Научная статья

УДК 004.434
DOI 10.25205/1818-7900-2022-20-1-18-27

Предметно-ориентированный язык для генерации заданий из исходных текстов программ

Игорь Андреевич Жуков [1]
Юрий Леонидович Костюк [2]

[1, 2] Национальный исследовательский Томский государственный университет
Томск, Россия

[1] IgZhukov963@yandex.ru, https://orcid.org/0000-0001-7995-3688
[2] kostyuk_y_l@sibmail.com, https://orcid.org/0000-0002-8914-8161

Аннотация

Обсуждаются подходы автоматизированной генерации заданий для различных дисциплин, отмечено, что для программирования параметрический способ не может быть применен. Авторы развивают идею применения конструктивно-выборочного метода для генерации заданий по программированию. В конструктивно-выборочном методе задание дополняется набором компонентов, из которых обучающийся составляет свой ответ (программу). Применение этого метода позволяет разнообразить задания по программированию, создает новый способ автоматизированного контроля на основе содержания ответа. При этом правильные ответы учитываются как полностью, так и частично. Проиллюстрированы на примерах основные понятия модели представления задания по программированию. Предложен предметно-ориентированный язык разметки и его транслятор, позволяющий генерировать задания из исходных текстов программ преподавателя. Приведен пример применения транслятора.

Ключевые слова

конструктивно-выборочный метод, контроль знаний по программированию, автоматизированный контроль знаний, подготовка заданий, предметно-ориентированный язык

Для цитирования

Жуков И. А., Костюк Ю. Л. Предметно-ориентированный язык для генерации заданий из исходных текстов программ // Вестник НГУ. Серия: Информационные технологии. 2022. Т. 20, № 1. С. 18–27. DOI 10.25205/1818-7900-2022-20-1-18-27

A Domain-Specific Language for generating tasks from Programs Source Code

Igor A. Zhukov [1], **Yuriy L. Kostyuk** [2]

[1, 2] National Research Tomsk State University
Tomsk, Russian Federation

[1] Ig.Zhukov963@yandex.ru, https://orcid.org/0000-0001-7995-3688
[2] kostyuk_y_l@sibmail.com, https://orcid.org/0000-0002-8914-8161

Abstract

Approaches to task generation for various disciplines are discussed, mentioned that a parameter method cannot be applied to programming. The authors developed an application of the idea of a constructive-selective method for creating programming tasks. A task created for constructive-selective methodical contains a set of components, from which

a student makes up his answer (a program). Application of this method allows diversifying of tasks in programming and creates a new way of assessment based on the content of the answer. It's also allows to grade both fully and partially correct answers. The main concepts of the model for representing programming tasks are illustrated by examples. The domain-specific markup language and its translator that allows generating tasks from a teacher's program source text are proposed. An example of a translator usage is given.

Keywords

constructive-selective method, programming knowledge assessment, automated knowledge assessment, task preparation, domain-specific language

For citation

Zhukov I. A., Kostyuk Yu. L. A Domain-Specific Language for generating tasks from Programs Source Code. *Vestnik NSU. Series: Information Technologies*, 2022, vol. 20, no. 1, p. 18–27. (in Russ.) DOI 10.25205/1818-7900-2022-20-1-18-27

Введение

Актуальной проблемой для системы образования является подготовка контрольно-измерительных материалов по различным дисциплинам. При этом особое место занимает процесс формирования заданий с применением технических средств. Анализ источников показал, что проблему рассматривают для различных технических и математических дисциплин. В работе [1] проанализированы типовые элементы деталей, используемых в заданиях по теме «Разрезы и сечения» в рамках дисциплины «Инженерная графика», и разработан алгоритм генерации новых заданий при помощи заранее созданных элементов. Каждое сгенерированное задание имеет решение. Сообщается о возможности получить 10^8 различных уникальных вариантов заданий. В [2] представлен генератор заданий по теме «Импеданс» для подготовки инженеров-электротехников. Обучающемуся предлагается электрическая схема со случайно заданными параметрами: сопротивления резисторов, индуктивность катушки и емкость конденсатора. Генератор проводит расчет, позволяющий отобрать корректные задания. Также предлагается программное обеспечение для представления обучающимся задач и проверки правильности их решений.

Способ генерации заданий, при котором формулируют типовые задачи с параметрами, значения которых подбираются с помощью генератора случайных чисел, в дальнейшем будем называть параметрическим. Этот способ распространен в математических дисциплинах. Следует отметить, что подбор числовых параметров может быть случайным выбором текстового элемента из заранее определенного списка. Авторы [3] используют математический пакет MATLAB для подготовки заданий по теме «Собственные числа и собственные векторы линейного оператора» в рамках дисциплины «Линейная алгебра». Средствами MATLAB выполняются генерация и отбор матриц, имеющих целые собственные числа. В работе [4] описана разработка системы для генерации заданий по теме «Линейное программирование». При генерации задания выбирается четыре случайных числа из заранее заданного диапазона. Эти значения подставляют в шаблон, затем происходит проверка наличия решения задачи с найденными параметрами. Также эта система реализует возможность проверки решений обучающихся. В [5] описаны правила алгоритма генерации для семи типовых задач по дисциплине «Численные методы». Используются как числовые параметры, так и список некоторых основных элементарных функций. Этот список позволяет, например, формировать трансцендентные уравнения для выдачи заданий обучающимся. В правила алгоритма генерации входит проверка корректности задачи. Также представлена программная реализация этого алгоритма. Автор [6] разработал формальный язык для описания генераторов заданий на основе деревьев. В узлах дерева содержатся данные (числа и строки текста), которые могут войти в условие формируемой задачи. Узел и его потомки могут быть связаны конструкциями «И» и «ИЛИ». По умолчанию используется конструкция «ИЛИ», при этом только один из потомков войдет в задачу. Если используется конструкция «И», то все потомки узла будут использованы. В качестве примера рассмотрена задача по теории вероятностей.

Практически во всех рассмотренных источниках для формирования заданий используется параметрический способ. Разнообразие заданий достигается за счет изменения числовых параметров при стандартизованном условии задачи. Обучающийся должен применить алгоритм решения типовой задачи к конкретному набору данных. Для дисциплин, связанных с программированием, параметрический способ в литературе не обсуждается. Это объяснимо, поскольку обучающийся, решая задачу по программированию, должен написать программу, используя стандартный или разработанный самостоятельно алгоритм. Параметрический способ вступает в противоречие со свойством массовости алгоритма. Согласно этому свойству алгоритм должен решать задачу для наборов данных из некоторой области, например, для всех целых чисел. Для генерации математической задачи можно взять квадратный трехчлен и подбирать его коэффициенты. Каждый набор коэффициентов определяет отдельную задачу – квадратное уравнение, которое можно дать обучающемуся. Для программирования можно в этом случае поставить только одну задачу – написать программу решения квадратного уравнения.

Авторы статьи для отработки навыков по программированию предлагают использовать конструктивно-выборочный метод. Задание реализовать конкретный алгоритм дополняется набором конструкций языка программирования, из которых обучающийся составляет программу. В этом случае при подготовке задания потребуется подобрать соответствующий набор конструкций. Преподаватель, как правило, уже имеет программы для всех предлагаемых заданий. В статье описаны предметно-ориентированный язык разметки и его транслятор, позволяющий перевести программный код в набор конструкций для выдачи задания. Кроме того, разработанный транслятор подготавливает задание для автоматизированного контроля.

Основные положения модели представления задания

Команды языка разметки опираются на понятия компонента, элемента и шаблона, введенные в [7]. Поясним эти понятия на примерах. Рассмотрим задание: вычислить сумму целых чисел от 1 до *n* включительно. Допустим, преподаватель работает с первым вариантом программы:

```
readln(n);
S:=0;
for i:=1 to n do
   S:=S+i;
writeln(S);
```

Компонент – это пара из натурального числа и строки программного кода, например, инициализация переменной, заголовок цикла. Компоненты для рассматриваемого примера будут следующими:

```
1. readln(n);          4. S:=S+i;
2. S:=0;               5. writeln(S);
3. for i:=1 to n do
```

Составить правильную программу из этих компонентов можно двумя способами: первую и вторую строки программы можно поменять местами. Первые два компонента образуют элемент типа «Перестановка компонентов», а остальные элементы будут иметь тип «Один компонент». При выдаче задания для обучающегося номера компонентов не указаны. Порядок компонентов можно изменить. Для данного примера получается 5!=120 возможных вариантов выдачи задания (118, если исключать варианты, в которых порядок компонентов соответствует правильным программам).

Преподаватель может предложить второй вариант для формирования задания – программу с использованием цикла while вместо for:

```
readln(n);
S:=0;
i:=1;
while(i<=n) do
begin
   S:=S+i;
   i:=i+1;
end;
writeln(S);
```

Обучающемуся будет предложено 9 компонентов (9!=362880 возможных вариантов выдачи задания). Составить правильную программу из этих компонентов можно шестью способами за счет перестановки первых трех строк. В этом примере первые три компонента образуют элемент типа «Перестановка компонентов», а остальные элементы будут иметь тип «Один компонент». Увеличение значения итератора цикла i может быть сделано альтернативным способом при помощи встроенной функции inc. Таким образом, получим третий вариант программы. Для этого следует добавить компонент со строкой кода «inc(i);», который в паре с компонентом «i:=i+1;» образует элемент с типом «Один из компонентов». Обучающемуся в таком случае будет предложено 10 компонентов (10!=3628800 возможных вариантов выдачи задания). Составить правильную программу из этих компонентов можно двенадцатью способами за счет перестановки первых трех строк и выбора между вариантами увеличения значения итератора. Для усложнения задания преподаватель может добавлять «шумовые» компоненты, которые не входят ни в одно верное решение.

При формировании задания можно одновременно использовать все описанные выше программы. В общем случае для одного языка программирования существует множество программ, которые решают поставленную задачу. В дальнейшем это множество будем называть U-шаблоном по аналогии с универсальным множеством U. Отметим, что программы, отличающиеся только идентификаторами (имена переменных и функций), не будем считать различными. Отличиями являются последовательность компонентов (если их порядок не влияет на правильность ответа и корректность программы), незначимые синонимы, значимые синонимы. В первом варианте программы предполагается перестановка компонентов, которая не влияет на правильность ответа (корректность программы). Третий вариант отличается от второго наличием незначимого синонима (способом увеличения значения итератора). Второй вариант отличается от первого наличием значимого синонима – использованием цикла while вместо for. Наличие значимых синонимов соответствует частям разбиения U-шаблона. В дальнейшем каждую часть разбиения будем называть шаблоном. Три программы, рассмотренные выше, не представляют U-шаблон полностью, поскольку возможны иные варианты написания цикла. На основе приведенных программ будет создано два шаблона. Для первого варианта программы преподавателя может быть создан один шаблон. А второй и третий варианты программы могут быть описаны другим шаблоном. Для записи шаблонов используется формальный язык, подробно описанный в [7].

Степень совпадения ответа обучающегося шаблону определяет контролирующий алгоритм. Частичное совпадение может считаться частично правильным ответом. Для преподавателя при подготовке шаблона предусмотрена возможность дать рекомендации по классификации вероятных ошибок как критических и некритических. Например, из предложенных компонентов составлена программа:

```
readln(n);
for i:=1 to n do
   S:=S+i;
writeln(S);
```

Эта программа похожа на первый вариант программы преподавателя, за исключением инициализации значения переменной S. Эту ошибку можно считать некритической. Оценка такого ответа будет незначительно снижена.

Рассмотрим другой пример ответа обучающегося:

```
readln(n);
S:=0;
S:=S+i;
writeln(S);
```

Эта программа также похожа на первый вариант программы преподавателя, но в ней отсутствует цикл for. Эта ошибка критическая, такая программа не решает поставленную задачу.

Для классификации критических и некритических ошибок в модели предусмотрены метки элементов. Метка может быть одного из двух типов: «допускается отсутствие» для некритических ошибок и «рубежный» для критических. При отсутствии меток значимость ошибок не различается.

Язык разметки

Для того чтобы воспользоваться транслятором, необходимо разметить файлы исходного кода программ, затем отправить размеченный файл на вход транслятора. Выходом будет набор компонентов и шаблон. Язык разработан так, чтобы управляющие команды начинались с символов строчного комментария в языке программирования. Таким образом, размеченная программа остается корректной с точки зрения языка программирования.

Определение шаблона имеет вид *//$ pattern (компонент | управляющая_команда) {(компонент | управляющая_команда)} //$ end*. Определение может начинаться в любом месте текстового файла. Текст вне определения игнорируется. Текстовый файл может содержать более одного шаблона. Приведем пример определения шаблона:

```
//$ pattern
readln(n);
S:=0;
for i:=1 to n
S:=S+1;
//$ end
```

Внутри определения шаблона могут быть строки и управляющие команды. Каждая строка текста является компонентом, который нумеруется автоматически. Управляющие команды делятся на 2 типа: однострочные и многострочные. Однострочная команда языка разметки находится перед строкой (строками) программы, на которые она действует. Многострочные команды имеют начало и конец, а все затронутые строки программы находятся в теле команды. Управляющие команды начинаются с символа *//!*. Элементы могут быть помечены: метка «допускается отсутствие» имеет значение *optional* (без учета регистра), метка «рубежный» имеет значение *important* (без учета регистра). Все строки внутри шаблона, не затронутые какой-либо командой, считают определением элемента типа «один компонент» без метки. Для определения элемента «один компонент» с меткой требуется однострочная команда, которая действует на следующую строку и имеет вид *//! метка*. Приведем пример:

```
//$ pattern
//! optional
S:=0;
for i:=1 to n
//$ end
```

Для задания элементов типа «один из компонентов» и «перестановка компонентов» можно использовать как однострочные, так и многострочные управляющие команды. Элементы типа «один из компонентов» определяется при помощи ключевого слова *oneof*. Однострочная команда имеет вид *//! oneof количество_строк l [метка]*. Количество строк указывает, на сколько последующих строк действует эта команда. Многострочная команда имеет вид *//! oneof [метка] один_или_больше_компонентов //! end.* Все компоненты, находящиеся внутри многострочной команды, относятся к определяемому элементу. Например:

```
//$ pattern
//! oneof 2l optional
i:=i + 1;
inc(i);
until i > n;
//$ end
```

```
//$ pattern
//! oneof
i:=i + 1;
inc(i);
//! end
end;
//$ end
```

Элементы типа «перестановка компонентов» определяются при помощи ключевого слова *permutation*. Однострочная команда имеет вид *//! permutation количество_строк l [метка_для_перестановки]*. Многострочная команда имеет вид *//! permutation [метка_для_перестановки] один_или_больше_компонентов //! end.* Приведем пример:

```
//$ pattern
//! permutation 2l
S:=0;
i:=1;
while i <= n do
//$ end
```

```
//$ pattern
//! permutation optional
writeln(f);
writeln(r);
//! end
//$ end
```

В соответствии со стандартом [1] запишем все порождающие правила в РБНФ:

команда = ("//" | "#") "!";

компонент = любой_символ {любой_символ};

один_или_больше_компонентов = компонент {компонент};

количество_строк = число "l";

метка = "optional" | "important";

метка_для_перестановки = "optional";

команда_завершения = команда "end";

начало_шаблона = ("//" | "#") "$" "pattern";

конец_шаблона = ("//" | "#") "$" "end";

обычная_метка = команда метка;

начало_один_из = команда "oneof" [метка];

многострочная_команда_один_из = команда количество_строк "oneof" [метка];

начало_перестановки = команда "permutation" [метка_для_перестановки];

многострочная_команда_перестановка = команда количество_строк "permutation" [метка_для_перестановки];

определение_один_из = начало_один_из один_или_больше_компонентов команда_завершения;

многострочное_определение_один_из = многострочная_команда_один_из один_или_больше_компонентов;

определение_перестановки = начало_перестановки один_или_больше_компонентов команда_завершения;

[1] ИСО/МЭК 14977:1996 Информационная технология. Синтаксический метаязык. Расширенная форма Бэкуса – Наура.

многострочное_определение_перестановки = многострочная_команда_перестановка один_или_больше_компонентов;

помеченый_элемент = обычная_метка компонент;

содержимое_шаблона = (компонент | управляющая_команда) {(компонент | управляющая_команда)};

управляющая_команда = определение_перестановки | определение_один_из | многострочное_определение_один_из | многострочное_определение_перестановки | помеченый_элемент;

определение_шаблона = начало_шаблона содержимое_шаблона конец_шаблона;

исходный_файл = (компонент | определение_шаблона) {(компонент | определение_шаблона)};

Грамматика оптимизирована для разбора алгоритмом синтаксического анализа LALR (Look-Ahead Left-to-Right), описанного в [8].

Пример использования

Возьмем пример из дисциплины «Основы программирования». Задания по этой дисциплине должны контролировать степень освоения определенного набора базовых алгоритмов и умение реализовать эти алгоритмы. Рассмотрим пример 2.3 из [9], который описывает рекуррентный алгоритм для приближенного вычисления значения функции $\sin x$. Приведем пример кода программы, которая решает эту задачу:

```
S := x;
f := x;
k := 2;
while abs(f) >= eps do
begin
   f := -f * x * x / ((2 * k – 1) * (2 * k – 2));
   S := S + f;
   k := k + 1;
end;
```

Для корректности программы неважен порядок инициализации переменных S, f, k, а также порядок изменения значений переменных S и k в теле цикла while. С формальной точки зрения это будут различные решения. Размеченный код программы может иметь следующий вид:

```
//$ pattern
   //! permutation
   S := x;
   f := x;
   k := 2;
   //! end
   //! important
   while abs(f) >= eps do
   begin
      //! important
      f := -f * x * x / ((2 * k – 1) * (2 * k –
2));
      //! 2l permutation
      S := S + f;
      k := k + 1;
   end;
//$ end
```

После трансляции из размеченной программы будут получены список компонентов (табл. 1) и шаблон, имеющий следующий вид:

{(1;2;3);4*;5;6*;(8;7);9}.

Все элементы получившегося шаблона приведены в табл. 2. Шаблону соответствует 3! * 2!=12 программ. Последовательности компонентов, описывающие эти программы, приведены в табл. 3.

Таблица 1

Список компонентов

Table 1

List of components

1. S := x;	4. while abs(f) >= eps do	7. S := S + f;
2. f := x;	5. begin	8. k := k + 1;
3. k := 2;	6. f := -f * x * x / ((2 * k – 1) * (2 * k – 2));	9. end;

Таблица 2

Элементы шаблона

Table 2

Elements of the pattern

Тип элемента	Метка	Компоненты, входящие в элемент
Перестановка компонентов	Без метки	1, 2, 3
Один компонент	Рубежный	4
Один компонент	Без метки	5
Один компонент	Рубежный	6
Перестановка компонентов	Без метки	7, 8
Один компонент	Без метки	9

Таблица 3

Полностью правильные последовательности компонентов

Table 3

Fully correct sequences of components

1;2;3;4;5;6;8;7;9	1;2;3;4;5;6;7;8;9	1;3;2;4;5;6;8;7;9
1;3;2;4;5;6;7;8;9	2;1;3;4;5;6;8;7;9	2;1;3;4;5;6;7;8;9
2;3;1;4;5;6;8;7;9	2;3;1;4;5;6;7;8;9	3;1;2;4;5;6;8;7;9

В размеченном файле компонент может встречаться многократно. Например, операторные скобки «begin» и «end». Когда в рамках трансляции одного размеченного файла компонент встретится впервые, ему будет присвоен номер. Если компонент встречается в файле многократно, то в шаблоне каждый раз будет использован изначально присвоенный номер.

Заключение

Применение конструктивно-выборочного метода дает возможность разнообразить задания по программированию. Перед тем как самостоятельно создавать программу от начала до конца, обучающиеся могут поработать с упрощенными заданиями. На основе конструктивно-выборочного метода предложен новый способ автоматизированного контроля по содержа-

нию текста программ, сконструированных обучающимися. При этом могут быть учтены частично правильные ответы. Разработанный авторами статьи инструмент позволяет без значительных трудозатрат перевести уже имеющиеся исходные тексты программ в задания, готовые для автоматизированного контроля знаний и выдачи обучающимся.

Список литературы

1. **Касаткина Е. П., Хесина Е. А., Чахеев Е. Я., Суховерхий В. А.** Генератор заданий по инженерной графике // Информатизация инженерного образования: Тр. Междунар. науч.-практ. конф. ИНФОРИНО-2016, Москва, 12–13 апреля 2016 года. М.: ИД МЭИ, 2016. С. 142–145.
2. **Jörg Vollrath**. An open access minimum automatic task generation live feedback system for electrical engineering. In: 2015 IEEE Global Engineering Education Conference (EDUCON 2015). (18–20 March 2015, Tallinn University of Technology, Tallinn, Estonia). IEEE, 2015, pp. 494–498. DOI 10.1109/EDUCON.2015.7096015
3. **Власова Е. А., Попов В. С., Пугачев О. В.** Создание фонда оценочных средств и новых образовательных технологий с использованием MATLAB при изучении линейной алгебры // Вестник Моск. гос. обл. ун-та. Серия: Физика-математика. 2016. № 4. С. 77–85.
4. **Хабибулина Н. Ю., Афанасьева Е. И.** Использование генератора индивидуальных заданий при разработке автоматизированной обучающей системы «Линейное программирование» // Электронные средства и системы управления. Материалы докладов Международной научно-практической конференции. 2011. № 1. С. 230–234.
5. **Обади А. А.** Разработка алгоритма и программного обеспечения для генератора задач тестовой системы // Вестник Технологического университета. 2019. Т. 22, № 1. С. 106–111.
6. **Зорин Ю. А.** Интерпретатор языка построения генераторов тестовых заданий на основе деревьев И/ИЛИ // Докл. Том. гос. ун-та систем управления и радиоэлектроники. 2011. № 1 (27). С. 75–79.
7. **Жуков И. А., Костюк Ю. Л.** Модель представления многовариантных заданий для автоматизированного контроля знаний по программированию // Вестник Том. гос. ун-та. Управление, вычислительная техника и информатика. 2020. № 53. С. 110–117. DOI 10.17223/19988605/53/11.
8. **Ахо А., Лам М., Сети Р.** Компиляторы. Принципы, технологии и инструментарий: Пер. с англ. 2-е изд. М.: И.Д. Вильямс, 2008. 1184 с.
9. **Костюк Ю. Л.** Лекции по основам программирования: Учеб. пособие. Томск: Изд-во ТГУ, 2019. 260 с.

References

1. **Kasatkina E. P., Khesina E. A., Chakheev E. Ya., Sukhoverkhy V. A.** Assignments generator on engineering graphics. In: International Conference on Information Technologies in Engineering Education (Inforino). Moscow, 2016, p 142–145. (in Russ.)
2. **Jörg Vollrath**. An open access minimum automatic task generation live feedback system for electrical engineering. In: 2015 IEEE Global Engineering Education Conference (EDUCON 2015). (18–20 March 2015, Tallinn University of Technology, Tallinn, Estonia). IEEE, 2015, pp. 494–498. DOI 10.1109/EDUCON.2015.7096015
3. **Vlasova E., Popov V., Pugachev O.** Creation of a fund of assessment tools and new educational technologies with the use of MATLAB in the study of linear algebra. *Bulletin of Moscow Region State University. Series: Physics and Mathematics*, 2016, no. 4. pp. 77–85. (in Russ.)
4. **Habibulina N. Yu., Afanaseva E. I.** Ispol'zovanie generatora individual'nyh zadanij pri razrabotke avtomatizirovannoj obuchajushhej sistemy “Linejnoe programmirovanie” [Usage of

the generator of individual tasks in the development of an automated lerning system “Linear programming”]. *Elektronnye sredstva i sistemy upravlenija. Materialy dokladov mezhdunarodnoy nauchno-prakticheskoy konferentsii*, 2011, no. 1, pp. 230–234. (in Russ.)
5. **Obadi A. A.** Development of an algorithm and software for a test system tasks generator. *Bulletin of the Technological University*, 2019, vol. 22, no. 1, pp. 106–111. (in Russ.)
6. **Zorin Yu. A.** The interpreter of programming language for design generators of tests based on AND/OR trees. *Proceedings of TUSUR University*, 2011, no. 1 (27), pp. 75–79. (in Russ.)
7. **Zhukov I. A., Kostyuk Yu. L.** Model of representation of multivariate tasks for automated control of programming knowledge. *Tomsk State University Journal of Control and Computer Science*, 2020, no. 53, pp. 110–117 (in Russ.) DOI 10.17223/19988605/53/11
8. **Aho A., Lam M., Sethi R., Ullman J.** Compilers: Principles, Techniques, and Tools. Pearson Education, Inc., 2006, 1010 p.
9. **Kostyuk Yu. L.** Lektsii po osnovam programmirovaniya [Lectures on the Fundamentals of Programming]. A Study Guide. Tomsk, TSU Press, 2019, 260 p. (in Russ.)

Информация об авторах

Игорь Андреевич Жуков, аспирант
Юрий Леонидович Костюк, доктор технических наук, профессор

Information about the Authors

Igor A. Zhukov, Post-Graduate Student
Yuriy L. Kostyuk, Doctor of Technical Sciences, Professor

Статья поступила в редакцию 01.12.2021;
одобрена после рецензирования 01.02.2022; *принята к публикации 01.02.2022*
The article was submitted 01.12.2021;
approved after reviewing 01.02.2022; *accepted for publication 01.02.2022*