Educational Technology
Образователни технологии

XML МОДЕЛ ЗА АВТОМАТИЧНО ГЕНЕРИРАНЕ НА ПРОГРАМНИ ТЕКСТОВЕ

Павел Азълов
Пенсилвански държавен университет

Резюме. Предмет на тази статия е автоматичното генериране на примери, представляващи програмни текстове. Процедурата за автоматично генериране изисква предварително да се разработи програмен шаблон. Той се конструира от програмен текст, съобразен с темата, а също и с целта, за която ще се използва – разглеждането му в час, за самоподготовка или изпит. Представеният модел за автоматично генериране на примери използва XML технология. За целта е веведен нов XML език, наречен xGen. Синтаксисът му е дефиниран чрез езика DTD (Document Type Definition). Езикът xGen позволява да се описват програмни текстове с разнообразна структура. Той е независим от конкретния език за програмиране. За да се илюстрират възможностите на езика xGen, в статията е използван езикът за програмиране C++. Съществена възможност на xGen е автоматично да оценява трудността на генерираните примери. Дефинирано е понятието генетичен код, с помощта на което се описва структурата на всеки от генерираните примери. Чрез генетичия код се вевежда частична наредба в множеството от примери, генерирани в една и съща сесия. По този начин примерите могат да се разделят в отделни подмножества, а всяко от тях да се подреди в ненамаляваща по сложност редица от примери. В края на статията е разгледан пример, илюстриращ цялостния процес на генериране на програмни примери.

Keywords: automatic example generation, sequernce of problems, XML model, example-based learning

1. Въведение

Трудно е да си представим преподаването в уводни курсове по програмиране без използване на примери. Преди много години това най-добре го е изказал Алберт Айнщаин: "*Example isn't another way to teach, it is the only way to teach*"[1]. Примерите са подходящи за разглеждане в час, за самоподготовка на обучаемите, а могат и да се формулират като въпроси в изпитни теми.

От много публикации се вижда, че преподавателите използват примери за въвеждане на нови понятия и методи в програмирането. Примерите са в основата и на всяка тема в учебниците. Съществуват и софтуерни системи, които подпомагат разработването на примери за целите на обучението. Най-често примерите в про-

грамирането представляват програмни фрагменти, отделни функции или относително кратки програми. Чрез анализа на такива програмни текстове лесно се обясняват синтактичните и семантичните характеристики на управляващите структури и типовете данни в съответните езици за програмиране. Обучемите също по-лесно възприемат изучавания материал, когато е представен чрез примери.

1.1. Примери с еднаква и с различна структура

Основни в тази статия са понятията *пример* и *шаблон*.

Примерът е текст, написан на конкретен език за програмиране. Създаден е, за да се илюстрира ново понятие. Той може да се използва като задача при самообучение или като въпрос от изпитна тема.

Шаблонът е текст, с който се описват структурата и съдържанието на генерираните примери. Всеки шаблон може да се разглежда като *конструктор* на примери от един и същ тип.

Примерите в езиците за програмиране могат да варират от отделен фрагмент до пълна програма, състояща се от отделни функции. На фиг. 1 е даден пример, с който частично се илюстрира синтаксисът на оператор за избор в езика C++. Този пример може да се формулира и като въпрос „*Какъв ще е резултатът от изпълнение на програмния фрагмент от фиг.* 1?“

```
int n = 5;
if (n > 0)
   cout << 1;
else
{
   cout << 2;
   cout << 3;
   cout << 4;
}
```

Фигура 1. Програмен фрагмент, илюстриращ оператор за избор

Не е трудно да се създадат и много други програмни фрагменти със същата структура и съдържание, при това автоматично. Тази възможност е илюстрирана с таблицата от фиг. 2. Лявата колонка представлява шаблон, от който са генерирани два примера (екземпляра от същия тип) с идентична структура и подобно съдържание. Резултатите от изпълнението на програмните фрагменти във втората и третата колонка зависят от конкретната стойност на псевдопроменливата (параметър) <randInt>, въведена в шаблона. Нейните стойности са случайни числа, които се определят по време на генерирането на примерите.

int n = <randInt>;
if (n > 0)
  cout << 1;
else
{
  cout << 2;
  cout << 3;
  cout << 4;
}
	int n = -3;
	if (n > 0)
	  cout << 1;
	else
	{
	  cout << 2;
	  cout << 3;
	  cout << 4;
	}
		int n = 7;
		if (n > 0)
		  cout << 1;
		else
		{
		  cout << 2;
		  cout << 3;
		  cout << 4;
		}

Отговор: ?	*Отговор*: 234	*Отговор*: 1

Фигура 2. Илюстрация на шаблон (първа колонка) и два примера, които могат да се генерират от него

Съществено е да се спомене, че автоматично генерираните текстове могат да бъдат с различна структура. Идеята за това е дадена на фиг. 3. Примерите в двете колонки на таблицата са създадени с помощта на един и същ шаблон. Както се вижда, двата програмни текста са не само различни по съдържание, но имат напълно различна структура.

```
void main()
{
  int a = 3;
  int result = a;


  cout << result << endl;
}
```

```
void p(int&);
void main()
{
  int a = 7;
  int result = a;
  p(a);
  result += a;
  cout << result << endl;
}

void p(int& m)
{
 m = 6 + m;
 }
```

Фигура 3. Примери с различно съдържание и структура, генерирани от един и същ шаблон

1.2. Кратък литературен обзор

Системите, подпомагащи обучението в началните курсове по програмиране, могат да се класифицират в две основни групи: системи за анализ и системи за синтез.

А. Системите за анализ изискват от обучаемия да напише програмен текст като отговор на даден въпрос. Системата анализира въведения текст и извършва проверка за верността му. WebToTeach (Arnow, D. & O. Barshay, 1999) е добре позната система от този тип. Сега тя се предлага под новото име CodeLab като уеб базирана интерактивна система за анализ и изпълнение на програми. Използва се в уводни класове, в които се преподават езици за програмиране като Java, C++, C, Python и други.

В. Системите за синтез изискват от обучаемите „ръчно да изпълнят“ програмен текст или да посочат синтактичните грешки, ако има такива. Програмните текстове (примерите) могат да се генерират (синтезират) предварително или по време на текущата сесия от работата на системата.

По-долу следва кратък преглед на някои основни резултати, отнасящи се до системите за синтез.

1.2.1. Системи, поддържащи предварително създадени примери

Системата CFX (Reed, D., S. John, R. Aviles & F. Hsu, 2004) предоставя интерфейс, с който могат да се създават примери и да се извършва достъп до тях. По време на час примерите се използват от преподавателя, а обучаемите работят с тях по време на самоподготовка. Добавянето на нови примери към базата от данни е възможно, включително и добавянето на примери от използваните учебници. След като базата от примери е създадена, в нея може да се извършва контекстуално търсене с помощта на ключови думи.

Най-използвани са системите, в които примерите са формулирани като въпроси (Traynor D. & J. Gibson, 2005). При съставяне на конкретна изпитна тема от базата с данни на случаен принцип се извличат въпроси от съответната тема. Всеки обучаем от дадена група получава индивидуална тема, която по обем и тежест е равностойна с темите на останалите обучаеми от групата. При необходимост тестът може да бъде повторен, без да има повторение на примерите. Отговорите на въпросите с многовариантен избор също могат да се променят, както и редът, в който са записани.

1.2.2. Системи, използващи шаблони

Има множество публикации, описващи системи, които генерират примери на базата на шаблони. Следва кратко представяне на две от тях.

Водеща идея в системата QuizPACK (Pathak, S. & P. Brusilovsky, 2002) е създаването на параметризиран програмен код (шаблон), в който се въвежда псевдопроменлива. На фиг. 4 тя е означена с буквата Z. Заместването ѝ с подходящи, но случайно избрани стойности води до генерирането на екземпляри (примери) на шаблона. Два такива примера са представени във втората и третата колонка на

таблицата от фиг. 4. Те се различават единствено по стойността на псевдопроменливата Z. Нека да отбележим, че екземплярите на шаблона от фиг. 4 винаги ще имат една и съща структура.

Параметризиран код (шаблон)	Пример, в който Z е заместена с числото 16	Пример, в който Z е заместена с числото 11

int main ()
 	
{
  int i = 0;
  int s = **$Z;**
  for (; i < s; i++)
  s = s - i;
}
	int main ()
	
	{
	  int i = 0;
	  int s = **16**;
	  for (; i < s; i++)
	  s = s - i;
	}
		int main ()
		
		{
		  int i = 0;
		  int s = **11**;
		  for (; i < s; i++)
		  s = s - i;
		}

Каква е стойността на s?	*Каква е стойността на s?*	*Каква е стойността на s?*

Фигура 4. Примери, генерирани от системата QuizPack

В (Shah, H. & A. Kumar, 2002) е описана друга идея за конструиране на шаблон. Използван е език, сходен с метаезика Бакус-Наур. Нетерминалните символи в шаблона са имена на променливи {<V1>, <V2>, ... }, стойности на случани числа {(<R1>, <R2>, ...}, имена на променливи указатели {<P1>, <P2>, ...}, типове данни {<T1>, <T2>, ...; където <T0> е типът void}, имена на функции {<F1>, <F2>, ...; където <F0> представлява главната функция main} и други. Терминалните символи са думи и знаци като например if, else, while, do, for, [ ] се използва за индекси на масив, ( ) – за списък от параметри на функция, { } – за програмен блок, и други. Един шаблон и два негови екземпляра са дадени на фиг. 5.

<T0><F0>()	void main()	void main()

{
  <T1#integer#><P1>;
  <T1><V1>=<R1#1<=R1<=9;#>;
  {
  <T1><V2>=<R2#10<=R2<=19;#>;
  <P1>= &<V2>;
  }
  << <V1>;
  << <P1>;
}
	{
	  int *ptr;
	  int num = 3;
	  {
	  int count = 12;
	  ptr = &count;
	  }
	  cout << num;
	  cout << *ptr;
	}
		{
		  int *p;
		  int n = 8;
		  {
		  int c = 17;
		  p = &c;
		  }
		  cout << n;
		  cout << *p;
		}

Фигура 5. Шаблон (първа колонка) и два екземпляра, генерирани от него

Стойностите, които константите R1 and R2 могат да получат, са определени съответно с интервалите [1 .. 9] и [10 .. 19]. Не е трудно да се забележи, че екземплярите на шаблона от фиг. 5, макар и различни, винаги ще имат една и съща структура.

Идеята за автоматичното генериране на примери, представена в (Azalov, P. & F. Zlatarova, 2003), е близка с тази, описана в настоящата статия. Различията се отнасят до езика за описания на шаблони, както и до възможността за структуриране на множеството от примери в редици с ненамаляваща сложност.

2. Един нов XML език за описание на шаблон

По-долу се въвежда нов XML език, наречен xGen (E**x**ample **Gen**eration). За описание на синтаксиса му е използван езикът DTD (Document Type Definition). В този смисъл може да се каже, че всеки шаблон е XML документ. Езикът xGen е предназначен за създаване (конструиране) на шаблони. Системата, която на базата на такъв шаблон генерира примери, също е наречена xGen.

Генерирането на примери се извършва в три стъпки:

– [*Начален програмен текст*] Най-напред се създава т. нар. начален програмен текст (програмен фрагмент, функция, или малка програма, съдържаща фунции). Да го означим с p. Този текст е „ядрото“ на примерите, които ще бъдат генерирани. Съдържанието и структурата му зависят от темата и понятията, които преподавателят желае да илюстрира чрез примерите.
– [*Създаване на шаблон*] Със средствата на езика xGen и началния програмен текст p се създава шаблон, който по-нататък ще бъде означаван с Tp.
– [*Генериране на примери*] Като използва шаблона Tp, системата xGen генерира необходимия брой примери.

Основните характеристики на езика xGen могат да се резюмират по следния начин:

– [XML *език*] Езикът xGen е XML език. Неговият синтаксис е определен чрез езика DTD.
– [*Независимост*] Езикът xGen не зависи от езика за програмиране, на който се генерират примерите (C, C++, Java, Pithon, SQL, ...).
– [*Управляващи структури*] В езика xGen има управляващи структури, с които началният програмент текст може да се трансформира в множество от примери, имащи подобна и/или различна структура.
– [*Стойности по подразбиране*] Много от атрибутите на елементите на езика xGen имат стойности по подразбиране, които са интуитивно разбираеми.
– [*Оценяване на генерираните примери*] В езика xGen има средства, с които при определени условия може да се оцени относителната сложност на

един пример спрямо друг пример. По този начин множеството на генерираните примери от един и същ шаблон в рамките на една сесия на системата xGen може да се „структурира“ в редици от примери с ненамаляваща сложност.

2.1. Синтаксис на xGen шаблон

Структурата на един xGen шаблон се определя от три XML елемента (секции): identification, definition и generation.

```
<!ELEMENT xGen (identification, definition, generation)>
<!ATTLIST xGen source CDATA #IMPLIED>
```

Всеки от тези три елемента е представен по-долу.

2.1.1. Елемент identification

Елементът identification се използва за описание на характеристиките на всеки отделен шаблон. Състои се от следните поделементи: problem, title, topic, author, date, instruction, figure и comment. Ето и пълната дефиниция на елемента identification.

```
<!ELEMENT identification (problem, title, topic*, author*, date?, instruction, figure*,
comment?)>
<!ELEMENT problem (#PCDATA)>
<!ATTLIST   problem  id CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT topic (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT instruction (#PCDATA)>
<!ELEMENT figure (#PCDATA)>
<!ATTLIST   figure src CDATA #REQUIRED>
<!ELEMENT comment (#PCDATA)>
```

Всеки шаблон е еднозначно определен от атрибута id на елемента problem. Предназначението на генерираните примери се определя с елемента instruction. Уточняването на предназначението на примерите може да се извърши чрез елемента figure, като за целта се добавят подходящи фигури, таблици или графики. Смисълът на останалите елементи (title, topic, author, date, и comment) се подразбира от съответните им имена.

Пример

```
<identification>
  <problem id=“123“/>
  <title>Introduction to C++</title>
```

```
<topic>Functions</topic>
<topic>Value and Reference Parameters</topic>
<author>Dr. Andrew Robertson </author>
<date>10/01/2014</date>
<instruction>What is the output of the following program?</instruction>
</identification>
```

2.1.2. Елемент definition

Понятията *системна константа* и *системна променлива* играят важна роля при конструирането на всеки шаблон. Те са поделементи на елемента definition, чийто синтаксис е определен по следния начин:

```
<!ELEMENT definition (#PCDATA | const | oper | var)*>
```

При дефинирането на име, което е системна константа, трябва да се посочи съответният ѝ тип данни и интервалът от възможните ѝ стойности. Ако типът данни е string, тогава интервалът от стойности се интерпретира като интервал на допустимата дължина на низа. Ето и пълната дефиниция на елемента const:

```
<!ELEMENT const EMPTY>
<!ATTLIST const  id CDATA #REQUIRED>
<!ATTLIST const  datatype (bool | char | short | int | long | float | double | string) „int“>
<!ATTLIST const  value CDATA #IMPLIED>
<!ATTLIST const  min CDATA #IMPLIED>
<!ATTLIST const  max CDATA #IMPLIED>
```

Примери

```
<const id=“r1“ min=“0“ max=“9“/>
<const id=“r2“ min=“1“ max=“6“/>
<const id=“Iter“ value=“2“/>
<const id=“Loop“ min=“1“ max=“3“/>
<const id=“bp“ datatype=“bool“/>
<const id=“bq“ datatype=“bool“/>
<const id=“b“ datatype=“bool“/>
```

От дефинициите на системните константи r1, r2 и Loop се вижда, че те ще получат стойности съответно в интервалите [0 .. 9], [1 .. 6] и [1 .. 3]. Стойностите на последните три константи bp, bq и b ще бъдат три случайно генерирани логически стойности. Стойността на константата Iter е директно посочена и тя е 2. Всичките шест константи се използват в някои от следващите примери.

Системните променливи се дефинират чрез елемента var. Те имат три атрибута id, datatype и value. С тях се указват съответно името, типът данни и началната стойност на всяка променлива. По-долу следва пълната дефиниция на понятието системна променлива.

```
<!ELEMENT var EMPTY>
<!ATTLIST var  id CDATA #REQUIRED>
<!ATTLIST var  datatype (bool | char | short | int | long | float | double) „int“>
<!ATTLIST var  value CDATA #IMPLIED>
```

Началната стойност на системната променлива може да се запише като израз, съдържащ, предварително дефинирани системни константи.

Примери

```
<var id=“v1“ value=“r1 + r2“/>
<var id=“v2“/>
```

Типът данни и стойността на системната променлива v2 не са указани. По подразбиране типът данни ще е int, а стойността ще бъде случайно число.

С елемента oper (операция) се дефинира системна променлива, чиято стойност е операция от някакъв тип данни. По подразбиране операцията е бинарна, а типът данни е int. Ето и пълната дефиниция на елемента oper:

```
<!ELEMENT oper EMPTY>
<!ATTLIST oper  id CDATA #REQUIRED>
<!ATTLIST oper  datatype (bool | char | short | int | long | float | double) „int“>
<!ATTLIST oper  arg (1 | 2) „2“>
```

Пример

```
<oper id=“ar“/>
```

Името на дефинираната операция в примера е ar. Тя ще бъде случайно избрана бинарна аритметична операция от типа int.

Атрибутите на елементите, дефинирани в секция definition, се използват в изрази, дефиниращи стойностите на атрибутите в секция generation.

2.1.3. Елемент generation

Процедурата, с която се генерират примери по даден xGen шаблон, се описва чрез елемента generation. Той има следните поделементи: set, valueOf, del, if, repeat и weight.

```
<!ELEMENT generation (#PCDATA | set | valueOf | del | if | repeat | weight)*>
```

Съгласно процедурата за генериране на примери, описана в началото на раздел 2, първата стъпка е да се напише началният програмен текст, който се явява яд-

рото на съдържанието на елемента generation. Следва *параметризиране* на текста чрез вмъкване на подходящи елементи от езика xGen. По този начин се определя необходимата трансформация на първичния текст, за да се получат отделните примери. Ако даден ред от началния програмен текст не съдържа xGen елементи, то този ред ще остане непроменен. Ако в някой ред от началния програмен текст има вмъкнати xGen елементи, тогава текстът в него ще се промени съгласно семантиката на съответните xGen елементи.

По-долу са приведени някои от основните семантични характеристики на поделементите на елемента generation.

- Елементът set се използва за присвояване на стойност на системна променлива. Той не оказва директно влияние върху генерирания пример. Нормално е да се запише в самостоятелен ред на шаблона, т.е. да не се вмъква в ред от програмния текст.
- Елементът valueOf се използва за „четене“ на стойността системна променлива. Записва се в рамките на съответния ред от началния програмен текст.
- Елементът weight се използва за пресмятане на теглото (трудността) на ред от генерирания текст. Той не влияе върху процеса на генериране на примери.
- Елементът del се използва за изтриване (премахване) на ред или на част от ред на началния програмен текст.
- Елементите valueOf и del въздействат само на един ред от началния програмен текст. Могат да се прилагат многократно в рамките на един ред.
- Елементите if and repeat са опростени варианти на операторите за избор и цикъл. Могат да се прилагат върху един или върху няколко реда от началния програмен текст.

Елементите valueOf, set и weight не променят структурата на началния програмен текст и затова ще бъдат наречени *статични елементи* в xGen.

С елементите del, if и repeat може да се променя структурата на началния програмен текст чрез изтриване на редове (с del и if) или чрез вмъкване на редове (чрез repeat). Това е причината по-нататък тези елементи да се наричат *динамични елементи* в xGen.

Елементът set. С елемента set се присвоява стойност на системна променлива. Името и стойността ѝ се указват чрез атрубутите id и value.

```
<!ELEMENT set EMPTY>
<!ATTLIST set  id CDATA #REQUIRED>
 <!ATTLIST set  value CDATA #REQUIRED>
```

В най-простия случай стойността, която се присвоява на системна променлива, е системна константа. Нейният тип данни би трябвало да е съвместим с този на сис-

темната променлива. В общия случай стойността на системната променлива може да е израз, съдържащ преди това дефинирани системни променливи и константи.

Пример

```
<set  id="v3" value="(r2 + v1)%6 + 1"/>
```

Елементът valueOf. Достъп до стойността на една системна константа или променлива може да се извърши чрез елемента valueOf. Неговата дефиниция е следната:

```
<!ELEMENT valueOf EMPTY>
<!ATTLIST valueOf id CDATA #REQUIRED>
```

„Изпълнението" (интерпретацията) на елемента по време на генерирането на примери се състои в замяна на самия елемент със съответната стойност на системната константа или променлива, посочена с атрибута id.

Пример

```
m = <valueOf id="r2"/><valueOf  id="ar"/> m;
```

Ако текущите стойности на r2 и ar са съответно цялото число 3 и оперцията за умножение „*", системата ще генерира текста: m = 3*m;

Елементът del. Този елемент се използва са изтриване на ред или на част от ред, принадлежащ на началния програмен текст. „Изпълнението" на елемента зависи от логическо условие, записано като стойност на атрибута cond. Съответният знаков низ (ред, част от него) се изтрива само ако стойността на условието е true. Началната позиция, от която започва изтриването, се определя от позицията на елемента del в рамките на реда от програмния текст. Дължината на низа, подлежащ на изтриване, се определя от атрибута length. Ако елементът del се намира в началото на реда и не е посочена дължината на низа за изтриване, тогава се изтрива целият ред. В случай, че в един ред са вмъкнати няколко del елемента, интерпретацията им се извършва от ляво надясно.

Следва синтаксисът на елемента del.

```
<!ELEMENT del EMPTY>
<!ATTLIST del cond CDATA #REQUIRED>
<!ATTLIST del length CDATA #IMPLIED>
```

Примери

```
<del  cond="bp"/>void p(int<del  cond="b" length ="5"/>&amp;);
<del  cond="bq"/>int q(int);
```

В първия от по-горните примери елементът del присъства два пъти. При „изпълнението" му са възможни следните случаи:

- Ако bp = true, тогава целият ред ще бъде изтрит.
- Ако bp = false и b = false, тогава нищо няма да се изтрие от реда и системата ще генерира текста: void p(int&);
- Ако bp = false and b = true, тогава ще се генерира текстът void p(int).

Елементът if. Елементът if е едновариантен оператор за избор. Неговото тяло ще бъде „избрано“ и включено в генерирания пример или няма да бъде „избрано“ в зависимост от логическото условие, което е стойност на атрибута cond. Условието е израз, който може да включва системни константи и променливи. Тялото на if елемента може да съдържа set, valueOf и del елементи. В някои случаи един del елемент може да бъде заменен с if елемент. Ето и пълната дефиниция на елемента if.

```
<!ELEMENT if (#PCDATA | set | valueOf | del | weight)*>
<!ATTLIST if cond CDATA #REQUIRED>
```

Пример

```
<if cond="bq">
int q(int m)
{
  return <valueOf id="r2"/><valueOf  id="ar"/> m;
}
</if>
```

Функцията q ще бъде включена в текущия генериран елемент само ако стойността на bq е true.

Тук ще отбележим, че при „изпълнението“ на елементите del и if е възможно да се изтрие един знак, низ от знаци или цял ред от началния програмен текст. По този начин съдържанието на началния програмен текст „намалява“. В този смисъл може да се приеме, че генерираният пример ще съдържа не по-сложен (не по-труден) за анализиране (разбиране) текст, отколкото началния програмен текст.

Елементът repeat. Елементът repeat е оператор за цикъл с управляваща променлива. Броят на итерациите се определя от стойността на атрибута value, чиято стойност се посочва от системна константа. Неговият синтаксис е следният:

```
<!ELEMENT repeat (#PCDATA | set | valueOf | del | weight)*>
<!ATTLIST repeat  value CDATA #REQUIRED>
```

Пример

```
<const id="Iter" value="2"/>
<repeat value="Iter">
<del cond="bp"/>f(a);
<del cond="bp"/>r += a;
<del cond="bq"/>r += g(a);
</repeat>
```

Тялото на цикъла от примера ще се изпълни два пъти. Ако стойностите на bp и bq са false, ще се генерира следният текст:

```
f(a);
```

```
r += a;
r += g(a);
f(a);
r += a;
r += g(a);
```

Ако стойността на Iter е равна на 1, а стойностите на bp и на bq са съответно true и false, в текущо генерирания пример ще присъстват само два реда:

```
f(a);
r += a;
```

Елементът weight. Елементът weight се използва, за да се укаже тежестта (трудността) на даден ред от началния програмен текст, ако този ред бъде включен в текущия пример. Елементът weight се записва в края на реда или самостоятелно на отделен ред. Общото тегло на всеки генериран пример е сумата от теглата на отделните редове. Теглото на всеки ред по подразбиране е 1, а на празния ред е 0. Общото тегло се използва при търсене на примери от даден интервал. Ето и пълната дефиниция на елемента weight.

```
<!ELEMENT weight EMPTY>
<!ATTLIST weight value CDATA #REQUIRED>
<!ATTLIST weight cond CDATA #IMPLIED>
```

Атрибутът value е от реален тип данни. Елементът се изпълнява само ако стойността на атрибута cond е true.

Примери

```
a += 3 + b++; <weight value="1.5"/>
<del cond="bp"/>f(a); <weight value="3.5" cond="!bp"/>
```

В първия ред на горния пример общото тегло ще нарасне с 1.5. Теглото на втория ред зависи от стойността на логическата константа bp.

– Ако bp = true, вторият ред ще бъде изтрит.
– Ако bp = false, низът „f(a);“ от втория ред ще бъде включен в генерирания пример. Понеже стойността на !bp е true, текущото тегло на примера ще нарасне с 3.5.

3. Структуриране на множеството от генерирани примери

Нека p е произволен начален програмен фрагмент, а Tp е шаблон, конструиран от него. Означаваме с S(Tp) множеството на примерите, които могат да се генерират чрез шаблона Tp. Шаблонът, както и всички примери, генерирани от него, са текстови файлове и се състоят от отделни редове. Някои от тях могат да са празни, а други могат да съдържат един или няколко оператора от съответния език за програмиране. Празните редове не оказват влияние на съдържанието и структурата на примерите.

Определение 1 [*Дължина на пример*] Дължината на пример x от множеството S(Tp) се определя от броя на непразните редове в x и ще бъде означавана с |x|.

Ако шаблонът Tp съдържа динамични xGen елементи, в множеството S(Tp) ще има примери с различни дължини, включително и такива, които са с минимална и максимална дължина. Да отбележим, че примерите с максимална (минимална) дължина не са непремено идентични. Те имат един и същ брой редове, но някои от редовете могат да се разливат по съдържанието си. Най-простият случай, който може да се има предвид, е този, в който се използват системни константи със случайно генерирани стойности.

Всеки пример с максимална дължина съдържа:

– всички редове на началния програмен текст, евентуално с променено съдържание на някои отделни редове;
– максималния брой редове, които се генерират от елемента repeat.

Определение 2 [*Дължина на шаблон*] Дължината |Tp| на произволен шаблон Tp се определя от максималната дължината на пример, който може да се генерира от него, т.е. |Tp| = max { |x| | x ∈ S(Tp) }.

За удобство ще въведем понятието *фиктивен ред*, който е празен ред, но е маркиран по някакъв начин, за да се различава от останалите празни редове. За разлика от празния ред фиктивният ред участва в пресмятане на дължината на пример.

Определение 3 [*Максимално разширение на пример*] Един пример x*, x* ∈ S(Tp) е максимално разширение на даден пример x, x ∈ S(Tp), ако:

(1) е с максимална дължина, т.е. |x*| = |Tp|;
(2) всеки ред от първичния програмен текст p, който е включен в x, е включен и в x* (евентуално променен), а редовете от p, които не са включени в x, са включени в x* като фиктивни редове;
(3) всички добавени редове с елемента repeat са включени като фиктивни редове в x*.

3.1. Генетичен код на пример

Определение 4 [*Генетичен код на шаблон*] Функцията gc_{Tp}, наречена *генетичен код* на шаблона Tp, съпоставя на всеки пример x, x ∈ S(Tp) двоичен вектор с |Tp| компоненти по следния начин:

gc_{Tp}: S(Tp) → {0,1}$^{|Tp|}$

gc_{Tp} (x) = < x*(1), x*(2), . . . , x*(|Tp|)>, където x* е максималното разширение на x.

Компонентата x*(*k*) се нарича *k*-ти *ген* на примера x и се дефинира по следния начин:

$$x^*(k) = \begin{cases} 0, \text{ ако } k\text{-ят ред на } x^* \text{ е фиктивен ред;} \\ 1, \text{ ако } k\text{-ят ред на } x^* \text{ не е фиктивен ред.} \end{cases}$$

По-нататък се предполага, че шаблонът е произволен, но фиксиран, и за удобство функцията генетичен код ще се записва само с gc.

Определение 5 [*Релация на доминиране*] Нека x и y са два примера от $S(Tp)$. Генетичният код на y *доминира* генетичния код на x, ако $x^*(k) \leq y^*(k)$ за $k = 1, 2, ..., |Tp|$ и това ще се записва като: $gc(x) \preccurlyeq gc(y)$.

Релацията на доминиране „$\preccurlyeq$“ има две важни свойства:

– тя е рефлексивна, антисиметрична и транзитивна, т.е. тя дефинира *частична наредба* в множеството $S(Tp)$;
– ако $x \in S(Tp)$, $y \in S(Tp)$, и $gc(x) \preccurlyeq gc(y)$, тогава съществува шаблон Ty, за който $x \in S(Ty)$.

Първото свойство се проверява непосредствено. Доказателството на второто свойство се свежда до параметризирането на примера y в шаблон Ty, в който чрез вмъкване на del и/или if елементи няма да се допусне в генерираните примери да запишат редове от y, които не са редове от x.

Второто свойство на релацията доминиране води към следното важно заключение.

Ако $gc(x) \preccurlyeq gc(y)$, $x \in S(Tp)$ и $y \in S(Tp)$, тогава програмният текст в пример x ще бъде част от програмния текст на пример y. Това дава основание да се приеме, че анализът на пример x не би трябвало да е по-сложен (по-труден) от този на пример y.

3.2. Сравними и подобни примери

Определение 6 [*Сравними примери*] Два примера x и y, $x \in S(Tp)$ и $y \in S(Tp)$ са *сравними*, ако $gc(x) \preccurlyeq gc(y)$ или $gc(y) \preccurlyeq gc(x)$.

Означаваме с $S^*(Tp)$ множеството на генерираните примери в рамките на една сесия на системата xGen. Нека s е едно подмножество на $S^*(Tp)$ – такова, че всички примери в него са сравними помежду си. Понеже s е крайно множество с частична наредба, то в него има минимален елемент. Това е достатъчно условие, за да се приложи топологическа сортировка на примерите от s (Rossen, K. 2007). По този начин множеството s ще бъде *линеаризирано*, започвайки от *най-лесния* (един от тях) и завършвайки с *най-трудния* (един от тях) пример. Описаната процедура може да се приложи към всяко подмножество на $S^*(Tp)$, в което примерите са сравними. Така множеството на всички генерирани примери $S^*(Tp)$ ще се декомпозира в списъци от примери, подредени в ненамаляваща трудност.

Определение 7 [*Релация на структурно подобие*] Два примера x и y, генерирани от един и същ шаблон, са *подобни* ($x \approx y$), ако имат един и същ генетичен код, т.е. $gc(x) = gc(y)$.

Релацията на структурно подобие има две важни свойства:

– Тя е рефлексивна, симетрична и транзитивна, т.е. тя е релация на *еквивалентност*.
– Ако един шаблон Tp е конструиран само със статични елементи, тогава за всеки два примера x и y, x ∈ *S*(Tp) и y ∈ *S*(Tp) е в сила x ≈ y.

Връщайки се към разгледаните в раздел 1.2 системи, може да се каже, че те генерират само подобни примери, тъй като шаблоните им допускат само статични елементи. Използването на подобни примери е подходящо в случаи на подготовка на писмени теми (тестове) за групово изпитване. Генерирането на примери с различна структура и линеаризирането им в списъци с ненамаляваща сложност са подходящи за самоподготовка на обучаемите.

4. Представяне на цялостен пример „C++ функции и параметри“

По-долу следва цялостен пример, представящ цялостната процедура за генериране на примери. Най-напред ще запишем първите две секции на шаблона, в който не се използват текстове на началния програмен текст.

```
<identification>
  <problem id=“123“/>
  <title>C++ Functions</title>
  <topic>Parameters</topic>
  <author>Dr. Andrew Richardson</author>
  <date>07/01/2011</date>
  <instruction>What is the output of the following program?</instruction>
</identification>
<definition>
  <const id=“r1“ min=“0“ max=“9“/>
  <const id=“r2“ min=“1“ max=“6“/>
  <const id=“r3“ min=“1“ max=“15“/>
  <const id=“Loop“ min=“1“ max=“3“/>
  <const id=“bp“ datatype=“bool“/>
  <const id=“bq“ datatype=“bool“/>
  <const id=“b“ datatype=“bool“/>
  <oper id=“ar1“/>
  <oper id=“ar2“/>
</definition>
```

В двуколонната таблица от фиг. 8 са записани началният програмен текст и секцията generation на шаблона. Координацията на редовете в двете колонки е извършена за по-лесното разбиране на начина, по който е конструиран шаблонът.

The Initial Program Text Written in C++	The *generation* Section (part of the xGen template)

void f(int&);
int g(int);
void main()
{
  int a = 3;
  int r = a;
 	
  f(a);
  r += a;
  r += g(a);
 	
  cout << r;
}
 	
void f(int& m)
{
  m = 2 + m;
 	
}
 	
int g(int m)
{
  return 2 + m;
}
	<generation>
	<del cond="bp"/>void f(int<del cond="b" length ="5"/>&);
	<del cond="bq"/>int g(int);
	void main()
	{
	  int a = <valueOf id="r1"/>;
	  int r = a + <valueOf id="r3"/>;
	  <repeat value="Loop">
	  <del cond="bp"/>f(a); <weight value="1.5" cond="!bp"/>
	  <del cond="bp"/>r += a; <weight value="3.5" cond="!bp"/>
	  <del cond="bq"/>r += g(a);
	  <weight value="2.5" cond="!bq"/>
	  </repeat>
	  cout << r;
	}
	
	<if cond="bp">
	void f(int<del cond="b" length ="5"/>& m)
	{
	  m = <valueOf id="r2"/><valueOf id="ar1"/> m;
	  <weight value="2.5" cond="!b"/>
	}
	</if>
	<if cond="bq">
	int g(int m)
	{
	  return <valueOf id="r2"/><valueOf id="ar2"/> m;
	 }
	</if>
	</generation>

Фигура 8. Начален програмен текст и секция generation на примерен шаблон

Системните логически константи bp и bq са свързани с включването или невключването на прототипите на функциите f и g, на техните дефиниции и на обръщенията към тях. Ясно е, че ако прототипът на една функция не е включен в даден пример, то не трябва да включва и дефиницията на функцията и естествено не може да има и обръщение към нея. Подобна е и ролята на системната логи-

ческа константа b. Ако b = false, тогава параметърът m на функцията f ще бъде деклариран като параметър псевдоним, а в противен случай знакът "&" ще бъде изтрит и параметърът ще се предава като стойност. Трите логически константи са използвани и за допълнително определяне на теглото на някои оператори като стойности на атрибута cond в елемента weight.

На фиг. 9 са представени пет примера, генерирани от шаблона от по-горе.

Пример e1	Пример e2	Пример e3	Пример e4	Пример e5

void main()
{
int a = 3;
int r = a + 2;
 	
cout << r;
}
	void f(int&);
	
	void main()
	{
	int a = 7;
	int r = a + 12;
	
	f(a);
	r += a;
	f(a);
	r += a;
	f(a);
	r += a;
	
	cout << r;
	}
	
	void f(int& m)
	{
	m = 6 + m;
	}
		void f(int);
		int g(int);
		void main()
		{
		int a = 4;
		int r = a + 6;
		
		f(a);
		r += a;
		r += g(a);
		
		cout << r;
		}
		
		void f(int m)
		{
		m = 3 * m;
		}
		
		int g(int m)
		{
		return 3 - m;
		}
			01 void f(int&);
			02 int g(int);
			03 void main()
			04 {
			05 int a = 0;
			06 int r = a + 9;
			
			07 f(a);
			08 r += a;
			09 r += g(a);
			10 f(a);
			11 r += a;
			12 r += g(a);
			13 f(a);
			14 r += a;
			15 r += g(a);
			16 cout << r;
			17 }
			
			18 void f(int& m)
			19 {
			20 m = 5 * m;
			21 }
			
			22 int g(int m)
			23 {
			24 return 5 / m;
			25 }
				int g(int);
				
				void main()
				{
				int a = 5;
				int r = a + 14;
				
				r += g(a);
				
				r += g(a);
				
				cout << r;
				}
				
				int g(int m)
				{
				return 2 * m;
				}

Фигура 9. Пет примера, генерирани от шаблона от фиг. 8

За по-лесното разглеждане и съпоставяне на отделните примери редовете на пример 4 са номерирани, а редовете на останалите примери са синхронизирани с тях. В таблицата на фиг. 10 са записани стойностите на системните константи и променливи, дължините и общото тегло на всеки от петте примера.

Пример e1	Пример e2	Пример e3	Пример e4	Пример e5
r1 = 3 r2 = ? r3 = 2 Loop = ? bp = true bq = true b = ? ar1 = ? ar2 = ? |e(1)| = 6 weight = 6.0	r1 = 7 r2 = 6 r3 = 12 Loop = 3 bp = false bq = true b = false ar1 = „+“ ar2 = ? |e(2)| = 17 weight = 26.5	r1 = 4 r2 = 3 r3 = 6 Loop = 1 bp = false bq = false b = true ar1 = „*“ ar2 = „-“ |e(3)| = 19 weight = 21.5	r1 = 0 r2 = 5 r3 = 9 Loop = 3 bp = false bq = false b = false ar1 = „*“ ar2 = „*“ |e(4)| = 25 weight = 37.5	r1 = 5 r2 = 2 r3 = 14 Loop = 2 bp = true bq = false b = ? ar1 = ? ar2 = „*“ |e(5)| = 13 weight = 16.0

Фигура 10. Параметри на петте генерирани примера

По-долу са записани генетичните кодове на петте примера и релацията на доминиране.

gc(e1) = (0,0,1,1,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0)
gc(e2) = (1,0,1,1,1,1,1,1,0,1,1,0,1,1,0,1,1,1,1,1,1,0,0,0,0)
gc(e3) = (1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1)
gc(e4) = (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
gc(e5) = (1,0,1,1,1,1,0,1,0,0,0,1,0,0,0,1,1,0,0,0,0,0,1,1,1,1)
gc(e1) ≼ gc(e2), gc(e1) ≼ gc(e3), gc(e1) ≼ gc(e4), gc(e1) ≼ gc(e5)
gc(e2) ≼ gc(e4)
gc(e3) ≼ gc(e4)
gc(e5) ≼ gc(e4)

Редиците от примери с ненамаляваща сложност, които могат да се формират от тези примери, са следните: (e1, e2, e4), (e1, e3, e4) и (e1, e5, e4). Примерите e2, e3 и e5 са несравними помежду си.

5. Заключение

В натоящата статия с представен формален модел за генериране на примери (програмни текстове) на базата на предварително конструиран шаблон. Съществуващите подходи, с които се решава подобна задача, позволяват да се генерират само статични програмни текстове, т.е. такива, които се различават единствено по стойностите на някои константи. Съществената разлика в предлагания подход е генерирането на програмни текстове, които се различават не само по съдържание, но и по структура. Въвеждането на генетичен код на пример е от съществено

значение за оценката на отделните примери и структурирането им в редици с ненамаляваща трудност.

Описаният в статията език xGen е XML език и може лесно да бъде разширяван. Едно съществено разширение на езика би могло да бъде в посока на автоматичното генериране на резултатите от изпълнението на генерираните примери.

БЕЛЕЖКИ

1. Обучението с примери не е другият, а единственият начин на обучение.

REFERENCES / ЛИТЕРАТУРА

Arnow, *D. & O. Barshay* (1999). WebToTeach: An Interactive Focused Programming Exercise System. *Proceedings of FIE '99* (San Juan, Puerto Rico, November 1999), IEEE Press.

Azalov, P. & F. Zlatarova (2003). SDG – A System for Synthetic Data Generation. P*roceedings of the International Conference on IT: Coding and Computing*, IEEE Computer Society Press, Volume 1, pp. 69 – 75.

Pathak, S. & P. Brusilovsky (2002). Assessing Student Programming Knowledge with Web-based Dynamic Parameterized Quizzes, Barker, P. and S. Rebelsky (eds.). *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, pp. 1548 – 1553.

Reed, D., S. John, R. Aviles & F. Hsu (2004). CFX: Finding Just the Right Examples for CS1, *SIGCSE'04*, March 3 – 7, Norfolk, Virginia, USA.

Rossen, K. (2007). Discrete Mathematics and Its Applications, sixth edition, McGraw Hill.

Shah, H. & A. Kumar. (2002). A Tutoring System for Parameter Passing in Programming *Languages*, *The Seventh Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE* 2002*)*, Aarhus, Denmark, 6/24 – 26/2002.

Traynor D. & J. Gibson. (2005). Synthesis and Analysis of Automatic Assessment Methods in CS1. Generating intelligent MCQs. *SIGCSE'05*, February 23.27, 2005, St. Louis, Missouri, USA, pp. 495 – 499.

AN XML MODEL FOR AUTOMATIC GENERATION OF PROGRAM TEXTS

Abstract. This paper discusses the automatic generation of examples, which are program texts. The procedure for the automatic example generation requires a program template to be created in advance. This template should be based on a specific program text corresponding to the topic in consideration and to the objectives of the teaching

process. The generated examples could be used in class, for self-preparation, or in a variety of assignments. The proposed model for automatic example generation was developed by implementing an XML-related technology. A new XML language, xGen, was introduces. Its syntax was defined by using the DTD (Document Type Definition) language. The xGen language allows the description of program texts of diverse structures and is independent from the specific programming language of the program texts. In this paper, the possibilities of the xGen language are illustrated by using the C++ programming language. The possibility for complexity assessment of the generated examples represents an essential xGen feature. The notion of genetic code was also defined to be used when describing the structure of every generated example. The genetic code allows introducing a partial order relation in the set of the examples that are generated during the same computer session. In this way, the examples could be distributed in separate subsets. Each of these subsets is ordered in ascending order as a series of examples starting with the easiest example and ending with the most difficult example. A case study illustrating the overall process for automatic example generation is also presented in the end of the paper.

✉ **Dr. Pavel Azalov, Assoc. Prof.**
Pennsylvania State University
Hazleton Campus, U.S.A.
E-mail: pka10@psu.edu