ISSN 2310-0370

nauchforum.ru

НаучФорум

Оставь свой след в науке

XVI Студенческая международная заочная научно-практическая конференция

МОЛОДЕЖНЫЙ НАУЧНЫЙ ФОРУМ: ТЕХНИЧЕСКИЕ И МАТЕМАТИЧЕСКИЕ НАУКИ

№ 9 (16)

г. МОСКВА, 2014

nauchforum.ru
НаучФорум
Оставь свой след в науке

МОЛОДЕЖНЫЙ НАУЧНЫЙ ФОРУМ: ТЕХНИЧЕСКИЕ И МАТЕМАТИЧЕСКИЕ НАУКИ

Электронный сборник статей по материалам XVI студенческой международной заочной научно-практической конференции

№ 9 (16)
Сентябрь 2014 г.

Издается с марта 2013 года

Москва
2014

УДК 62+51
ББК 30+22.1
М 75

М 75 Молодежный научный форум: Технические и математические науки. Электронный сборник статей по материалам XVI студенческой международной заочной научно-практической конференции. — Москва: Изд. «МЦНО». — 2014. — № 9 (16) / [Электронный ресурс] — Режим доступа. — URL: http://www.nauchforum.ru/archive/MNF_tech/9(16).pdf

Электронный сборник статей XVI студенческой международной заочной научно-практической конференции «Молодежный научный форум: Технические и математические науки» отражает результаты научных исследований, проведенных представителями различных школ и направлений современной науки.

Данное издание будет полезно магистрам, студентам, исследователям и всем интересующимся актуальным состоянием и тенденциями развития современной науки.

ББК 30+22.1

ISSN 2310-0370

Оглавление

СЕКЦИЯ 1.

АРХИТЕКТУРА, СТРОИТЕЛЬСТВО

СОВРЕМЕННЫЕ МЕТОДЫ УСИЛЕНИЯ ЖЕЛЕЗОБЕТОННЫХ КОНСТРУКЦИЙ

Нестеренко Юлия Анатольевна
студент Тольяттинского политехнического техникума,
РФ, г. Тольятти

Юрьев Алексей Владимирович
научный руководитель, преподаватель
Тольяттинского политехнического техникума,
РФ, г. Тольятти

Усиление конструкций — это главная составляющая любого строительного процесса, связанного с повышением общей прочности любого сооружения.

В процессе эксплуатации зданий и сооружений периодически возникает необходимость проведения ремонтов. Это объясняется наличием различных воздействий на строительные конструкции — непроектных нагрузок, аварий, перепланировок, воздействием агрессивных химических сред. Кроме того, к проведению ремонтов могут вынуждать допущенные при проектировании либо проведении строительных работ ошибки. В свете вышесказанного становится востребованным усиление строительных конструкций для продления их срока эксплуатации.

Для усиления железобетонных конструкций разработано большое количество способов:

- увеличение геометрических размеров поперечных сечений конструктивных элементов, что сопровождается увеличением собственного веса конструкций и увеличением строительной высоты;

- устройство внешних стяжек, подпоров, поясов, шпренгелей, приводящее к изменению архитектурного вида сооружений и значительным временным и материальным затратам;
- приклеивание металлических пластин или их сварка.

Тем не менее, как видно из опыта, усиление железобетонных конструкций традиционными методами не всегда оказывается эффективным. В последнее время на отечественном рынке появляются современные методы усиления конструкций, широко применяемые за рубежом.

Подобные методы достаточно эффективны и просты.

Так, усиление железобетонных конструкций, путем наклейки композиционных материалов позволяет в значительной степени увеличить их несущую способность и жесткость, а также продлить срок эксплуатации всего сооружения. Здесь следует отметить основные следующие преимущества материала:

- совместная работа элемента внешнего армирования с усиливаемой конструкцией на всех этапах ее загружения (такая работа обеспечивается надежным клеевым соединением);
- высокая долговечность и стойкость к коррозии;
- высокие механические характеристики (прочность и модуль упругости) материалов, составляющих систему усиления;
- высокое относительное удлинение материалов усиления;
- простота монтажа и малый собственный вес.

О надежности подобного усиления можно спорить, но факт в том, что она доказана экспериментально. О каких механических характеристиках идет речь? Это, во-первых, прочность. Она начинается от 35 тыс. кг/см^2, если мы говорим об углеволокне. Во-вторых, это модуль упругости — до 640 тыс. МПа, т. е. практически в 3 раза больше, чем у стали. Для склейки используются специальные монтажные эпоксидные клеи. Они обладают технологическим совершенством, т. к. их можно наносить на железобетонную конструкцию, имеющую естественную влажность. Безусловно, у рассматриваемого усиления

имеются и недостатки. Кроме высокой стоимости самих элементов армирования, это и необходимость их защиты от огня. Дело в том, что температура стеклования эпоксидного клея составляет только 60°—65^0 С, даже в случае самых лучших эпоксидов. Поэтому необходимо очень тщательно готовить бетонную поверхность для обеспечения надежной анкеровки, а это и регламентные работы, которые необходимо проводить для усиления.

Сравнение метода усиления конструкций композитными материалами из углеродного волокна с методом усиления конструкций стальными полосами показано в таблице 1.

Таблица 1.

Таблица сравнений

Усиление конструкций стальными панелями		**Усиление конструкций композитными материалами**	
Достоинства:	**Недостатки:**	**Достоинства:**	**Недостатки:**
• сталь обладаетотносительно • низкой стоимостью • стальные панели • достаточно • универсальны • •стальные панели • обладают достаточной • усталостной прочностью	• •возможна коррозиястальных элементов • •стальные элементы • обладают значительным весом • высокая трудоемкость работы, влекущая высокую стоимость рабочей силы • •для выполнения работ требуются площадки больших размеров • стальные панели ограничены в размерах	○ отличная стойкость к коррозии ○ композитные панели имеют уникальную прочность на растяжение, на порядок выше, чем стальные, а также обладают очень высокой усталостной прочностью ○ быстрота процесса ○ отсутствует необходимость устройства рабочих площадок (работы могут выполняться с автоподъемника) • • простое соединение композитного материала с усиливаемым элементом при помощи клея	○ относительно высокая стоимость • • необходимость защиты от огня

Так же применяется способ оклейки конструкций стекловолокном. Он дешевле, но модули упругости и прочность у него не такие высокие (до 100 тыс. МПа и 1700 МПа, соответственно), но это тоже очень прилично.

Метод усиления элементами внешнего армирования из высокопрочных волокон имеет ряд обстоятельств.

Первое — на сегодня данный метод является самым совершенным в техническом отношении. Он осуществим, в отличие от традиционных методов. Очень часто выходит, что традиционное усиление практически не работает, включить его в совместную с усиливаемой конструкцией работу нельзя. Второе обстоятельство — метод усиления элементами внешнего армирования из высокопрочных волокон является очень бережным методом. Внешнее вмешательство в усиливаемую конструкцию минимальное. Фактически мы располагаем элементы внешнего армирования именно в том направлении, чтобы, парировать возможное развитие нежелательных деформаций, трещин и т. д. вдоль линии главных растягивающих напряжений. В результате усиливаемая конструкция становится еще более надежной, причем без дополнительной анкеровки. И третье обстоятельство — комплексный подход к ремонту и усилению элементы внешнего армирования не монтируются на поврежденную железобетонную конструкцию. Имеющиеся дефекты устраняются методами строительной химии.

Благодаря использованию метода усиления конструкций композитными материалами на основе углеволокна становится возможным усиление стен, колонн, проемов, балок, перекрытии а также, позволит эксплуатировать объект безопасно, на протяжении многих лет.

Список литературы:

1. Грановский А.В. Сейсмостойкость стен, усиленных композитными.
2. Шилин А.А., Пшеничный В.А., Картузов Д.В. Внешнее армирование железобетонных конструкций композиционными материалами. М., 2007.

СЕКЦИЯ 2.

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

ПОИСК ПОДСТРОК С ПОМОЩЬЮ КОНЕЧНОГО АВТОМАТА И РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

Вандина Александра Игоревна
студент Армавирского механико-технологического института филиала ФГБОУ ВПО КубГТУ, РФ, г. Армавир

Зуева Виктория Николаевна
научный руководитель, доц. Армавирского механико-технологического института филиала ФГБОУ ВПО КубГТУ, РФ, г. Армавир

Поиск информации — одно из основных использований компьютера. Одна из простейших задач поиска информации — поиск точно заданной подстроки в строке. Тем не менее, эта задача чрезвычайно важна — она применяется в текстовых редакторах, СУБД, поисковых машинах [1; 2; 3; 4].

С помощью конечных автоматов можно успешно решать обширный класс задач. Это обстоятельство подмечено давно, поэтому в литературе по проектированию программного обеспечения часто приводятся рассуждения на тему применения автоматов. Однако в процессе моделирования автомат рассматривается с более высокого уровня, нежели это делается в момент его реализации с использованием конкретного языка программирования [1; 3].

Рассмотрим, как описать конечный автомат для поиска подстроки в строке, когда они совершенно произвольны. На рисунке 1 представлен граф данного автомата.

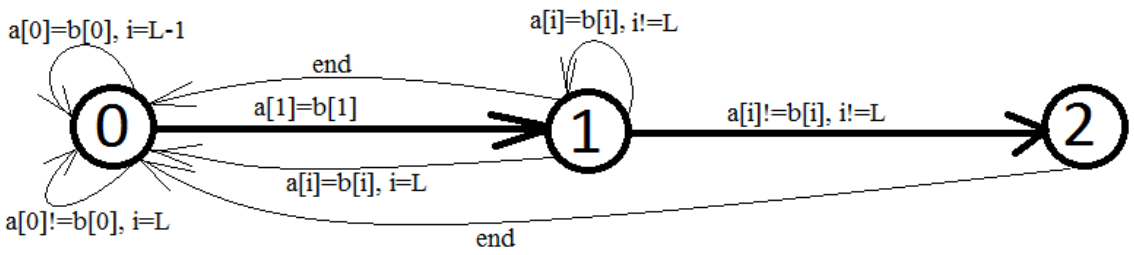

Рисунок 1. Граф конечного автомата поиска произвольной подстроки в строке

Здесь **a** — проверяемая часть строки, **b** — искомая подстрока,

L=L_a=L_b — длина подстроки;

end говорит о выходе из автомата и перехода в начальное состояние в ожидании следующего элемента для проверки.

Программно на языке высокого уровня C# такой автомат реализуется следующим образом:

```
for (int b = 0; b < isub; )
switch (q)
{
case 0:
{
if (str0[0] == sub[0])
{
if (b == isub - 1)
{
                kol++;
                q = 0;
                b++;
}
else
{
                q = 1;
                b = 1;
```

```
}
}
else
{
q = 2;
b++;
if (b == isub) q = 0;
}
} break;
case 1:
{
if (str0[b] == sub[b])
{
q = 1;
b++;
if (b == isub) { kol++; q = 0; }
}
else
{
q = 2;
b++;
if (b == isub + 1) q = 0;
}
} break;
 case 2:
{
 b = isub;
 q = 0;
} break;
}
```

Такой способ поиска подстроки в строке работает эффективно, но его использование не целесообразно, т. к. для работы с текстом имеется свой собственный язык — регулярные выражения. Их работа тоже строится на конечных автоматах, но «скрытно», т. е. программист не беспокоится о построении автомата вручную, а лишь обращается к регулярным выражениям.

Регулярные выражения можно представить себе как мини-язык программирования, имеющий одно специфическое назначение: находить подстроки в больших строковых выражениях.

Язык регулярных выражений предназначен специально для обработки строк. Он включает два средства: [1; 4].

- набор управляющих кодов для идентификации специфических типов символов;
- система для группирования частей подстрок и промежуточных результатов таких действий.

С помощью регулярных выражений можно выполнять достаточно сложные и высокоуровневые действия над строками:

- идентифицировать (и возможно, помечать к удалению) все повторяющиеся слова в строке;
- сделать заглавными первые буквы всех слов;
- преобразовать первые буквы всех слов длиннее трех символов в заглавные;
- обеспечить правильную капитализацию предложений;
- выделить различные элементы в URI (например, имея http://www.professorweb.ru, выделить протокол, имя компьютера, имя файла и т. д.).

Вот пример поиска и выделения цветом подстроки в тексте с помощью регулярного выражения:

```
MatchCollection allIp = Regex.Matches(rtb.Text.ToLower(), Class1.Poisk.ToLower());
foreach (Match ip in allIp)
```

```
{
 rtb.SelectionStart = ip.Index;
rtb.SelectionLength = ip.Length;
 rtb.SelectionBackColor = Color.FromArgb(255, 160, 122);
}
```

Здесь в тексте rtb.Text ведется поиск слова Class1.Poisk. Функция ToLower() организует игнорирование регистра, чего можно добиться и стандартными командами регулярных выражений, как RegexOptions.IgnoreCase [4]. Приведем пример программы, выполнение которой построено на основе регулярных выражений. На рисунке 2 представлено окно данной программы. Ее задачей является поиск заданного слова в списке файлов MS Office Word, выбранных пользователем, подсчет числа совпадений и вывода этого результата на экран в виде таблицы.

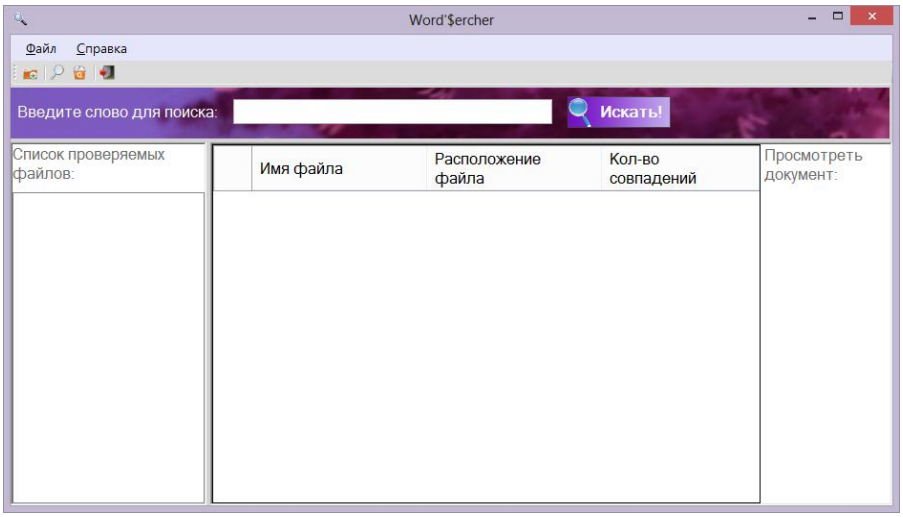

Рисунок 2. Начальное окно разработанной программы

Поиск заданной подстроки как раз и реализуется с помощью регулярных выражений. Кроме того, используя их можно не только найти и подсчитать, но и выделить полученные совпадения, за что и отвечает приведенный немного выше программный код. Продемонстрируем описанные действия в программной реализации (рисунки 3, 4):

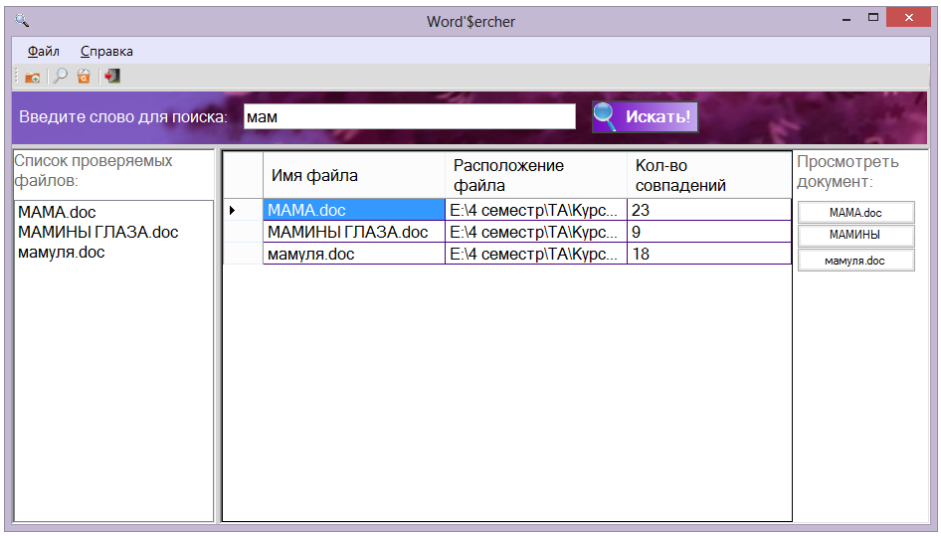

Рисунок 3. Пример выполнения программы

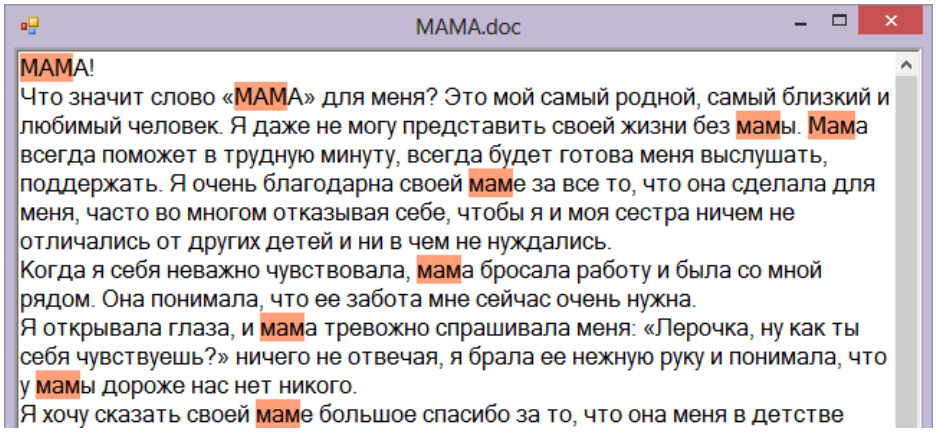

Рисунок 4. Просмотр документа с выделением найденных совпадений

Здесь была представлена лишь малая часть из возможностей регулярных выражений, но даже так можно утверждать, что при работе с текстом такой вид поиска подстроки заметно короче и явно удобнее, чем использование непосредственно конечных автоматов.

Список литературы:

1. Брауэр В. Введение в теорию конечных автоматов / пер. с англ.; под ред. Ю.И. Журавлева. — М.: Радио и связь, 1987. — 392 с.
2. Зуева В.Н. Адаптивный поиск информации в Internet // Современные тенденции технических наук: материалы междунар. заоч. науч. конф. (г. Уфа, май 2013 г.). — Уфа: Лето, 2013. — С. 7—10.
3. Карпов Ю.Г. Теория автоматов: учебник для вузов. — М.; Питер, 2002. — 207 с.
4. Фаронов В.В. Программирование на языке C#. — СПб.: Питер. 2007. — 240 с.

ИСПОЛЬЗОВАНИЕ ТРАНСЛИРУЮЩИХ ГРАММАТИК ПРИ ПРОЕКТИРОВАНИИ ТРАНСЛЯТОРА

Манин Максим Павлович
студент Армавирского механико-технологического института филиала ФГБОУ ВПО КубГТУ, РФ, г. Армавир

Зуева Виктория Николаевна
научный руководитель, доц. Армавирского механико-технологического института филиала ФГБОУ ВПО КубГТУ, РФ, г. Армавир

В настоящее время в мире насчитывается свыше тысячи языков программирования. Однако язык останется мертворожденным, если он не будет реализован на ЭВМ. Реализация языка означает разработку и использование специальной программы — транслятора, «понимающей» тексты на этом языке и осуществляющей процесс трансляции, то есть перевода исходного текста программы в исполнительный файл или программу на другом языке программирования.

Примерами таких языков могут служить системы запросов, языки управления некими объектами, различные препроцессоры, языки систем автоматизированного проектирования и т. д.

Существование различных языков программирования зачастую обязано возникающей актуальной до настоящего времени задача представления, то есть описания языка программирования, значимость которой особенна тем, что любой подобного рода формализм позволяет перейти на более высокий уровень абстракции, охватывающий уже гораздо больше конкретных вариантов, т. е. более высокий уровень абстракции позволяет решать более сложные задачи в достаточно обширных областях.

Традиционно описание языка состоит из описания его лексических свойств, описывающих правила составления отдельных слов языка или лексем, синтаксических свойств, описывающих правила составления из лексем отдельных предложений, принадлежащих описываемому языку и семанти-

ческих свойств, определяющих корректность совокупности предложений языка с точки зрения смысла. Для первых двух составляющих разработаны соответствующие формализмы. Существуют методы и для описания семантики, но их формальное применение при разработке транслятора представляется достаточно затруднительным [1].

Механизм описания формального языка основан на принципах предварительного метапрограммирования, представляющего собой вид программирования, связанный с созданием программ, которые порождают другие программы как результат своей работы. В частности при описании формальных языков языком метапрограммирования является форма Бэкуса-Наура или формальная система описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие.

С помощью подобной иерархической парадигмы метапрограммирования можно описывать довольно сложные синтаксические структуры и языки программирования.

Однако при более специализированных взаимосвязях сложных синтаксических конструкций, описать которые с помощью подобного рода грамматик невозможно на практике их расширяют понятиями атрибутных грамматики, которые позволяют наследовать атрибуты для вложенных грамматических правил, позволяя тем самым осуществлять, например, проверку на соответствие типов.

Зачастую сам процесс трансляции осуществляется в три этапа: лексический и синтаксический анализ и непосредственно трансляция программы в целевую форму. При этом также существует множество подходов реализации данного процесса, одним из которых является использование транслирующих грамматик, которые в отличии от обычных грамматик включают в свой состав так называемые символы действия. При этом процесс синтаксического анализа объединяется вместе с процессом трансляции: то есть при проверке входной строки на её соответствие синтаксису при встрече в искомой грамматике символа действия транслятор выполняет ассоциированное с ним действие,

вследствие чего результатом работы транслятора является последовательность актов, каждый из которых способен определенным образом конфигурировать выходную строку [1; 3].

С помощью данного подхода можно организовать перевод математических выражений из инфиксной формы в постфиксную, являющимся предметом настоящего исследования.

Постфиксная запись или обратная польская нотация (ОПН) — форма записи математических выражений, предложенная польским математиком Яном Лукасевичем в которой операнды расположены перед знаками операций [3].

Подобный стиль записи математических выражений является наиболее оптимальным при автоматическом вычислении значения выражения с использованием стековой архитектуры. При этом порядок вычисления элементарен: если следующий символ в выражении — число, то оно заносится на вершину стека, если это операция, то из стека выталкиваются два верхних элемента, над которыми эта операция выполняется, а результат снова заносится на вершину стека.

В структуру синтаксического блока некоторых компиляторов включается специальный блок перевода арифметических выражений в постфиксную форму. Реализацию поставленной задачи реализации подобного рода предварительного транслирующего блока принято осуществлять с проектирования целевой транслирующей грамматики, лежащей в его основе с использованием формы Бэкуса-Наура, которая в результате принимает следующий вид, где ε — пустой символ.

<S> → <Id>{Id}<Spaces>={=}<Spaces>< Expression ><Spaces>;{PopAllSign}{;}<Spaces><S> | ε

<Spaces> → <Space><Spaces> | ε

<Space> → | \r | \n | \t

<Id> → <FirstChar><Chars>

<FirstChar> → _ | a | b | c | … | z | A | B | C | … | Z

<Chars> → <Char><Chars> | ε

<Char> → <FirstChar> | <Digit>

<Expression> → <Operand><Spaces><Boundary>

<Boundary> → <Sign>{TryPopSign}<Spaces><Expression> | ε

<Sign> → + | – | * | /

<Operand> →
({PushBraket}<Spaces><Expression><Spaces>){PopToBracket} |

<Number>{<Number>} | <Id>{<Id>}

<Number> → <Digits><FloatDigits>

<FloatDigits> → .<Digits> | ε

<Digits> → <Digit><Digits> | ε

<Digit> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

В представленной грамматике используется ряд операционных нетерминальных символов, обеспечивающих процесс трансляции входной строки в результирующую строку в постфиксной форме в ходе синтаксического анализа.

Операционный нетерминал {TryPopSign} обозначает операцию, выполняющую попытку выталкивания предыдущего символа арифметической операции из стека операций, при этом производится сравнивание предыдущей и текущей операции и первая выталкивается на выходную ленту только в случае её равенства или старшинства по сравнению с текущей. Далее в любом исходе текущий символ операции вталкивается в стек операций, ожидая следующей проверки.

Операционный нетерминал {PushBraket} обозначает операцию вталкивания в операционный стек открывающей скобки, что позволяет выделять выражения, объединенные в скобки.

Операционный нетерминал {PopToBracket} обозначает завершение выражения объединенного в скобку и выполняет выталкивание всех операций из операционного стека в выходную ленту в порядке их размещения, пока не встретится соответствующая открывающая скобка.

Операционный символ {PopAllSign} обозначает завершение всего выражения и выполняет выталкивание на выходную ленту оставшихся операций.

Остальные операционные символы {Id}, {=}, {;}, {<Number>} и {<Id>} просто отображают в выходную строку предшествующие лексемы чисел, идентификаторов и служебных символов на выходную ленту.

В качестве практической реализации представленного транслятора математических выражений выступает разработанное в рамках данного исследования приложение на языке C# с использованием библиотеки WPF [2].

Процесс трансляции осуществляется в виде двухступенчатого конвейера, при котором каждая ступень передает управление следующей после полного анализа принятых данных, а полученные результаты передаются на вход следующего анализатора.

С помощью представленной организации транслятора можно разделить идентификацию ошибок по их принадлежности к классу лексических или синтаксических ошибок, которые в окне ввода и логе ошибок отображаются соответствующим образом, пример которого представлен на рисунке 1.

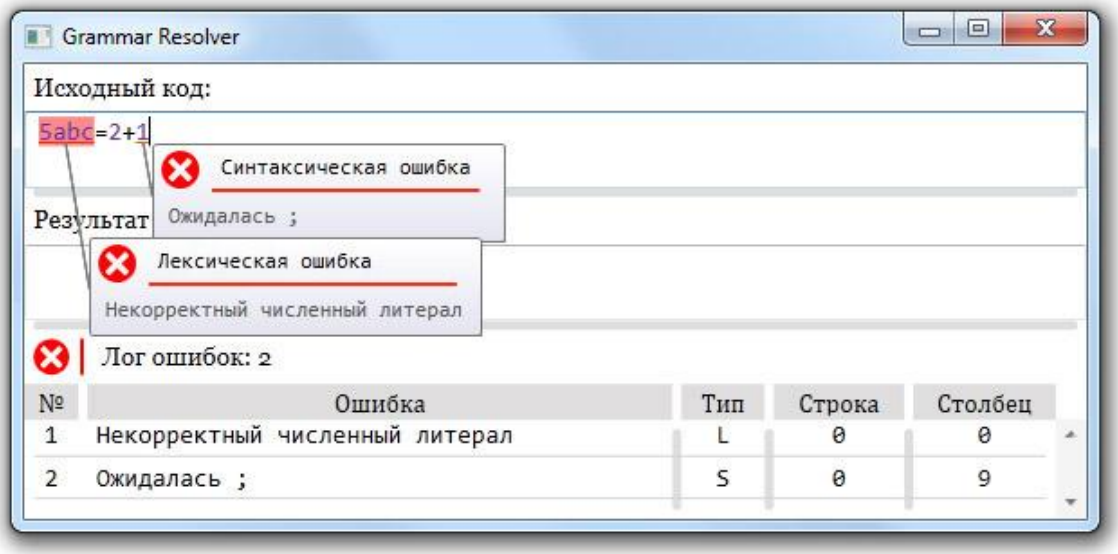

Рисунок 1. Пример обработки ошибок

По объективным причинам при наличии ошибок нижнего или верхнего уровня процесс трансляции результата работы не осуществляется, а возвращаются только списки ошибок.

Результат корректной трансляции представлен на рисунке 2.

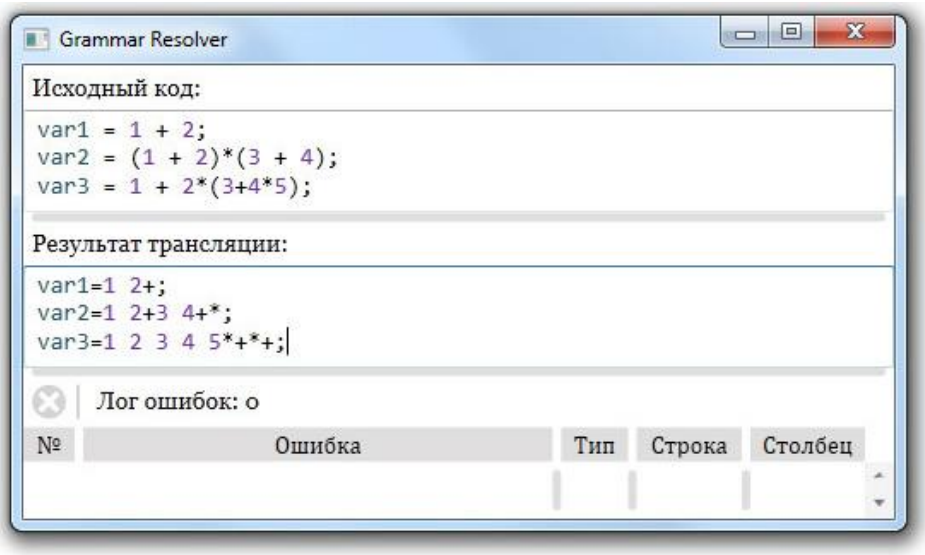

Рисунок 2. Пример корректной трансляции

В качестве заключения нужно сказать, что исследования, проводимые в области теории языков программирования, методов трансляции, разработки новых методик формализации и абстрагирования от оборудования являются достаточно востребованными, так как сложность современных программных продуктов, систем автоматизированного управления растет и их применение расширяется, соответственно существует потребность в средствах эффективной, качественной и быстрой программной разработке.

Список литературы:

1. Кревский И.Г., Селиверстов М.Н., Григорьева К.В. Формальные языки, грамматики и основы построения трансляторов: Учебное пособие / Под ред. А.М. Бершадского — Пенза: Издательство ПГУ, 2002. — 124 с.
2. Мэтью Мак-Дональд. Pro WPF in C# 2010: Windows Presentation Foundation in .NET 4. — Москва, Санкт-Петербург, Киев: Издательский дом «Вильямс», 2011. — 1024 с.
3. Свердлов С.З. Языки программирования и методы трансляции: Учебное пособие. — СПб.: Питер, 2007. — 638 с.

АСИНХРОННОЕ ПРОГРАММИРОВАНИЕ В ASP.NET MVC 4

Сорокин Алексей Александрович
студент 4 курса, кафедра ИТ, ДИТИ НИЯУ МИФИ,
РФ, Ульяновская область, г. Димитровград

Белянина Дарья Николаевна
ассистент, ДИТИ НИЯУ МИФИ,
РФ, Ульяновская область, г. Димитровград

Язык программирования C# не стоит на месте, и за свою короткую историю вдохнул жизнь в многочисленные коммерчески успешные и полезные с точки зрения потребителя проекты, такие как браузер Opera, Banshee, Beagle, F-spot, MonoDevelop, Tomboy и многие другие. С каждой новой версией язык приобретает новые очертания, открывает новые возможности, одним словом, развивается.

В данной статье будет рассмотрено одно из многочисленных новшеств, появившихся в языке — написание асинхронных методов. Работа с асинхронными операциями была введена на уровне языка, что говорит о серьезности подхода к данной проблеме разработчиками. Стоит заметить, что вопросы относительно асинхронного программирования ложились на плечи только самым квалифицированным специалистам, у которых хватало терпения, опыта, знаний и выдержки справиться с поставленной задачей, чье решение неэффективно стандартными методами.

Решение асинхронной задачи привычными синхронными методами приводит к непродуктивным задачам. Это касается действий, требующих длительного времени для выполнения. В итоге операция блокирует основной поток до своего завершения, и программа сильно теряет позиции в производительности. Для веб-приложений, работа которых связана с многочисленными запросами к различным веб-сервисам будет полезна асинхронность передачи запросов.

В программировании понятие «асинхронность» тесно связано с многопоточностью. Таким образом, когда необходим параллелизм операций, часто

применяется модель нитей и процессов. Но она подходит для большого объема вычислений, так как несет в себе значительные накладные расходы, связанные со сменой контекста синхронизации, памятью и т. п., которые перекладываются на операционную систему.

В ИТ-сфере растет популярность ASP .NET MVC в сфере веб-приложений. До ASP.NET MVC 4 у программистов были такие средства как асинхронные контроллеры, асинхронный javascript и XML(AJAX) или специальные классы, способные переводить действия в отдельные потоки, например, класс BackgroundWorker.

Рассмотрим пример: необходимо разработать страницу, которая сформирует сведения о наличии (отсутствии) у юридического лица задолженности в ФНС (федеральная налоговая служба) и сведения, содержащиеся в ЕГРЮЛ (единая государственная регистрация юридических лиц). Для этого имеется два метода, которые посылают соответствующие запросы: sendXmlTaxInfo и sendXmlEGRInfo. Аргумент-строка содержит xml-разметку, несущую в себе информацию о запрашиваемом юридическом лице, например:

```
<Document ИдЗапрос=”6F9619FF-8B86-D011-B42D-00CF4FC964FF” >
<ИННЮЛ>7726297700</ИННЮЛ>
<ОГРН>1027739280557</ОГРН>
</Document>
```

В момент выполнения выше представленных методов, этот блок вставляется в шаблон xml-документа, извлекаемого из базы данных. При последовательной отправке запросов пользователю придется ждать, пока ответ на первый запрос не будет получен, и только после этого будет отправлен второй запрос. Целесообразно в данном случае выполнять запросы параллельно, сократив время ожидания вдвое.

До выхода версии C# 5 можно было воспользоваться асинхронным контроллером, а теперь – асинхронной моделью на основе событий:

```
public class SendRequestController : AsyncController
{
```

```
public void SendRequestFNSAsync (string inxml)
{
AsyncManager.OutstandingOperations.Increment(2);
sendXmlTaxInfoComplited += (sender, e) =>
{
AsyncManager.Parameters["TaxInfo"] = e.Value;
AsyncManager.OutstandingOperations.Decrement();
};
sendXmlTaxInfoAsync(inxml);
sendXmlEGRInfoComplited += (sender, e) =>
{
AsyncManager.Parameters["EGRInfo"] = e.Value;
AsyncManager.OutstandingOperations.Decrement();
};
sendXmlEGRInfoAsync(inxml);
}

public ActionResult SendRequestFNSCompleted(string TaxInfo, string EGRInfo)
{
ULInfoModel ul = new ULInfoModel();
ul.generateInfo(TaxInfo, EGRInfo);
return View("TaxEGRInfo", ul);
}
}
```

Контроллер в обязательном порядке необходимо унаследовать от AsyncController, он позволит выполнять обработку асинхронных запросов. Можно заметить, что все методы имеют две части выполнения, которые обязательно заканчиваются словами Async и Completed. Все методы Async в пределах выполняемого метода запускаются одновременно. По завершении

соответствующего метода срабатывает обработчик Completed. Класс AsyncManager — вспомогательный класс. Свойство OutstandingOperations сообщает ASP.NET о количестве операций в очереди — в примере выполняются две операции, посылаются два запроса. Для передачи результатов запросов используется словарь Parameters. Таким образом, время выполнения стало значительно меньше, благодаря асинхронным вызовам.

Ниже рассмотрена задача на уровне языка.

```
public class SendRequestController : Controller
{
public async Task<ActionResult> SendRequestFNSAsync()
{
Task<string> getRequest_1 = sendXmlTaxInfo(inxml);
Task<string> getRequest_2 = sendXmlEGRInfo(inxml);
ULInfoModel ul = new ULInfoModel();
string taxInfo = await getRequest_1;
string EGRInfo = await getRequest_2;
ul.generateInfo(taxInfo, EGRInfo);
return View("TaxEGRInfo", ul);
}
}
```

Для написания асинхронного метода нужно использовать два ключевых слова:

1. async для объявления метода асинхронным. При этом возвращаемым значением должно быть: Task<ResultClass>, если результатом исполнения метода является класс ResultClass; Task, если метод не должен ничего возвращать; void, если есть необходимость написать асинхронный обработчик событий;

2. await, чтобы объявить компилятору ожидание окончания выполнения метода;

Так как результатом запроса должна быть строка string, содержащая xml-разметку, объявляется задача, которая гарантирует вернуть тип string следующим кодом:

Task<string> getRequest_1 = sendXmlTaxInfo(inxml);

Следующий за этим объявлением код беспрепятственно исполняется параллельно с запросом, пока не встретит команду:

string taxInfo = await getRequest_1;

Она сообщает компилятору об ожидании результата запроса, без которого дальнейшее исполнение программы невозможно. Получив результаты исполнения обоих запросов, в модели происходит обработка результатов и сама модель передается в представление.

Таким образом, код получился проще, короче и читабельнее. Главное преимущество второго варианта — простота написания, отладки и сопровождения в сложных приложениях. Здесь используется асинхронная модель на основе задач — самое современное средство для разработки асинхронных методов.

Список литературы:

1. Дэвис А. Асинхронное программирование в C#5 (Async in C# 5.0) [Текст]: Учеб. пособие / А. Дэвис. — М.: ДМК Пресс, 2012. — 120 с.
2. Сандерсон С. ASP .NET MVC Framework с примерами на C# для профессионалов [Текст]: Учеб. пособие / С. Сандерсон. — М.: ООО «И.Д. Вильямс», 2010. — 560 с.
3. Чедвик Дж. ASP.NET MVC 4: разработка реальных веб-приложений с помощью ASP.NET MVC (Programming ASP.NET MVC 4: Developing Real-World Web Applications with ASP.NET MVC) [Текст]: Учеб. пособие / Дж. Чедвик — М.: ООО «И.Д. Вильямс», 2013. — 432 с.

ДЛЯ ЗАМЕТОК

МОЛОДЕЖНЫЙ НАУЧНЫЙ ФОРУМ: ТЕХНИЧЕСКИЕ И МАТЕМАТИЧЕСКИЕ НАУКИ

Электронный сборник статей по материалам XVI студенческой международной заочной научно-практической конференции

№ 9 (16)
Сентябрь 2014 г.

В авторской редакции

Издательство «МЦНО»
127106, г. Москва, Гостиничный проезд, д. 6, корп. 2, офис 213

E-mail: mail@nauchforum.ru