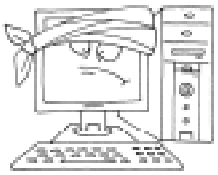
Богданов Михаил Сергеевич

АВТОМАТИЗАЦИЯ ПРОВЕРКИ РЕШЕНИЯ ЗАДАЧИ ПО ФОРМАЛЬНОМУ ОПИСАНИЮ ЕЕ УСЛОВИЯ

ПРОБЛЕМА ПРОВЕРКИ РЕШЕНИЙ ЗАДАЧ В ПРОГРАММНО-ПЕДАГОГИЧЕСКИХ СРЕДСТВАХ



С каждым днем компьютер и информационные технологии занимают все большее место в учебном процессе. Это и учебные материалы, доступные в глобальной сети, и различные компьютерные лаборатории, моделирующие те или иные физические процессы и позволяющие проводить виртуальные эксперименты.

С другой стороны, на данный момент возможности компьютерных технологий существенно уступают человеку в оценке усвоенных учеником знаний. Они не позволяют проверять правильность решений задач, представленных в естественной форме, и не способны оценить процесс мышления. Большинство подходов к оценке усвоения материала базируется на тестах (выбор правильного ответа из списка заданных), что вряд ли можно назвать хорошим способом проверки знаний, так как такой подход порождает специальную субкультуру «тестовых знаний», мало связанную с формированием подходов к решению реально существующих проблем.

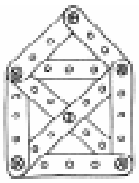
Другой интересный подход к проверке знаний реализован в олимпиадах по программированию (например в ACM), где критерием успешного решения задачи по программированию является успешность прохождения программой, в форме которой представлено решение, всех тестов. Если программа-решение правильно обрабатывает все тестовые данные и время ее работы укладывается в заранее оговоренный интервал времени, то задача считается решенной, в противном случае решение не принимается.

Однако и такой подход не решает полностью проблему проверки знаний: при автоматической проверке решения задачи важна не только проверка конечного результата в виде ответа, но и проверка логики всего решения. Для одной и той же задачи может быть представлено и красивое короткое решение, и тяжеловесное объемное решение, но система тестирования решения этого «не заметит». На данный момент с помощью существующих компьютерных средств реализовать проверку решений во всех деталях трудно, однако даже при проверке ответов к задачам имеются технологические и до сих пор не решенные проблемы.

Одной из таких проблем является проблема создания и хранения правильного ответа к задаче в той или иной форме. Так в тестовых программах ответ ученика сравнивается с эталонным ответом непосредственно, а в системах проверки программ-решений на олимпиаде тестирующие программы используют результаты, полученные на эталонной программе, которая и является «правильным» ответом.

В настоящей статье рассматривается иной подход к проверке ответа. Идея состоит в таком развитии языка описания математических задач, чтобы стало возможным использовать условие задачи для проверки ответа ученика (исключая, таким образом, этап сравнения ответа с эталонным). Для реализации этой идеи требуется не только разработать язык описания задач, но и построить механизмы превращения условий задач в алгоритмы проверки их решения. Из общей теории алгоритмов следует, что достичь поставленной цели в общем случае невозможно, однако это не значит, что нельзя достичь цели в частных случаях. В статье рассматривается применение данной идеи к задачам по комбинаторике. Задачи по комбинаторике обладают важным свойством, отличающим их, например, от задач математического анализа: они поставлены на конечном множестве объектов, и решение задачи может быть получено перечислением всех объектов, обладающих указанными в постановке задачи свойствами.

В представленной программной среде ответ задачи формируется дважды. Один раз ответ генерируется автоматически – программой по формальному описанию задачи. Второй раз ответ вводится учеником посредством специального интерфейса (для ввода ответов на стандартные школьные задачи по комбинаторике нужно иметь возможность ввода чисел и арифметических выражений, содержащих числа и стандартные комбинаторные функции, такие как вычисление числа перестановок, размещений, сочетаний). Проверка осуществляется сравнением этих ответов.



КОНСТРУКТОР КОМБИНАТОРНЫХ КОЛЛЕКЦИЙ (KKK)

В этом разделе будет представлено подробное описание среды KKK для конструирования комбинаторных задач, в которой задача описывается на языке, допускающем автоматическую проверку ответа.

В описании любой задачи, предполагающей автоматизированную проверку решения (даже в тестах), можно выделить две основные части. Первая предназначена для представления задачи в понятной человеку форме (например, словесное описание условия задачи), вторая представляет формализованное описание, понятное машине, вычислительной среде или интерпретатору, которые, собственно, и осуществляют проверку решения.

В представлении задачи для человека можно выделить еще один важный момент: возможность параметризации условия, когда по одному такому описанию среда генерирует разные «клоны» исходной задачи. Собственно, эта часть мало зависит от способа формализации решения для компьютера и может состоять из перечисления параметров и их значений. Ниже приведен пример такого описания, использующегося как часть параметрически генерируемой задачи в системе KKK. Описание данной части задачи состоит из описания параметров генерации задачи и из словесного описания задачи с возможностью использовать в нем описанные параметры (листинг 1).

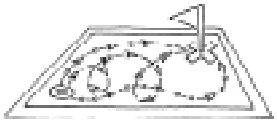
По данному описанию система генерирует четыре однотипных задачи с разными параметрами в условии: два значения для возможной системы счисления (base) и два для возможной длины билета (length). Каждый раз, когда пользователь выбирает данную задачу, система генерирует ее условие со случайным набором параметров (систему можно настроить и таким образом чтобы для каждого пользователя генерировались свои параметры).

При описании задачи также важно обеспечить, чтобы сложность задач, генерируемых в рамках данного описания, была одинаковой, в противном случае пришлось бы связывать с каждым набором параметров соответствующее количество балов, начисляемое за ее успешное решение. В данной системе такой связи не осуществляется, а предполагается, что составитель сам обеспечивает правильную параметризацию условия.

В системе KKK имеется несколько встроенных стратегий проверки – это может быть сравнение с уже имеющимся в описании задачи ответом (этот вариант ближе к традиционному, хотя и здесь спектр

допустимых форматов ввода шире обычного), либо автоматическое построение ответа по описанию задачи с последующим сравнением его с ответом пользователя. Система также поддерживает и возможность добавления новых стратегий. Ниже подробно обсуждаются способы описания задач, принятые в данной системе:

Прежде всего отметим, что даже в случае явного задания правильного ответа система не проверяет ответ пользователя на основе строковой эквивалентности, а рассматривает его в соответствующем формате (число, функция и пр.) и осуществляет проверку на основе сравнения результатов вычислений. При наличии параметров система вычисляет значения выражений на различных наборах параметров и сравнивает их.



ПРОВЕРКА РЕШЕНИЯ ДЛЯ РАЗНЫХ СТРАТЕГИЙ

В системе реализованы две стратегии проверки решения:

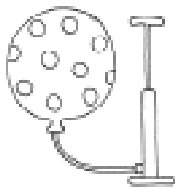
– основанная на идее полного перебора;

– основанная на сравнении с известным ответом.

Первая стратегия перебирает все элементы задаваемого множества и подсчитывает количество элементов, удовлетворяющих некоторому соотношению, после чего сравнивает свой ответ с ответом пользователя. Если задача задается в общем виде, то для нее генерируется несколько подзадач с конкретными наборами параметров, и решение системы сравнивается с решением пользователя вычисленное для данных параметров. Требуемое соотношение, как и само множество, описывается с помощью примитивов системы (языка описания задач).

Вторая стратегия работает проще – она сравнивает ответ пользователя с ответом, заданным составителем, причем в случае постановки задачи в общем виде значения формулы-ответа вычисляются для различных наборов параметров и для каждого из них сравниваются с ответом пользователя, вычисленным на нужном наборе.

ТОЧКИ РАСШИРЕНИЯ



Для добавления новых примитивов, расширяющих покрываемую системой область задач, имеются две точ-

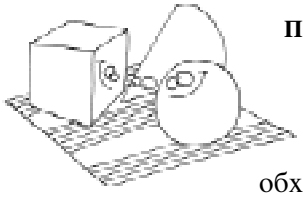
Листинг 1. Возможное описание задачи в формате *xml*, предназначенное для человека.

```
<task title="Счастливые билеты"> // Начало блока описания задачи
 <description-params>             // Блок описания параметров генерации
  <param name="length">           // Описание параметра length
   <value text="двух">4</value>   // Описание одного из значений параметра
   <value text="трех">6</value>
  </param>
  <param name="base">             // Описание параметра base
   <value text="девяти">9</value>
   <value text="семи">7</value>
  </param>
 </description-params>
 <description> // Словесное описание задачи с использованием параметров
  Трамвайный билет называется счастливым по-питерски, если сумма первых
  ${length-text} цифр равна сумме последних ${length-text} цифр.
  Трамвайный билет называется счастливым по-московски, если сумма его цифр,
  стоящих на четных местах, равна сумме цифр, стоящих на нечетных местах.
  Сколько существует билетов, счастливых и по-питерски и по-московски,
  если для записи билета используются цифры от нуля до ${base-text}?
 </description>
 <Формальное описание/>
</task >
```

ки расширения: первая позволят добавлять поддержку новых множеств к системе, вторая – новых функций. Для добавления нового примитива необходимо реализовать соответствующий интерфейс – *Set* для множеств и *Function* для функций. Также предусмотрена точка расширения для стратегий проверки решения – интерфейс *Verifier*.

В системе используется абстрагирование от того, какой природы является элемент множества или параметр функции, в качестве данной абстракции выступает класс *Element*.

Все функции и множества оперируют с данной абстракцией. Такое абстрагирование позволяет сделать систему легко масштабируемой. Стратегия же проверки оперирует абстракциями функций и множеств – интерфейсами *Set* и *Function*.



ПОДДЕРЖИВАЕМЫЕ ПРИМИТИВЫ

В данной системе реализованы необходимые средства описания задач по комбинаторике для поддержки автоматизированной проверки их решений, но эти средства позволяют описывать не только задачи этой конкретной области. В первую очередь, это набор базовых множеств:

– отрезок числовой прямой – натуральные числа от k до n, $k \geq n$;

– декартово произведение множеств;

– множество сочетаний;

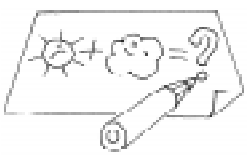
– множество размещений.

Кроме этого, реализован набор необходимых функций, предикатов и операций над элементами данных множеств:

– логические операции;

– арифметические операции;

– функции для работы с картежами элементов и др.



ФОРМАЛЬНОЕ ОПИСАНИЕ ЗАДАЧИ

Формат описания задачи для данной системы предполагает, помимо наличия описания задачи, предназначенного для человека, также наличие формального описания, которое понимает система проверки. Оно состоит из указания стратегии проверки и необходимых для данной стратегии элементов, например для переборной стратегии необходимо описать перебираемое множество и критерий выбора элементов для подсчета, для стратегии сравнения с уже известным ответом – описать ответ. Описание необходимых элементов стратегии осуществляется с помощью примитивов системы – функций и множеств, причем одни и те же примитивы могут использоваться в разных стратегиях.

Например, если мы хотим подсчитать все элементы множества размещений из n элементов по k, у которых первый элемент картежа равен 2 ($2 \leq k \leq n$), то описание множества и описание критерия подсчета будет выглядеть так (см. листинг 2).

Понятно, что для такой простой задачи можно сразу написать ответ $(n-1)!/(n-k)!$ и воспользоваться другой стратегией проверки решения. Но это не всегда легко сделать, а описание ответа посредством описания условия задачи гарантирует его правильность. Заметим, что обучаемый будет вводить ответ в «формульном» виде. Для данного примера это могут быть ответы:

$(n-1)!/(n-k)!$, или $A(n-1; k-1)$,

или $C(n-1; k-1)*(k-1)!$ и т. д.

Как видно из примеров, при описании задачи каждый используемый примитив описывается с помощью отдельного *xml*-элемента, у которого в атрибуте *type* указывается имя нужного примитива, а в остальных атрибутах специализирующие параметры (например атрибут *axis* у примитива *Projection* в примере выше), для функций это *xml*-элемент с именем *function*, а для множества – *set*.

Используемая стратегия задается в элементе *verifier*. В данном случае это переборная стратегия:

```
<verifier type="SimpleVerifier">
```

Если задача задается в общем виде, то необходимо описать значения параметров, на которых будет производиться проверка решения. Описание этих параметров имеет такой же вид, как и описание параметров генерации условия. Для примера, приведен-

Листинг. 2. Пример описания множества размещений с фиксированным первым элементом:

```
<set type="LayoutSet" length="${k}"> // множество размещений
  <set type="NumericSet" first="1" last="${n}" /> // отрезок числовой прямой
</set>
```

и описания критерия подсчета:

```
<function type="Equal">//проверяем на равенство два вложенных элемента
<function type="Projection" axis="1"> // возвращает первый элемент картежа
  <current-set-element />          // текущий элемент множества -
                                   // картеж из k  элементов
</function>
<constElement>2</constElement>    // значение с которым сравниваем
</function>
```

ного выше, оно будет выглядеть так (см. листинг 3).

Структура файла-описания задачи в системе имеет следующий вид (см. листинг 4).

Для обращения к текущему значению параметра (генерации или проверки) используется следующая конструкция – **${имя_параметра}**.

Ниже приведен пример полного описания задачи о счастливых билетах, в котором также предусмотрена генерация условия с различными параметрами (листинг 5).

Рисунок 1. Пользовательский интерфейс решения задачи.

ИНТЕРФЕЙС ОБУЧАЕМОГО (ПОЛЬЗОВАТЕЛЯ)

На рисунке 1 приведен вид пользовательского интерфейса системы, предоставляющего доступ к различным наборам задач (олимпиадам), а также к

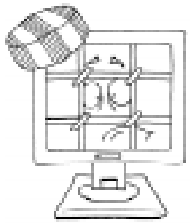
Листинг 3.

```
<verifier-params>  //начало блока описания параметров проверки
<param name="n">   //описание параметра n - размера исходного множества
   <value>10</value> //описание одного из возможных значений параметра
   <value>7</value>
   <value>12</value>
</param>
<param name="k">      //описание параметра k -
                      //количества элементов в размещении
   <value>6</value>  //описание одного из возможных значений параметра
   <value>5</value>
</param>
  </verifier-params>
```

Листинг. 4. Общая структура файла-описания задачи

```
<начало блока описания задачи>
   <начало блока описания параметров генерации условия>
      <начало блока описания параметра a1>
         <описание одного из значений параметра/>
      </конец блока  описания параметра a1>

      <начало блока описания параметра a2>
        ...
      </конец блока  описания параметра a2>
      ...
   </конец блока описания параметров генерации условия>

  <описание задачи предназначенное для человека/>

   <начало блока описания параметров проверки решения>
     //такая же структура, как и для блока параметров генерации условия
   </конец блока описания параметров проверки решения>

  <начало блока описания задачи предназначенного для системы>
     <начало блока описания стратегии проверки>
        //необходимые элементы стратегии
     </конец блока описания стратегии>
   </конец блока описания задачи предназначенного для системы>
</конец блока описания задачи>
```

каждой из задач набора в отдельности. Для ввода ответа предназначена виртуальная клавиатура, содержащая список всех доступных для записи ответа примитивов.



ЗАКЛЮЧЕНИЕ

Описанная в статье система (Конструктор комбинаторных коллекций) представляет собой прототип нового программно-педагогического средства, позволяющего автору ставить новую математическую задачу перед членами некоторого общего информационного пространства. При этом автор может не знать решения поставленной задачи, а система позволяет проверить правильность найденного решения как членами сообщества, так и самим автором. Таким образом, среда KKK дает новые возможности для конструирования общих образовательных информационных пространств.

Листинг. 5. Полное описание задачи на примере задачи о счастливых билетах

```
<?xml version="1.0" encoding="windows-1251" ?>
<task title="Счастливые билеты">
  <description-params>
    <param name="length">
      <value text="двух">4</value>
        <value text="трех">6</value>
    </param>
    <param name="base">
      <value text="девяти">9</value>
      <value text="пяти">5</value>
    </param>
  </description-params>
  <description>
    Трамвайный билет называется счастливым по-питерски, если сумма первых
    ${length-text} цифр равна сумме последних ${length-text} цифр. Трамвайный
    билет называется счастливым по-московски, если сумма его цифр, стоящих на
```

```
    четных местах, равна сумме цифр, стоящих на нечетных местах.
    Сколько существует билетов, счастливых и по-питерски и по-московски,
    если для записи билета используются цифры от нуля до ${base-text}?
  </description>
  <mathDescription>
    <sourceSet>
      <set type="DecartSet">
        <for name="i" first="1" last="${length}" inc="1">
           <set type="NumericSet" first="0" last="${base}" />
        </for>
      </set>
    </sourceSet>
<verifier type="SimpleVerifier">
  <function type="And"> // билет счастливый и по-питерски, и по-московски?
    <function type="Equals">//билет счастливый по-питерски?
      <function type="Add"> //сумма первой половины цифр
        <for name="i" first="1" last="${length}/2" inc="1">
           <function type="Projection" axis="${i}">
             <current-set-element />
           </function>
         </for>
        </function>
         <function type="Add">//сумма второй половины цифр
           <for name="i" first="${length}/2+1" last="${length}" inc="1">
             <function type="Projection" axis="${i}">
                <current-set-element />
             </function>
           </for>
         </function>
        </function>
         <function type="Equals"> //билет счастливый по-московски?
           <function type="Add">//сумма цифр на нечетных местах
             <for name="i" first="1" last="${length}" inc="2">
               <function type="Projection" axis="${i}">
                  <current-set-element />
               </function>
             </for>
           </function>
             <function type="Add"> //сумма цифр на четных местах
               <for name="i" first="2" last="${length}" inc="2">
                 <function type="Projection" axis="${i}">
                    <current-set-element />
                 </function>
               </for>
             </function>
            </function>
           </function>
         </verifier>
    </mathDescription>
</task>
```

Богданов Михаил Сергеевич,
магистр факультета информационных технологий и программирования СПбГУ ИТМО.