может легко соединиться с Maxima любым методом, потому что она не ограничена моделью безопасности браузера для исполняемых скриптов. Если же генератор исполняется внутри браузера, чтобы иметь возможность соединить его с Maxima, необходимо создать веб-интерфейс к Maxima по протоколу HTTP. Технически это реализовать можно, поэтому обращение к Maxima из JavaScript может работать и внутри браузера.

Код написанных выше программ не похож на JavaScript, но его можно незначительно преобразовать, чтобы он стал корректной JavaScript программой. Разделы «условие» и «ответ» оформляются как присвоение значений переменным statement и answer соответственно. Проверка условий оформляется через функцию assert, которая проверяет записанное в ней условие, и, если оно не выполняется, запускает генерацию заново. Перепишем на JavaScript генератор для задачи про сумму двух чисел:

```
statement = "Сложите числа %a и %b";
answer = "%c";
a = rnd(1,8);
b = rnd(1,8);
c = a + b;
assert( c <= 9 );
```

Функция **rnd** выбирает случайное число из заданного диапазона. Ниже мы обсудим другие возможности генератора случайных чисел, а сейчас остановимся на возможностях формирования текста.

ФОРМИРОВАНИЕ ТЕКСТА

В приведенном только что коде продемонстрированы базовые возможности формирования текста. После окончания исполнения программы интерпретатор обрабатывает переменные **statement** и **answer**, заменяя в них выражения с **%** на значения соответствующих переменных. Тип переменной может быть произвольным, во всех случаях ее значение преобразуется к строке. Если переменная содержит объект, используется его метод **toString()**.

Такую замену переменных внутри строки можно делать и самостоятельно, для этого используется функция **php()** (по названию языка, в котором используется такая же работа со строками). Например, выражение **php("%a+%b")** вычисляется в строку **"2+2"**, если значения переменных *a* и *b* равны 2. Для отделения имен переменных используются фигурные скобки: строка **"%{a}dollars"** означает замену для переменной *a*. Чтобы использовать в строке символ **%** его надо продублировать. Выражение **php("%%")** вычисляется как **"%"**. Дублирование используется вместо более привычного в этой ситуации экранирования символом ****, потому что экранирование и так используется в JavaScript, в TeX, а три уровня экранирования еще и с функцией **php** было бы очень трудно использовать.

Экранирование всегда вызывает определенные проблемы, поэтому обсудим его подробнее. Вспомним, что условие задачи выводится в формате TeX. В TeX символ процента означает комментарий, а для вывода символа на печать требуется его экранировать: **"\%"**. Поэтому, если мы хотим на печати получить процент, в условии его надо оформить как **"\\%%"**. Что происходит при работе генератора? Так как мы написали **"\\%%"** в коде генератора на языке JavaScript, первые обратный слэш означает экранирование, и поэтому эта строка реально является строкой из трех символов **"\%%"**. После конца работы генератора запускается функция **php**, которая подставляет параметры и заменяет **"\%%"** на **"\%"**. Именно эта строка отдается TeX, и он выводит на печать процент. Без слэшей TeX увидел бы процент, который означает для него комментарий. Про экранирование нужно помнить не только при использовании символа процента. Условие написано в формате TeX, а в TeX все команды начинаются с обратного слэша. Поэтому в условии все команды должны начинаться с двойного слеша: **"\\frac{2}{3}"**. Это вывод дроби две трети, в котором без двойного слэша JavaScript увидит последовательность **"\f"**, которая означает команду «подача страницы», а не то, что имелось в виду.

Кроме функции **php**, существуют и другие способы форматирования строк. Это привычная программистам c++ функция **sptintf()**. Пример ее работы: **sprintf("%s+%s=%s", 2, 2, 2+2)** возвращает строку "**2+2=4**", в этой функции вместо значений переменных подставляются значения выражений, записанных в качестве параметров функции. Любой шаблон кроме **%s** внутри этой функции считается ошибкой.

Последний способ формирования строчек – использование объекта **MagicString**. Покажем его работу на примере.

```
s = mc("%a+%b=%c");
s.a = 2;
s.b = 2;
s.c = 4;
str = s.toString();
println(str);
println(s);
```

В первой строке создается объект. Написанные внутри него шаблоны задают поля объекта. В конце содержимое объекта преобразуется в строку и выводится функцией **println()**, которую мы сейчас обсудим. Функция **println()** преобразует аргумент в его строковое представление и выводит в отладочную консоль. Для инструмента создания генераторов эта консоль совпадает с обычной. При работе в браузере отладочная консоль должна отображаться прямо на странице. Эта функция вместе с **print()** (вывод в консоль без перевода строки) – единственные функции, используемые для отладки. Для более серьезной отладки нужна среда разработки, позволяющая отлаживать программу по шагам.

Если функция **println()** получает на вход строку, то она сначала отдает ее функции **php()** и дальше пользуется полученным результатом. Поэтому возможна запись **println("a=%a")**, тогда **%a** заменится на значение переменной *a*. Подобное поведение верно для всех функций стандартной библиотеки. Функция **assert()**, ко-

торую мы рассмотрели, работает со строками так же, хотя мы еще не обсудили, какой в этом случае она имеет смысл. Разработчикам дополнительных библиотек (о библиотеках ниже) рекомендуется тоже реализовывать это поведение в своих функциях.

Такая работа со строками может вызвать проблемы, если в обрабатываемых строках используются символы процента. Но этот символ нужен редко, а если нужен, то приходится аккуратно следить за обработкой строк. В крайних случаях можно пользоваться функцией **php1()**, которая обратна **php()**, и тогда, например, **println(php1("2%"))** выведет в консоль 2 процента. В Maxima символ процента используется для стандартных констант, например, числа π и e вводятся в Maxima как **%pi** и **%e**.

СЛУЧАЙНЫЕ ЧИСЛА

Стандартная библиотека содержит функцию **rnd()**, действие которой зависит от количества и типов параметров. Функция **rnd()** – случайное число из полуинтервала [0, 1), здесь и далее все распределения равномерны. Функция **rnd(x)**, где x – целое, выдает случайное целое число от 0 до $x - 1$. Функция **rnd(x, y)** – для целых параметров это случайное число в диапазоне от x до y включительно. При этом, если в диапазон попадает ноль, он не учитывается. Чтобы все-таки учесть ноль, используется функция **rnd0()**. По опыту создания генераторов почему-то оказывается, что функция **rnd0()** нужна очень редко. Последний вариант функции **rnd()** – это **rnd** с параметром, являющимся массивом. В этом случае выбирается случайный элемент из массива. Синтаксически подобный вызов можно оформить так:

```
rnd(["sin", "cos", "tan", "cot"]),
```

квадратные скобки в JavaScript создают массив. Сокращением для

```
rnd([true, false])
```

является функция **coin()**. Та же функция с параметром выдает **true** с указанной в параметре вероятностью. Последняя возможность генератора случайных чисел, которую мы здесь обсудим, – это функция **shuffle()**. На вход она получает массив и возвращает его же с переставленными случайным образом элементами.

При использовании генератора случайных чисел в программе генератора полностью отсутствует стандартная для других языков возможность инициализации генератора случайных чисел. Это необходимо для того, чтобы генератор всегда работал детерминировано. Ничего не должно мешать создать два раза один и тот же набор условий задач. Генератор случайных чисел инициализируется интерпретатором в зависимости от идентификатора преподавателя и специального вводимого преподавателем числа – номера серии. Выбирая одну и ту же серию, преподаватель должен получать одни и те же условия задач. Понятие серии аналогично понятию серии из системы генерации задач Степанова [6].

СВЯЗЬ С MAXIMA

Связь с Maxima реализована в языке посредством функции **meval()**. Любая переданная в эту функцию строка отдается Maxima, после чего результат вычисления возвращается обратно строкой. Если при вычислении в Maxima произошла ошибка, например, в переданной строке содержалась синтаксическая ошибка, то функция бросает исключение, которое при желании можно обработать. Обработку ошибок лучше избегать и давать Maxima только выражения, которые она может вычислить, не сообщая об ошибках. Функция **meval()**, как и все другие стандартные функции, подставляет в свой аргумент все значения переменных. Поэтому, например, можно написать **meval("gcd(%a,%b)")** и получить НОД чисел, хранящихся в переменных a и b.

Любое обращение к Maxima можно выполнить с помощью функции **meval()**. Но при этом нужно хорошо знать язык Maxima, чтобы суметь сделать необходи-

мые действия. Язык Maxima имеет много неочевидностей, что уже обсуждалось, когда была отброшена идея использовать для создания генераторов только этот язык без JavaScript. Например, Maxima не вычисляет логические выражения, если ее явно об этом не попросить. `meval("2>3")` так и останется после вычисления `"2>3"`, а вот `meval("is(2>3)")` уже вычислится в ложь `"false"`. Для упрощения работы с Maxima создан ряд дополнительных функций, которые позволяют совершать стандартные действия, не зная тонкостей языка. Здесь мы перечислим только некоторые такие функции. Функция `mevalNum()` вычисляет численное значение выражения, например, `meval("4/6")` вернет строку `"2/3"`, а вот `mevalNum("4/6")` вернет `"0.66666666666667"`. Функция `tex()`, вычисляет переданный ей параметр и преобразует его в формат TeX. Например, `tex("4/6")` вернет `"{{2}\over{3}}"`. Функция `assert()`, которой в параметре передается строка, отдает эту строку на вычисление Maxima, и, если результатом вычисления является что-либо, кроме `"true"`, считается, что утверждение было неверно, и генерация запускается заново. К примеру, `assert(2>3)` и `assert("2>3")` в обоих случаях перезапускают генератор, но в первом случае вычисления происходят внутри JavaSctipt, во втором – внутри Maxima.

Список операторов, упрощающих работу с Maxima, постепенно расширяется. Например, сейчас нет возможности с помощью функции `tex()` вывести дробь 4/6. Эта функция сначала производит упрощения, а потом уже преобразует результат в формат TeX. Ограничение можно обойти, выполнив перед этим команду `meval("simp:false")`, но очевидно, что должна существовать библиотечная функция, позволяющая решать эти проблемы без знания тонкостей работы Maxima.

НАСТРАИВАЕМЫЕ ГЕНЕРАТОРЫ

Предположим, что требуется написать генератор для задачи поиска обратной матрицы. Будем считать, что размер матрицы во время генерации постоянен, но один и тот же генератор можно использовать для создания задач с матрицами разного размера. Размер матрицы задается в параметре перед запуском генератора, а для получения значения параметра внутри генератора необходимо написать примерно код: `param("MatrSize", 3)`. Первый аргумент функции – это название параметра, второй – значение по умолчанию, которое вернет функция, если окажется, что значение параметра `MatrSize` не задано. При распространении генератора автор должен указывать, какие параметры у нее есть и какие в них используются значения по умолчанию.

БИБЛИОТЕКИ

В стандартную библиотеку невозможно включить всё, что может понадобиться при создании генератора. Например, работа с комплексными числами или геометрическими объектами пока не предусмотрена в стандартной библиотеке. Для использования стороннего кода вне генератора используется метод `import()`. В его параметре пишется путь к файлу с библиотекой. Семантика работы метода проста, по данному пути находится скрипт, написанный на JavaScript, и исполняется. В этом скрипте обычно определяются функции, которые после исполнения скрипта можно использовать в коде генератора.

Так как генератор может исполняться в браузере в рамках веб-приложения, понятие путей не соответствует полностью путям в файловой системе. Пример вызова функции:

`import("iposov.lib.Geometry")`.

Для инструмента создания генераторов, работающего отдельно от браузера, эта функция ищет файл Geometry.js в каталоге lib, лежащем в каталоге iposov, который, в свою очередь, лежит в одном из каталогов, которые указаны в настройках интерпретатора как каталоги с библиотеками. При исполнении внутри браузера в

рамках веб-портала библиотеки отдельно загружаются пользователями по указанным им путям.

ПРИМЕР ПРОГРАММЫ

В начале статьи мы приводили пример программы генератора для восстановления многочлена по его значениям в двух точках и значению производной в одной из этих точек. Попробуем написать теперь этот же генератор на языке создания генераторов (см. листинг 3).

В первой и второй строках переменным statement и answer присваиваются значения. Далее программа будет подбирать значения для параметров. В следующем блоке выбираются значения коэффициентов многочлена и точек, в которых вычисляются значения многочлена.

Третий блок самый содержательный. Первым делом Maxima получает команду на определение функции $p(x)$. При выполнении запроса вместо **%a**, **%b** и **%c** подставляются конкретные числа, то есть $p(x)$ имеет конкретные значения коэффициентов. С этого момента при общении с Maxima можно пользоваться функцией $p(x)$. Оператор присваивания **:=** только на первый взгляд имеет что-то общее с привычным присваиванием на Паскале. Для присваивания значения переменным используется только символ двоеточие. Помимо этого Maxima имеет и другие операторы присваивания, например **::=**.

Затем, вычисляются значения $y1$, $y2$. Здесь синтаксис выражения, отдаваемого Maxima, вполне очевиден. Чтобы подставить значение $x0$ в функцию $p(x)$, необходимо написать $p(x0)$. Следующим шагом вычисляется значение производной в точке. Теперь синтаксис становится неочевидным, точнее, вместо обычной производной, пишется композиция функций, и все эти функции необходимо знать при использовании Maxima. Функция **diff** вычисляет производную заданного выражения относительно заданной переменной. Функция **at** вычисляет заданное выражение при заданных значениях переменных. В нашем случае выражение с производной вычисляется при заданном значении x. Интересно сравнить этот синтаксис с вариантом, который использовался бы в Mathematica. Там вычисление производной функции $p(x)$ в точке, допустим 1, выглядело бы так: $p'[1]$. Единственное отличие от стандартных математических

Листинг 3

```
statement = "Найдите квадратный трехчлен, если известно, что его значение
в точке $%x1$ равно $%y1$, производная в $%x1$ равна $%z1$, а значение
в $%x2$ равно $%y2$.";
answer = "$%p$";

x1 = rnd(-9, 9);
x2 = rnd(-9, 9);
a = rnd(-9, 9);
b = rnd(-9, 9);
c = rnd(-9, 9);

meval("p(x) := %a*x^2 + %b*x + %c");
y1 = meval("p(%x1)");
y2 = meval("p(%x2)");
z1 = meval("at( diff(p(x),x), x=%x1 )");
p = tex("p(x)");

assert(-10 < y1 && y1 < 10);
assert(-10 < y2 && y2 < 10);
assert(-10 < z1 && z1 < 10);
assert(x1 != x2);
```

обозначений – это использование квадратных скобок вместо круглых, но в Mathematica аргументы функций всегда пишутся в квадратных скобках.

Далее, переменной p присваивается форматированный в TeX многочлен $p(x)$. Эта переменная будет подставлена как параметр в условие задачи.

Последний блок программы – это условия на параметры. Некоторые условия могли бы быть написаны раньше, например различие точек $x1$ и $x2$ можно было проверить в самом начале программы. Заметим, что условия о том, что коэффициенты и точки не равны нулю, которое присутствовало в исходном варианте программы, здесь отсутствует. Оно не требуется, потому что функция **`rnd`**, выбиравшая случайные значения, не выдает ноль.

На рисунках ниже приведены фрагменты листов с получаемыми условиями задач (рис. 1) и отдельно ответы к ним (рис. 2). Эти листы созданы порталом на основе приведенной выше программы. Форматирование условий в портале производится с помощью модуля вывода на печать генератора задач Степанова [6].

ЗАКЛЮЧЕНИЕ

Мы обсудили проблему генерации задач и язык создания генераторов, используемый в инструменте создания генераторов, разработанном для веб-портала «База генерируемых задач». Язык создания генераторов основан на JavaScript, что позволяет ему потенциально запускаться внутри браузера без установки дополнительных компиляторов или интерпретаторов, чем можно будет воспользоваться в будущих версиях портала. Помимо этого, JavaScript прост в использовании непрограммистами, и достаточно мощен, чтобы с его помощью можно было создать генератор любой сложности. Основными возможностями, делающими рассмотренный язык удобным инструментом создания генераторов, являются возможности формирования текста и, самое главное, связь с системой компьютерной алгебры Maxima.

Язык требует развития в части наполнения стандартной библиотеки. Особенно много функций необходимо для упрощения работы с Maxima. Универсальная функция **`meval()`** требует хорошего знания языка Maxima для совершения некоторых даже стандартных действий.

Вар. 13 Найдите квадратный трехчлен, если известно, что его значение в точке -2 равно 9, производная в -2 равна -9, а значение в 1 равно 0.	**Вар. 14** Найдите квадратный трехчлен, если известно, что его значение в точке 1 равно -5, производная в 1 равна -7, а значение в -2 равно 7.
Вар. 15 Найдите квадратный трехчлен, если известно, что его значение в точке 1 равно 6, производная в 1 равна -7, а значение в -1 равно 0.	**Вар. 16** Найдите квадратный трехчлен, если известно, что его значение в точке -2 равно 2, производная в -2 равна -4, а значение в 1 равно 8.
Вар. 17 Найдите квадратный трехчлен, если известно, что его значение в точке -1 равно 8, производная в -1 равна -9, а значение в 3 равно 4.	**Вар. 18** Найдите квадратный трехчлен, если известно, что его значение в точке 1 равно -6, производная в 1 равна 1, а значение в -2 равно 0.

Рис. 1. Задачи

Вар.13 $2x^2 - x - 1$	**Вар.14** $-x^2 - 5x + 1$
Вар.15 $-5x^2 + 3x + 8$	**Вар.16** $2x^2 + 4x + 2$
Вар.17 $2x^2 - 5x + 1$	**Вар.18** $x^2 - x - 6$

Рис. 2. Ответы

Литература

1. *Шестаков А.П.* Генерация дидактических материалов по математике // http://comp-science.narod.ru/matem/matem.html (21.07.2010), 2000.

2. *Левинская М.А.* Автоматизированная генерация заданий по математике для контроля знаний учащихся // Educational Technology & Society, 2002. № 5(4). С. 214–221.

3. *Посов И. А., Смирнов И. Б.* Интернет-сервис для хранения базы генерируемых задач по математике // Материалы международного форума «Современное образование: содержание, технологии, качество», 21–22 апреля, СПб.: СПбГЭТУ, 2010.

4. *Dmitriev S.* Language Oriented Programming: The Next Programming Paradigm // http://www.jetbrains.com/mps/docs/Language_Oriented_Programming.pdf (15.07.2010), 2004.

5. *Посов И. А.* Автоматическая генерация задач // Компьютерные инструменты в школе, 2007. № 1. С. 54–62.

6. *Степанов А.В.* Система компьютерной генерации заданий по математике // Компьютерные инструменты в образовании, 2000. № 3/4. С. 28–31.

Abstract

Multivarian problem is a problem that has several variants of statement. The statements may be created automatically by means of programs called generators. The paper discusses the peculiarities of generators programming and a programming language based on JavaScript, developed for generators creation. The standard library of the language contains functions dealing with random numbers, string manipulation, and the most importantly, connecting with computer algebra system Maxima.

Keywords: Multivariant problems, problems generation, JavaScript, Maxima.

Посов Илья Александрович,
ассистент кафедры ВМ–2
СПбГЭТУ «ЛЭТИ»,
iposov@gmail.com.