ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

Научная статья

УДК 004.4
DOI: 10.18101/2304-5728-2021-2-17-27

РЕАЛИЗАЦИЯ АЛГОРИТМА ТЕСТИРОВАНИЯ КОНТЕКСТНО-СВОБОДНОЙ ГРАММАТИКИ НА ПРИНАДЛЕЖНОСТЬ К КЛАССУ *LL*(*K*)

© Федорченко Людмила Николаевна
кандидат технических наук, старший научный сотрудник,
Санкт-Петербургский федеральный исследовательский центр
Российской академии наук
Россия, 199178, г. Санкт-Петербург, 14 Линия В.О., 39
LNF@iias.spb.su

Аннотация. В статье рассматривается алгоритм тестирования КС-грамматики в форме Бэкуса — Наура на принадлежность ее к подклассу *LL*(*k*) грамматик. Это наибольший «естественный» класс левоанализируемых грамматик, в которых левосторонний анализ может быть реализован детерминированным образом. Они допускают построение левостороннего вывода входной цепочки языка с использованием знания лишь о *k* впереди идущих символах. Такой анализ, в свою очередь, дает детерминированный метод определения выхода правильной трансляции. Рассматривается реализация основных свойств таких грамматик, как FIRST-FOLLOW-Sigma и алгоритм тестирования. В качестве языка разработки был выбран язык C#. Приложение реализовано на платформе .NET Core 3.1, позволяющей создавать программы для различных операционных систем.
Тестирование кода осуществляется при помощи библиотеки XUnit, которая является одним из наиболее популярных решений для тестирования на платформе .NET.
Ключевые слова: контекстно-свободные грамматики; форма Бэкуса — Наура; алгоритм тестирования; *LL*(*k*) грамматики; функции FIRST, FOLLOW, Sigma.

Благодарности. Работа выполнена при участии студента математико-механического факультета СПбГУ Паршина Максима Алексеевича.

Для цитирования
Федорченко Л. Н. Реализация алгоритма тестирования контекстно-свободной грамматики на принадлежность классу *LL*(*k*) // Вестник Бурятского государственного университета. Математика, информатика. 2021. № 2. С. 17–27.

Введение

Одними из наиболее интересных, изучаемых и применяемых на практике свойств формальных грамматик являются те свойства, которые позволяют использовать эффективные методы трансляции языков. Среди

грамматик с такими свойствами выделяется подкласс контекстно-свободных грамматик — *LL*(k)-грамматики, наибольший класс левоанализируемых грамматик [1]. Они допускают построение левостороннего вывода входной цепочки языка с использованием знания лишь о k впереди идущих символах. Такой анализ, в свою очередь, дает детерминированный метод определения выхода правильной трансляции [2].

Таким образом, тестирование грамматики на принадлежность к классу *LL*(k) — это важная задача, решение которой позволяет облегчить построение транслятора.

Цель данной работы — реализовать алгоритм тестирования грамматики на принадлежность к классу *LL*.

1 Постановка задачи

Для достижения цели работы были сформулированы следующие задачи:
– провести обзор алгоритма тестирования КС-грамматики на принадлежность классу *LL*(k)-грамматик, приведенного в [1] и [2];
– реализовать алгоритм;
– реализовать пользовательский интерфейс;
– провести тестирование и отладку алгоритма.

2 Обзор

Определения

Введем определения основных понятий, используемых при описании алгоритма.

Контекстно-свободная грамматика — четверка $G = (V_N, V_T, P, S)$, где V_N, V_T — непересекающиеся алфавиты соответственно **нетерминалов** и **терминалов**, P — конечное множество правил вида $\alpha \to \beta$, где α — нетерминал, а β — цепочка символов; S — начальный нетерминал.

Функция FIRST(k) — для цепочки символов α

$$FIRST_k(\alpha) = \{w \in V_T^* : |w| < k \text{ и } \alpha \dot{\Rightarrow} w \text{ или } |w| = k | \text{ и } \alpha \dot{\Rightarrow} wx; x \in V_T^*\}$$

***LL*(k) грамматика** — контекстно-свободная грамматика, такая, что

$$FIRST_k(\beta\alpha) \cap FIRST_k(\gamma\alpha) = \varnothing$$

для всех α, β, γ, таких, что существуют правила $A \to \beta$, $A \to \gamma$, $\beta \neq \gamma$ и существует левосторонний вывод $S \dot{\Rightarrow} wA\alpha$.

Операция $\oplus$ — для алфавита Σ и цепочек символов $L_1, L_1 \subset \Sigma^*$

$$L_1 \oplus_k L_1 = \left\{ w \in \Sigma^{*k} \mid x \in L_1, y \in L_2, w = \begin{cases} xy, & \text{если } |xy| \le k \\ FIRST_k(xy), & \text{иначе} \end{cases} \right\}$$

Функция $\sigma(A)$ — для нетерминала A

$\sigma(A)=\{ L \subseteq V_T^{*k} : \exists$ левосторонний $S \stackrel{*}{\Rightarrow} wA\alpha,\ w \in V_T^*,\ L = FIRST_k(A)\}$

Функция FOLLOW(k) — для цепочки символов β

$$FOLLOW_k(\beta) = \{w \in V_T^* : S \stackrel{*}{\Rightarrow} \gamma\beta\alpha, w \in FIRST_k(\alpha)\}.$$

3 Алгоритм тестирования *LL(k)*-грамматик

Алгоритм получает на вход контекстно-свободную грамматику и возвращает «да», если грамматика является *LL(k)*-грамматикой для целого *k*, и «нет» — иначе.

1. Вычислить функцию $\sigma(A)$ для всех нетерминалов, для которых существует более одного альтернативных правила.
2. Пусть $A \to \beta$ и $A \to \gamma$ — различные альтернативные правила. Для каждого элемента $L \in \sigma(A)$ вычислить
$$f(L) = (FIRST_k(\beta) \oplus_k L) \cap (FIRST_k(\gamma) \oplus_k L).$$
3. Если $f(L) \neq \varnothing$, то завершить алгоритм с результатом «нет».
4. Повторять предыдущие шаги для всех нетерминалов грамматики.
5. Завершить алгоритм с результатом «да».

Алгоритм вычисления функции FIRST(k)

Для реализации описанного алгоритма необходимо сначала реализовать алгоритмы вычисления используемых в нем функций FIRST и $\sigma(A)$.

Алгоритм вычисления функции FIRST(*k*) получает на вход грамматику и цепочку символов $\beta = X_1X_2...X_n$ из ее алфавита и возвращает множество $FIRST_k(\beta)$.

Поскольку (лемма 2.1 [2])

$$FIRST_k(\beta) = FIRST_k(X_1) \oplus_k FIRST_k(X_2) \oplus_k \ \ldots \ \oplus_k FIRST_k(X_n),$$

достаточно вычислить значение функции для отдельных символов. При этом для терминалов и пустого предложения

$$FIRST_k(X) = \{X\}.$$

Алгоритм основан на построении последовательных приближений функций $F_i(X)$ для всех символов *X* грамматики.

1. $F_i(a) = \{a\}$ для всех терминалов и любого *i*.
2. $F_0(A) = \{ x \in V_T^{*k} : \exists\ A \to x\alpha \in P$, где $|x| = k$, или $|x| < k$ и $\alpha = \varepsilon \}$
3. Если $F_0(A) \ldots F_{i-1}(A)$ уже построены для всех нетерминалов, то
$$F_i(A) = F_{i-1}(A) \cup \{x \in V_T^{*k} : A \to Y_1...Y_m \in P, x \in F_{i-1}(Y_1) \oplus_k ... \oplus_k F_{i-1}(Y_m)\}$$
4. Продолжать шаг 3, пока при некотором $i = j$ не будет $F_j(A) = F_{j+1}(A)$ для всех нетерминалов. Приближения являются вложенными множествами, поэтому рано или поздно такое произойдет.
5. Положить $FIRST_k(A) = F_j(A)$.

Алгоритм вычисления функции σ(*A*)

Рассмотрим вспомогательную функцию

$$\sigma'(A,B)=\{\, L\subseteq V_T^{*k} : \exists\, A \Rightarrow wB\alpha, w\in V_T^*,\ L = FIRST_k(\alpha)\,\}$$

Так как $\sigma'(S,B)=\sigma(A)$, то достаточно рассмотреть алгоритм вычисления $\sigma'(A,B)$.

Алгоритм основан на построении последовательных приближений $\sigma'_i(A,B)$ для всех возможных пар нетерминалов грамматики.

1. $\sigma'_0(A,B) = \{\, L\subseteq V_T^{*k} : \exists\, A\to\beta B\alpha\in P, \beta,\alpha\in V^*,\ L = FIRST_k(\alpha)\,\}$.
2. Если $\sigma'_0(A,B)\ldots\sigma'_i(A,B)$ уже построены для всех пар нетерминалов, то положим

$$\sigma'_{i+1}(A,B) = \sigma'_i(A,B) \cup \{\, L\subseteq V_T^{*k} : \exists\, A\to X_1X_2\ldots X_m\in P;$$
$$L'\in\sigma'_i(X_p,B),\ X_p\in V_N,\ 1\le p\le m;\ L = L'\oplus_k FIRST(X_{p+1}\ldots X_m)\,\}.$$

3. Продолжать шаг 2, пока при некотором $i=j$ не будет

$$\sigma'_j(A,B) = \sigma'_{j+1}(A,B)$$

для всех пар нетерминалов. Приближения являются вложенными множествами, поэтому рано или поздно такое произойдет.

4. Положить $\sigma'(A,B) = \sigma'_j(A,B)$.
5. Положить $\sigma(A)=\sigma'(S,A)$. Если $\{\varepsilon\}\notin\sigma'(S,S)$, то $\sigma(S)=\sigma'(S,S)\cup\{\{\varepsilon\}\}$.

С помощью данного алгоритма также может быть вычислена функция **FOLLOW(k)** для нетерминалов, поскольку

$$FOLLOW_k(A) = \bigcup_{L\in\sigma(A)} L\,.$$

4 Реализация

Использованные технологии

В качестве языка разработки был выбран C#. Данный язык позволяет перегружать операторы (например, оператор ’+’ для сентенциальных форм) и использовать функции LINQ (Language Integrated Query)[1] для работы с последовательностями и множествами, что делает код более читаемым и понимаемым для человека.

Приложение реализовано на платформе .NET Core 3.1, позволяющей создавать программы для различных операционных систем и на разных операционных системах.

Тестирование кода осуществляется при помощи библиотеки XUnit, которая является одним из наиболее популярных решений для тестирования на платформе .NET.

[1] Language Integrated Query (LINQ). URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/ (дата обращения: 25.05.2021). Текст: электронный.

Архитектура

Грамматики и составляющие их элементы представлены следующими классами (рис. 1):

1. **GrammarSymbol** — класс, представляющий символ некоторого алфавита, который имеет поле Literal, определяющее строковое представление символа.
2. **Nonterminal** — класс, представляющий нетерминал грамматики.
3. **Terminal** — класс, представляющий терминал грамматики.

SententialForm — класс, представляющий слово, составленное из нескольких символов (GrammarSymbol). Дает возможность работы со словом как с массивом символов (доступ по индексу). Также определены операторы ’+’ и ’==’, позволяющие складывать символы, получая экземпляр SententialForm, и сравнивать слова поэлементно. Пустое предложение ε представлятся как пустой экземпляр SententialForm.

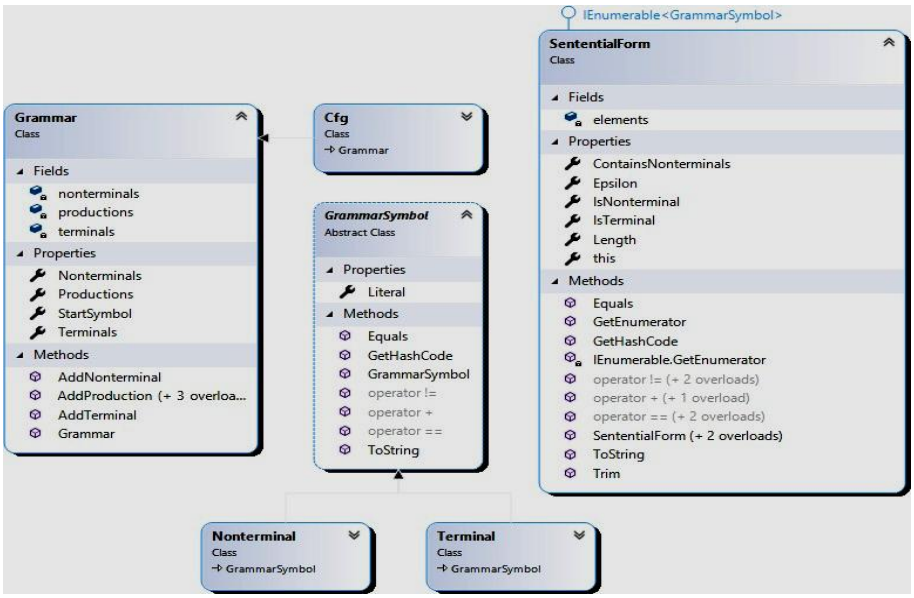

Рис. 1. Диаграмма классов грамматик и символов грамматик

1. **Grammar** — класс, представляющий формальную грамматику. Экземпляры этого класса имеют множества терминалов и нетерминалов, выделенный стартовый нетерминал и множество правил, представленных как пары экземпляров SententialForm.
2. **Cfg** — класс, представляющий контекстно-свободную грамматику. Отличается от Grammar тем, что позволяет добавлять только правила с одним нетерминалом в левой части.

Логика работы программы реализована в следующих классах (рис. 2):

3. **LLkFunctionsLogic.** Включает в себя реализацию функций FIRST(k), σ(A) и FOLLOW(k). Соответствующие функции First, Sigma и Follow принимают грамматику, нетерминал (или слово в случае First) и число k как параметры.
4. **LLkCheckerLogic.** Имеет метод Check, принимающий в качестве параметров грамматику и число k, и определяет, является ли данная грамматика $LL(k)$ грамматикой. Использует LLkFunctionsLogic.

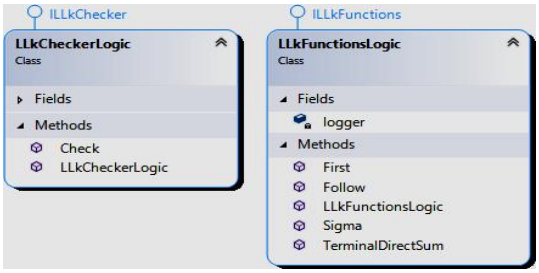

Рис. 2. Диаграмма классов, реализующих логику

Пользовательский интерфейс

Пользователь взаимодействует с приложением при помощи команд консоли (рис. 3). При запуске исполняемого файла в качестве аргумента командной строки передается путь к файлу с грамматикой, описанной в форме Бэкуса — Наура, например:

```
<S> ::= a <B> | b <A>
<A> ::= a | b <A> <A> | a <S>
<B> ::= b | a <B> <B> | b <S>
```

После запуска программы осуществляется разбор файла и генерация соответствующих экземпляров классов грамматики и символов. Затем полученные нетерминалы, терминалы и правила выводятся на экран и программа переходит в интерактивный режим, в котором доступны следующие команды:

- First. Принимает число k, запрашивает у пользователя слово в алфавите грамматики, считает функцию $FIRST(k)$ для него и выводит получившееся множество на экран.
- Follow. Принимает число k, запрашивает у пользователя нетерминал грамматики, считает функцию $FOLLOW(k)$ для него и выводит получившееся множество на экран.
- Check. Принимает число k, осуществляет проверку, является ли грамматика $LL(k)$-грамматикой, и выводит результат на экран.
- Exit. Выход из программы.

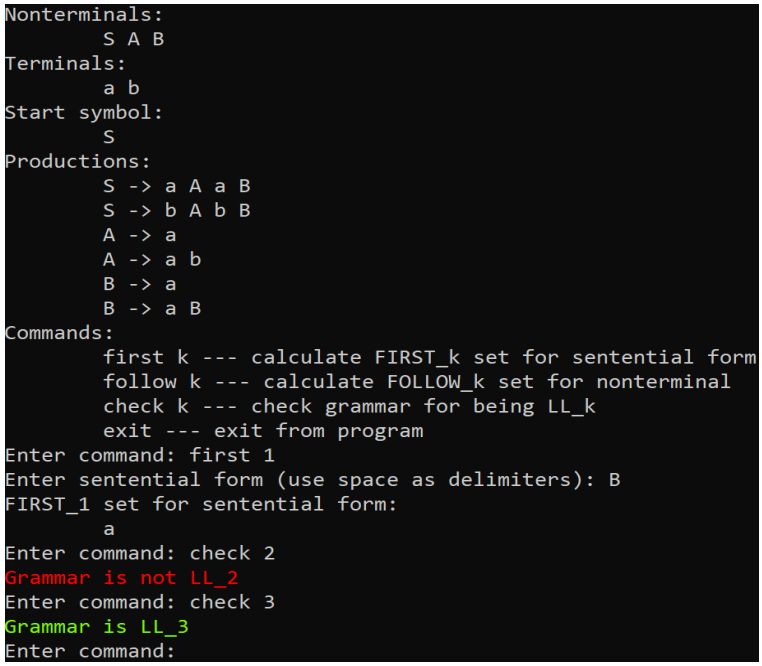

Рис. 3. Пользовательский интерфейс во время работы с программой

Фрагменты кода

Далее представлен исходный код реализации функций First, Sigma и Check.

Для построения последовательных приближений в функциях First и Sigma используется два экземпляра структуры данных HashSet — предыдущие приближения и строящиеся в данный момент. На каждой итерации происходит проверка этих множеств на равенство (функция DeepEquals), и в случае равенства происходит выход из цикла.

```
public HashSet<SententialForm> First(Cfg grammar, SententialForm ar-
     gument, int dimension = 1)
{
     if (dimension ≤ 0)
     {
          throw new ArgumentException(”Dimension must be a positive
          number.” ) ;
     }
     if (argument == SententialForm. Epsilon)
     {
          return new HashSet<SententialForm>() { SententialForm. Ep-
          silon };
     }
     foreach (var symbol in  argument)
```

```
{
     if (symbol is Terminal terminal && ! grammar. Terminals.
     Contains(
         terminal))
     {
      throw new  ArgumentException(”Terminal {terminal}  in
                          sententia is
not from grammar.” ) ;
     }
     if (symbol is Nonterminal nonterminal && ! gram-
     mar.Nonterminals.
         Contains(nonterminal))
     {
               throw new  ArgumentExcep-
          tion(”Nonterminal {nonterminal}  in
       sententia is not from
           grammar.” ) ;
     }
}
var approximations = new Dictionary<GrammarSymbol, Hash-
Set<SententialForm
    >>();
var prevApproximations = new Dictionary<GrammarSymbol, HashSet<
    SententialForm>>();
foreach (var terminal in  grammar. Terminals)
{
     approximations[ terminal] = new HashSet<SententialForm> {
     new
         SententialForm(terminal) };
                              }
}
foreach (var nonterminal in  grammar. Nonterminals)
{
approximations[ nonterminal] = new HashSet<SententialForm>() ;
     foreach (var production in  grammar. Productions. Where(p
         => p. left == nonterminal))
     {
          var trimmed = production. right. Trim(dimension) ; if
          ( ! trimmed.ContainsNonterminals)
          {
               approximations[ nonterminal ] . Add(trimmed) ;
          }
     }
```

```
}
...
(рамки объёма статьи не позволяют привести полный текст программы)
                if  (intersection. Count() > 0)
                    {
                        return false;
                    }
                }
            }
        }
}

return true;
                                }
```

5 Тестирование

Алгоритм был протестирован на проверку как свойства *LL(k)*, так и работы функций First, Follow и Sigma. Тестирование проводилось на нескольких грамматиках, для которых заранее были известны искомые свойства и значения функций (рис. 4). В результате программа была протестирована на 34 вариантах входных параметров для First, Follow и Sigma и 10 вариантах входных параметров для Check (рис. 5).

```
private static Cfg grammar1 = new Cfg(S)
    .AddNonterminal(A)
    .AddNonterminal(B)
    .AddTerminal(a)
    .AddTerminal(b)
    .AddProduction(S, a + B)
    .AddProduction(S, b + A)
    .AddProduction(A, a)
    .AddProduction(B, b)
    .AddProduction(A, b + A + A)
    .AddProduction(B, a + B + B)
    .AddProduction(A, a + S)
    .AddProduction(B, b + S) as Cfg;

private static Cfg grammar2 = new Cfg(S)
    .AddNonterminal(A)
    .AddNonterminal(B)
    .AddTerminal(a)
    .AddTerminal(b)
    .AddProduction(S, a + A + a + B)
    .AddProduction(S, b + A + b + B)
    .AddProduction(A, a)
    .AddProduction(A, a + b)
    .AddProduction(B, a)
    .AddProduction(B, a + B) as Cfg;
```

Рис. 4. Пример определения грамматики для тестирования

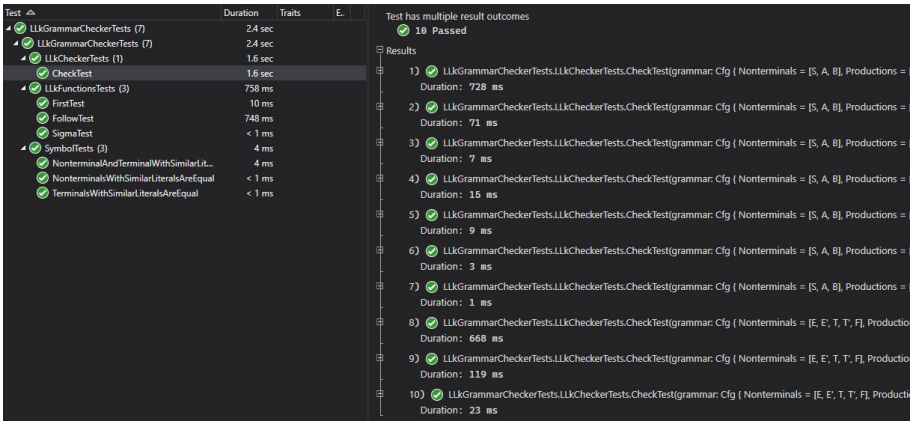

Рис. 5. Оповещение об успешно пройденных тестах в среде разработки Visual Studio

Заключение

В результате работы достигнуты следующие цели:

1. Проведен обзор алгоритма тестирования КС-грамматики на принадлежность классу грамматик $LL(k)$.
2. Алгоритм реализован в консольном приложении.
3. Проведены тестирование и отладка алгоритма.

Литература

1. Aho Alfred V., Ullman Jeffrey D. The Theory of Parsing, Translation, and Compiling. USA, New Jersey, 1972. 562 p.
2. Мартыненко Б. К. Языки и трансляции. Санкт-Петербург: Изд-во СПбГУ, 2013. 305 с. Текст: непосредственный.

Статья поступила в редакцию 31.05.2021; одобрена после рецензирования 25.06.2021; принята к публикации 19.08.2021.

IMPLEMENTATION OF THE ALGORITHM
FOR CF GRAMMAR TESTING FOR CLASS *LL(K)*

Ludmila N. Fedorchenko
Ph.D., Senior Researcher
St. Petersburg Institute for Informatics and Automation
of the Russian Academy of Sciences (SPIIRAS).
39, 14th Line V.O., St. Petersburg, 199178, Russia

Abstract. The article discusses an algorithm for testing a KS grammar in the Backus-Naur form for belonging to the *LL*(*k*) subclass of grammars. This is the most "natural" class of left-sided grammars in which left-sided analysis can be implemented in a deterministic manner. They allow the construction of left-hand input-output language strings using only the knowledge of k symbols ahead of going.

The implementation of the basic properties of such grammars as FIRST-FOLLOW-Sigma and the testing algorithm is considered. C # was chosen as the development language. The application is implemented on the .NET Core 3.1 platform, which allows you to create programs for various operating systems. Testing is carried out using the code XUnit library, which is one of the most popular solutions for testing on the .NET platform.

Keywords: context-free grammars; Backus-Naur form; testing algorithm; *LL(k)* grammars; FIRST, FOLLOW, Sigma functions.

For citation

Fedorchenko L. N. Implementation of the Algorithm for CF Grammar Testing for Class *LL*(k) // *Bulletin of Buryat State University. Mathematics, Informatics.* 2021; 2: 17–27 (In Russ.).

References

1. Aho Alfred V., Ullman Jeffrey D. The Theory of Parsing, Translation, and Compiling. USA, New Jersey, 1972. 562 p.
2. Martynenko B. K. Languages and Broadcasts. St. Petersburg State University Publishing House, St. Petersburg, 2013. 305 p.

The article was submitted 31.05.2021; approved after reviewing 25.06.2021; accepted for publication 19.08.2021.