УДК [378.016:796]:004.9 DOI 10.37972/chgpu.2020.28.84.027

И. А. Обломов[2], В. В. Ржавин[2], Н. В. Первова[2], А. Г. Герасимова[1]

ПРИМЕНЕНИЕ МОДЕЛИ СИНТАКСИЧЕСКИ УПРАВЛЯЕМОГО ПЕРЕВОДА АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ В ПРОЦЕССЕ ОБУЧЕНИЯ СТУДЕНТОВ

[1]Чувашский государственный педагогический университет им. И. Я. Яковлева, г. Чебоксары, Россия

[2]Чувашский государственный университет имени И. Н. Ульянова, г. Чебоксары, Россия

Аннотация. В статье рассматривается модель синтаксически управляемого перевода простых арифметических выражений и ее использование в процессе обучения. Атрибутно-транслируемая грамматика предполагает перевод последовательности актов в последовательность действий, которые, в свою очередь, будут являться исходными данными для следующих этапов трансляции. Раскрываются основные моменты обучения студентов декларативному языку программирования Пролог, делается упор на обработку множества символов действия.

Дальнейшие исследования предполагают разработку моделей синтаксического анализа с помощью средств императивных и функциональных языков программирования с целью получения и анализа объективных оценок эффективности полученных моделей в процессе обучения будущих специалистов.

Ключевые слова: *обучение информатике, студенты вуза, языки императивные и декларативные, синтаксический анализ, преподавание языка Пролог.*

I. A. Oblomov[2], V. V. Rzhavin[2], N. V. Pervova[2], A. G. Gerasimova[1]

APPLICATION OF MODEL OF SYNTACTICALLY CONTROLLED TRANSLATION OF ARITHMETIC EXPRESSIONS IN THE PROCESS OF TEACHING STUDENTS

[1]I. Yakovlev Chuvash State Pedagogical University, Cheboksary, Russia

[2]I. Ulyanov Chuvash State University, Cheboksary, Russia

Abstract. This article discusses the model of syntactically controlled translation of simple arithmetic expressions and its use in the learning process. The attribute-translated grammar involves the translation of a sequence of acts into a sequence of actions, which will be the source data for the next stages of translation. The article reveals the main points of teaching students the Prolog programming language, focuses on the processing of many action symbols.

Further research involves the development of models of syntactic analysis by means of imperative and functional programming languages in order to obtain and analyze the objective estimates of the effectiveness of the obtained models in the training of future specialists.

Keywords: *teaching Computer Science, university students, imperative and declarative languages, syntactic analysis, teaching Prolog language.*

Актуальность исследуемой проблемы. Традиционные методы обучения будущих специалистов, без привлечения современных технологий, не соответствуют уровню требований современного общества. Активное внедрение информатики в учебный процесс по-

вышает роль студентов в образовательном процессе, их стремление найти новые эффективные способы получения и освоения информации [2]. При наличии широкого спектра систем и языков программирования содержание обучения в этой области должно, с одной стороны, дать представление о наиболее актуальных методах и инструментах программирования в настоящее время, с другой – предсказать будущие тенденции в развитии технологий.

Применение декларативных языков в качестве инструментальных средств позволяет существенно снизить сложность проектов и затраты времени, позволяя обучающимся уделить больше внимания изучению теории компиляции. Кроме того, использование языка Пролог позволяет закрепить и углубить знания по курсу «Функциональное и логическое программирование». Применение декларативных языков при проектировании трансляторов – тема не новая, но актуальная и в настоящее время. Последняя, зарубежная, публикация относится к концу 80-х гг. XX в. [10].

Целью исследования является изучение способов применения модели синтаксически управляемого перевода арифметических выражений средствами языка Пролог в практической реализации при выполнении бакалаврами лабораторных и курсовых работ по курсам «Функциональное и логическое программирование», «Теория компиляции» с использованием универсальных средств разработки [1], [3]. Большой объем программного кода, наличие в нем множества табличных элементов, сложных алгоритмов разбора приводят к тому, что основная масса обучающихся не успевает в течение семестра довести проекты до стадии готовности. Специальные инструментальные средства [4] не облегчают, а скорее запутывают ситуацию, более того, их сложно интегрировать в современные программные среды.

Материал и методика исследований. Для исследования данной проблемы использовались работы о методологических аспектах обучения алгоритмическим языкам программирования средствами декларативного языка Пролог, которые раскрывают модель синтаксически управляемого перевода простых арифметических выражений. Для выявления основных аспектов мы использовали теоретический анализ проблемы и предмета исследования, а также опыт преподавателей.

Результаты исследований и их обсуждение. При подготовке бакалавров факультета информатики и вычислительной техники с использованием модели синтаксически управляемого перевода арифметических выражений появляется ряд вопросов. В зависимости от уровня исходного языка различают ассемблеры и компиляторы. В качестве инструментальных средств реализации трансляторов обычно выбираются императивные (алгоритмические) языки. Программы, занимающиеся трансляцией (переводом) исходных программ, написанных на языках программирования, в машинные коды конкретной ЭВМ, называют трансляторами [9]. В данной статье предлагается использование декларативного языка Пролог. Основаниями для его выбора являются следующие:

– язык Пролог относится к декларативным языкам программирования, в которых программа представляется как отношения между группой объектов, событий; он предназначен для обработки символьной информации;

– предложения на данном языке по синтаксису напоминают продукции контекстно-свободных грамматик, что позволяет описывать продукции в естественной форме;

– встроенные механизмы языка (прямой трассировки, возврата) и метод рекурсивного программирования, принятые в Прологе, семантически согласуются с основными методами, используемыми в теории компиляции;

– одному предложению языка Пролог соответствуют 7–8 предложений на императивном языке [9].

В действительности, происхождение Пролога связано с попыткой использования логических высказываний для выражения грамматических правил в формализации процессов синтаксического разбора. Наиболее распространенным подходом к реализации

синтаксического разбора средствами Пролога является применение грамматик, задаваемых определительными предложениями (definite clause grammar, или DCG) [6], [7], [8].

Синтаксический анализатор является составной частью любого транслятора. Его основная задача состоит в переводе потока лексем, поступающих от лексического анализатора, в дерево разбора, или же в последовательность действий. В качестве примера разбираемого языка выбрано подмножество КС-грамматики для обработки простых арифметических операций. В общем случае КС-грамматика определяется конечным множеством продукций (правил) вида <нетерминал>→<тело>. Грамматику арифметических выражений можно представить в следующем виде [2]:

<E>→<E>+<T>{+}
<E>→<T>
<T>→<P>*<T>{*}
<T>→<P>
<P>→(<E>)
<P>→Id,

где E – начальный символ грамматики, обозначающий выражение; Id – произвольный идентификатор или числовая константа, например a, b, c, – 13, 5.68e3, etc.; {+},{*} – символы действия.

Символы действия представляют собой процедуры, выполняющие соответствующие им операции, которые легко реализуются средствами любого императивного языка. Высокая интеграция современных языков высокого уровня дает возможность делать вставки частей кодов, написанных на одном языке, в программы на другом, что позволяет существенно снизить трудоемкость программы, сохранив при этом ее ясность и корректность [10], [5]. Названия процедур должны раскрывать логику выполняемой операции, например, ADD, MULTIPLY, etc.

В соответствии с приведенной грамматикой, для исходной цепочки a+b*c должна быть сформирована последовательность действий: {a}{b}{c}{*}{+}, которые будут выступать как выходные символы синтаксического перевода.

С учетом наследуемых и синтезируемых атрибутов представленная грамматика примет следующий вид:

<E> – начальный символ
$<E>_x \rightarrow <E>_q + <T>_r \{ADD_{y,z,p}\}$
(x,p)←newtr y←q z←r
$<E>_x \rightarrow <T>_p$
x←p
$<T>_x \rightarrow <P>_q * <T>_r \{MULTIPLY_{y,z,p}\}$
(x,p)←newtr y←q z←r
$<T>_x \rightarrow <P>_p$
$<P>_x \rightarrow (<E>_p)$
x←p
$<P>_x \rightarrow Id_p$
x←p
$<E>_x$ синтезируемый атрибут x
$<T>_x$ синтезируемый атрибут x
$<P>_x$ синтезируемый атрибут x
$\{ADD_{y,z,p}\}$ наследуемые атрибуты y, z, p
$\{MULTIPLY_{y,z,p}\}$ наследуемые атрибуты y, z, p.

Процедура newtr вычисляет указатель на новый табличный элемент, отводимый под промежуточный результат. Таблицы идентификаторов, меток, промежуточных результатов в программе могут быть организованы как список или в виде двоичного дерева, различие между которыми только внешнее, текстуальное.

Схематическая модель синтаксически управляемого перевода для отдельных арифметических продукций данной грамматики определяется следующей Пролог-программой:

```
parse (Source, Tree) : – arithmetic_expression (Source),
arithmetic_expression (X) : – constant (X); identifier (X),
arithmetic_expression (Operator, X, Y)) : – constant (X),
arithmetic_operator (Operator), arithmetic_expression (Y),
arithmetic_operator (Op) : – Op = ‘+’.
arithmetic_operator (Op) : – Op = ‘*’.
Constant (X) : – integer_value (X).
Identifier (X) :– identifier_table (X, Ptr).
```

Основной предикат parse – синтаксический анализатор (парсер) – имеет два аргумента: Source – конечная последовательность лексем, сформированная на этапе лексической обработки; Tree – внутреннее дерево разбора, используемое для генерации ассемблерного мнемокода или объектного кода. Дерево разбора – это последовательность символов действий, которую формируют с помощью встроенных предикатов Пролога для работы с динамическими данными. Ниже приводятся примеры формирования некоторых последовательностей действий с помощью встроенного предиката assert, добавляющего в текущую программу (базу данных) новое предложение. Например, assert (integer_value (X): – X=value) добавляет в программу предложение integer_value, позволяющее переменной X сопоставить значение целочисленной константы value в том случае, если в исходном выражении встретилась целочисленная константа.

Следующие предложения формируют символ действия $\{ADD_{y,z,p}\}$ и правила вычисления наследуемых атрибутов y, z, p.

Скажем несколько слов о предикате arithmetic_operator (Op) : – Op = ‘+’, приведенном выше в модели программы. Этот предикат предполагает подстановку символа ‘+’ в выходной поток, но правил вычисления атрибутов в нем нет. Для его формирования необходимо ввести оператор следующего вида :

```
– (600, yfx, ‘add’),
```

где 600 – приоритет оператора; Yfx – левоассоциативный бинарный оператор; ‘add’ – имя оператора.

После чего вводится предикат, уточняющий и дополняющий исходную программу с учетом правил вычисления атрибутов:

```
assert (arithmetic_operator (Y, Z, P) : – Y is Z add P)),
```

где Y – наследуемый атрибут, который предназначен для хранения промежуточного результата вычисляемого выражения.

В качестве таблицы промежуточных результатов используется список, пополняющийся по мере вычислений. По окончании вычислений текущего выражения список должен быть пуст, иначе синтаксический анализатор формирует сообщение об ошибке с помощью предиката error. Работа со списком организуется через процедуру newtr, основная задача которой состоит в добавлении нового элемента в него в случае появления во входной последовательности лексем операции сложения и удалении информации из списка

в случае использования промежуточного значения в последующих вычислениях. Добавление нового элемента в список осуществляется следующим предикатом:

add_list (New_element, [New_element|List]).

Этот предикат добавляет элемент New_element в List в качестве новой головы списка. Преимущество такого подхода состоит в том, что из списка легче всего выделить именно голову, кроме того, как показывает практика, значение, вычисленное последним, требуется раньше остальных.

Предикат newtr по сути повторяет предикат add_list с тем условием, что в роли указателя на промежуточный результат выступает голова списка:

newtr (X, P): – add_list (X, [P|_]).

Предикат удаления промежуточного результата из списка является комплементарным по отношению к add_list, удаляющим голову списка, и определяется следующим отношением:

delete_list ([Head|Tail), Tail).

Атрибуты Z и P обозначают соответственно левый и правый операнды арифметического выражения. Их значения могут быть получены из таблицы либо идентификаторов, сформированной лексическим анализатором, либо промежуточных результатов, как описано выше. В случае извлечения значений из таблицы идентификаторов используется предикат identifier (X) : – identifier_table (X, Ptr), который по сути связывает переменную X с указателем Ptr. Таблица идентификаторов также организована как список, то есть как наиболее простая структура данных. Как вариант, таблицу можно организовать в виде двоичного дерева. Дерево может оказаться предпочтительнее для организации быстрого поиска значений или при ограниченных машинных ресурсах. Для тестовых испытаний модели без использования лексической обработки можно сформировать несложный прототип таблицы идентификаторов с помощью набора фактов, например:

table_ identifier (‘a’, Ptr_1).
table_ identifier (b’, Ptr_2).
table_ identifier (‘c’, Ptr_3). % etc,

где a, b, c – имена переменных; Ptr_N – указатели на относительные адреса переменных в памяти машины в период выполнения программы.

Символ действия $\{MULTIPLY_{y,z,p}\}$ формируется аналогичным образом, с той лишь разницей, что должна быть определена операция умножения: – (500, yfx, ‘multiply’), у которой выше приоритет и другое название.

Правило вычислений наследуемых атрибутов операции умножения задается следующим предикатом:

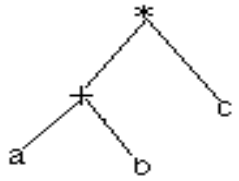
assert (arithmetic_operator (Y, Z, P): – Y is Z multiply P)).

В приведенных правилах операторы add и multiply символичны, в действительности, они не призваны выполнять действия, поэтому можно использовать общепринятые символы ‘+’ и ‘*’.

Добавленные в программу предложения, описывающие способы вычисления символов действия, рассматриваются как результат работы приведенной модели анализатора.

Несложно дополнить модель программы правилами разбора других арифметических операций.

Результатом работы программы, как было сказано ранее, является внутреннее дерево разбора (вывода), вершины которого представляют символы действия. Форма представления дерева вывода может быть различной: она определяется этапами трансляции. Например, для простого выражения a+b*c дерево вывода выглядит следующим образом:



Соответствующая ему последовательность действий представляется выражением {a}{b}{c}{*}{+}.

Возможен списковый вариант последовательности в виде предложения:

List=[a, b, c,*,+].

Открывающая и закрывающая фигурные скобки символов действия в силу их избыточности в состав списка не включены. Формирование такого списка – задача несложная, сводится в основном к простым операциям над списками.

В случае представления выходной последовательности в виде двоичного дерева его эквивалент запишется в следующем виде:

tree (*, tree (+, tree (a, void, void), tree (b, void, void)), tree (c, void, void)).

Такой формат представления кажется сложнее для восприятия, чем простой линейный список, однако рекурсивная структура Пролог-системы позволяет легко формировать, обработать и модифицировать подобные выражения.

Для проверки эффективности модели синтаксически управляемой трансляции на примере арифметических операций в процессе обучения был проведен эксперимент. Для этого были выбраны студенты 3 курса, изучающие дисциплины «Функциональное и логическое программирование» и «Теория компиляции». Выбор был обоснован следующими причинами: они изучают язык Пролог (и интеграция этой модели пройдет проще), уже имеют хорошую математическую вузовскую подготовку, что позволяет судить о примерно равном уровне их знаний. Контрольная и экспериментальная группы были сформированы из существующих групп курса. В каждую вошли по 20 человек. Обе группы получили одинаковые теоретические знания на лекциях. Разница заключалась в применении описанной модели на лабораторных занятиях в экспериментальной группе; контрольная группа выполняла те же задания, но без применения данной модели. После освоения дисциплин было проведено тестирование, которое состояло из 30 заданий: из них 10 были теоретическими, относились большей частью к лекционному курсу и частично – к лабораторным занятиям. Следующие 15 заданий представляли собой написанный на языке Пролог код. Тестируемым предлагалось исправить ошибку и/или написать вывод представленного кода. Оставшиеся 5 заданий требовали написания программы, решающей определенную задачу. Полученные в эксперименте результаты были сведены в таблице 1.

Таблица 1

Результаты тестирования

Задания	Контрольная группа, % верных ответов	Экспериментальная группа, % верных ответов
1 блок (10 заданий)	89	91
2 блок (15 заданий)	73	84
3 блок (5 заданий)	55	72

По результатам тестирования видно, что применение описанной модели улучшает понимание студентами теории и практики программирования по сравнению с теми, кто не знаком с этой моделью.

Резюме. В статье приведена модель синтаксически управляемой трансляции на примере арифметических операций, описаны способы ее реализации при выполнении бакалаврами лабораторных и курсовых работ, определены правила формирования атрибутов. При сопоставлении результатов исследования контрольной и экспериментальной групп было установлено, что во второй группе испытуемых уровень освоения материла изучаемых дисциплин с использованием модели повысился, что свидетельствует о ее эффективности.

ЛИТЕРАТУРА

1. *Альфред В. А., Моника С. Л., Рави С., Джефри Д. У.* Компиляторы. Принципы, технологии и инструментарий. – М. : Вильямс, 2016. – 1120 с.

2. *Герасимова А. Г.* Вопросы подготовки будущих учителей к использованию информационных и коммуникационных технологий в условиях информатизации образования // Актуальные проблемы методики обучения информатике в современной школе : сборник материалов Международной научно-практической интернет-конференции. – М., 2018. – С. 165–167.

3. *Марков В. Н.* Современное логическое программирование на языке Visual Prolog 7.5 : учебник. – СПб. : БХВ-Петербург, 2016. – 544 с.

4. *Обломов И. А.* Методологические аспекты преподавания декларативных языков программирования // Информатика и вычислительная техника : сборник научных трудов. – Чебоксары, 2016. – С. 138–142.

5. *Обломов И. А.* Объектно-ориентированное программирование : методические указания к курсовой работе. – Чебоксары : Изд-во Чуваш. ун-та, 2015. – 16 с.

6. *Солдатова О. П., Лезина И. В.* Логическое программирование на языке Visual Prolog : учебное пособие. – Самара : СНЦ РАН, 2010. – 81 с.

7. *Стерлинг Л., Шапиро Э.* Искусство программирования на языке Пролог. – М. : Мир, 1990. – 235 с.

8. *Фадеева К. Н., Герасимова А. Г.* Использование метода проектов как средства формирования ИКТ-компетентности бакалавров сервиса // Вестник Чувашского государственного педагогического университета им. И. Я. Яковлева. – 2016. – № 3(91). – С. 176–181.

9. *Шрайнер П. А.* Основы программирования на языке Пролог: курс лекций : учебное пособие. – М. : Интуит, 2016. – 176 с.

10. *Jacques C., Timothy J. H.* Parsing and compiling using Prolog // ACM Transactions on Programming Languages and Systems (TOPLAS). – 1987. – Vol. 9, Issue 2. – P. 125–163. – URL : https://dl.acm.org/citation.cfm?id=22946.

Статья поступила в редакцию 17.12.2019

REFERENCES

1. *Al'fred V. A., Monika S. L., Ravi S., Dzhefri D. U.* Kompilyatory. Principy, tekhnologii i instrumentarij. – M. : Vil'yams, 2016. – 1120 s.

2. *Gerasimova A. G.* Voprosy podgotovki budushchih uchitelej k ispol'zovaniyu informacionnyh i kommunikacionnyh tekhnologij v usloviyah informatizacii obrazovaniya // Aktual'nye problemy metodiki obucheniya informatike v sovremennoj shkole : sbornik materialov Mezhdunarodnoj nauchno-prakticheskoj internet-konferencii. – M., 2018. – S. 165–167.

3. *Markov V. N.* Sovremennoe logicheskoe programmirovanie na yazyke Visual Prolog 7.5 : uchebnik. – SPb. : BHV-Peterburg, 2016. – 544 s.

4. *Oblomov I. A.* Metodologicheskie aspekty prepodavaniya deklarativnyh yazykov programmirovaniya // Informatika i vychislitel'naya tekhnika : sbornik nauchnyh trudov. – Cheboksary, 2016. – S. 138–142.

5. *Oblomov I. A.* Ob"ektno-orientirovannoe programmirovanie : metodicheskie ukazaniya k kursovoj rabote. – Cheboksary : Izd-vo Chuvash. un-ta, 2015. – 16 s.

6. *Soldatova O. P., Lezina I. V.* Logicheskoe programmirovanie na yazyke Visual Prolog : uchebnoe posobie. – Samara : SNC RAN, 2010. – 81 s.

7. *Sterling L., Shapiro E.* Iskusstvo programmirovaniya na yazyke Prolog. – M. : Mir, 1990. – 235 s.

8. *Fadeeva K. N., Gerasimova A. G.* Ispol'zovanie metoda proektov kak sredstva formirovaniya IKT-kompetentnosti bakalavrov servisa // Vestnik Chuvashskogo gosudarstvennogo pedagogicheskogo universiteta im. I. Ya. Yakovleva. – 2016. – № 3(91). – S. 176–181.

9. *Shrajner P. A.* Osnovy programmirovaniya na yazyke Prolog: kurs lekcij : uchebnoe posobie. – M. : Intuit, 2016. – 176 s.

10. *Jacques C., Timothy J. H.* Parsing and compiling using Prolog // ACM Transactions on Programming Languages and Systems (TOPLAS). – 1987. – Vol. 9, Issue 2. – P. 125–163. – URL : https://dl.acm.org/citation.cfm?id=22946.

The article was contributed on December 17, 2019

Сведения об авторах

Обломов Игорь Александрович – кандидат технических наук, доцент кафедры вычислительной техники Чувашского государственного университета имени И. Н. Ульянова, г. Чебоксары, Россия; e-mail: ra4yes@rambler.ru

Ржавин Вячеслав Валентинович – кандидат технических наук, доцент кафедры вычислительной техники Чувашского государственного университета имени И. Н. Ульянова, г. Чебоксары, Россия; e-mail: grzhavv@gmail.com

Первова Наталия Викторовна – старший преподаватель кафедры вычислительной техники Чувашского государственного университета имени И. Н. Ульянова, г. Чебоксары, Россия; e-mail: nata4mail@mail.ru

Герасимова Алина Германовна – кандидат педагогических наук, доцент кафедры информатики и информационно-коммуникационных технологий Чувашского государственного педагогического университета им. И. Я. Яковлева, г. Чебоксары, Россия; e-mail: alina2902@mail.ru

Author information

Oblomov, Igor Aleksandrovich – Candidate of Technical Sciences, Associate Professor of the Department of Computer Engineering, I. Ulyanov Chuvash State University, Cheboksary, Russia; e-mail: ra4yes@rambler.ru

Rzhavin, Vyacheslav Valentinovich – Candidate of Technical Sciences, Associate Professor of the Department of Computer Engineering, I. Ulyanov Chuvash State University, Cheboksary, Russia; e-mail: grzhavv@gmail.com

Pervova, Natalia Viktorovna – Senior Lecturer, Department of Computer Engineering, I. Ulyanov Chuvash State University, Cheboksary, Russia; e-mail: nata4mail@mail.ru

Gerasimova, Alina Germanovna – Candidate of Pedagogics, Associate Professor of the Department of Informatics and Information and Communication Technologies, I. Yakovlev Chuvash State Pedagogical University, Cheboksary, Russia; e-mail: alina2902@mail.ru