УДК 004.4'422
DOI: 10.30987/article_5b5063e03ab429.41538229

А.А. Малявко

ЛЕКСИЧЕСКИЙ И СИНТАКСИЧЕСКИЙ АНАЛИЗАТОРЫ ТРАНСЛЯТОРА ЯЗЫКА EL

Рассматриваются задача реализации мультипарадигменного функционально-императивного языка программирования El, состав и структура его транслятора, краткое описание лексики и синтаксиса языка, основные функции лексического и синтаксического анализаторов транслятора. Приводятся фрагменты формальных определений лексики и синтаксиса языка. Описываются основные алгоритмы лексического и синтаксического анализаторов транслятора.

Ключевые слова: транслятор, лексический анализ, синтаксический анализ, формальные грамматики, системы регулярных выражений.

A.A. Malyavko

SCANNER AND SYNTACTIC ANALYZER OF EL LANGUAGE TRANSLATOR

This paper report the consideration of the problem in the realization of multi-paradigm functional-imperative El computer language, the composition and structure of its translator, a short description of vocabulary and language syntax, algorithms and basic functions of a scanner and a syntax analyzer of the translator. The elements of formal definitions of the vocabulary and language syntax are shown which were used for the automated formation of a scanner and a syntax analyzer of the translator in C++ with the use of client-server Webtranslab package.

Basic algorithms of the scanner are described which in this version of the translator performs macro-processing, realizes file inclusions, eliminates all insignificant symbol sequences and transforms correct language words into inner presentation – tokens. The structure of a syntax analyzer performing a top-down analysis and formed as a stack automatic machine with many states and controlled by a current input token and fields of a current state is under consideration. The composition of the cell of the automatic machine state and algorithms of its work and also mechanisms of the fulfillment of actions add-in into grammar realizing a functional of a semantic analyzer and a code generator are described. A current state in the realization of two constituents of El translator – a scanner and a syntax analyzer is described.

Investigation methods: a system analysis, computer-aided design, meta-languages for definition of lexical and syntax structures.

The investigation results are programs of a scanner and a syntax analyzer of the translator of new functional-imperative El.

The algorithms and programs described in the mentioned work check up syntactical correctness (and lexical one by means of scanner use) of input El program are constituents of the translator developed in NSTU of new functional-imperative El.

Key words: translator, lexical analysis, syntax analysis, formal grammars, systems of regular expressions.

Введение

Новые языки программирования появляются постоянно. Причин этому есть несколько. Во-первых, непрерывно совершенствуется аппаратура, появляются все новые технические возможности и расширяются существующие. Многие старые языки не позволяют эффективно использовать современное оборудование, для него необходимо создавать соответствующее программное обеспечение. Во-вторых, постоянно расширяются области применения компьютеров, что также требует появления адекватных средств программирования. В-третьих, можно сказать, что «аппетит приходит во время еды». Разрозненные технологические новинки, предоставляемые одними языками программирования, проникают в другие языки, при этом обычно меняется стиль мышления программистов. Разные парадигмы программирования, конкурируя между собой, одновременно друг друга обогащают.

Одним из новых мультипарадигменных языков является язык программирования El, предложенный в [1]. В нем реализованы такие отличительные характеристики функциональной парадигмы [2; 3], как чистота пользовательских функций, функции высшего порядка, операции сопоставления с образцом, возможная (вариативная) иммутабельность переменных, облегчающая тестирование и отладку про-

грамм. В то же время для языка характерно наличие ряда свойств, присущих императивной парадигме [4; 5]. Это последовательное исполнение операторов программы, возможная мутабельность переменных, примитивные и высокоуровневые типы данных, развитые управляющие структуры и т.д.

Для нового языка El, основным прототипом которого является Эрланг [6; 7], необходимо разработать исполняющую систему. Исполняющая система - это совокупность программных средств, предоставляющая пользователям (программистам) весь необходимый сервис по подготовке текстов программ, компиляции их на машинный язык или входной язык некоторой виртуальной машины, отладке и профилированию программ и т.д.

Основной составляющей частью исполняющей системы является транслятор. Для языка El на основе анализа его свойств и характеристик была выбрана схема двухэтапной трансляции с использованием уже существующего транслятора с языка ассемблера виртуальной машины LLVM [8] на втором этапе.

Для первого этапа должна быть выполнена разработка транслятора, преобразующего входную программу с языка El на язык ассемблера инфраструктуры разработки компиляторов LLVM.

Первая версия этого транслятора разрабатывается на языке C++, однако все его компоненты одновременно разрабатываются методом раскрутки (bootstrapping) [9] и на самом реализуемом языке El.

Структура транслятора

Разрабатываемый транслятор реализует пять логически связанных этапов:

1. Лексический анализ, преобразование входной программы в последовательность токенов (ПТ).

2. Синтаксический анализ, преобразование ПТ в постфиксную форму записи (ПФЗ).

3. Семантический анализ, формирование промежуточного псевдокода (ПК).

4. Оптимизация промежуточного кода.

5. Генерация объектного кода на языке ассемблера LLVM.

Технологически весь процесс трансляции управляется синтаксическим анализатором, из которого по мере необходимости вызываются функции, реализующие прочие этапы.

Лексический и синтаксический анализаторы транслятора для языка El были построены с использованием клиент-серверного пакета программ автоматизации проектирования трансляторов Вебтранслаб [10]. Для этого были разработаны совокупность регулярных выражений, определяющих лексику языка, и формальная грамматика, определяющая его синтаксис.

Лексический анализатор

Лексика языка El представляет собой систему регулярных выражений, содержащую правила определения более чем 110 токенов. Токены - это внутренние эквивалентные представления либо отдельных слов (ключевые слова, круглые, квадратные и фигурные скобки, знаки арифметических, логических и битовых операций, знаки операций присваивания / сопоставления с образцом, разделители, ограничители и т.д.), либо целых групп слов (атомы, идентификаторы, числовые и строковые литералы). Фрагмент системы системы регулярных выражений на входном языке пакета Вебтранслаб выглядит так:

```
ident : ([a-zA-Z][a-zA-Z_0-9]*) |
([_][a-zA-Z_0-9]+)
const : [0-9]+ ([.][0-9]*)?
const : [0-9]* [.][0-9]+
const : [0-9]+[.][0-9]*[eE][-+]?[0-
9]+
const : [1-9][0-9]?[$][0-9a-zA-Z]+
atom : ['][]+[']
string : ["][]*["]
addAssign : [+][=]
bitAndAssign : [$][&][=]
genHeadList : [<][?]
…
```

Лексический анализатор (далее - сканер) в самом первоначальном варианте был построен пакетом Вебтранслаб в виде

конечного автомата без памяти. Затем он был существенно переработан в связи с необходимостью реализации функций макропроцессора, обработки директив включения файлов и блочных вложенных комментариев. В настоящий момент сканер представляет собой функцию на языке C++, вызываемую синтаксическим анализатором и возвращающую одно целочисленное значение - код токена очередного слова, прочитанного из входного текста. Символьная последовательность прочитанного слова запоминается в специальной глобальной строковой переменной. Синтаксический анализатор, получив код токена, соответствующий целой группе слов (идентификаторы, атомы или строковые константы), может занести это слово в таблицу идентификаторов. Численные константы после обнаружения преобразуются во внутреннее представление для последующего хранения и обработки. Сканер является «жадным», т.е. входную цепочку символов «+++» он разобьет на слова так: «++»«+», а не так: «+»«++». Все незначащие последовательности символов входного текста, такие как разделительные пробелы, табуляции и комментарии, сканер игнорирует, не преобразуя в токены. Комментарии оформляются в стиле C/C++ и могут быть однострочными (от двух символов «//» до конца строки) и многострочными, т.е. блочными (между парами символов «/*» и «*/»). Блочные комментарии в отличие от многих других языков являются вложенными.

Сканер El-транслятора осуществляет и функции макропроцессора, обеспечивая распознавание и сохранение макроопределений, а также формирование и подстановку макрорасширений. Механизм макросов во многом похож на аналог в C/C++, но существует ряд отличий. Макроопределения начинаются с ключевого слова *macro* без символа «#» (как «#define»), которое должно быть первыми 5 символами строки. Далее должно следовать имя макроса со списком имен аргументов в круглых скобках через запятую. В этом списке допустимы табуляции и пробелы, он может быть пустым. После списка аргументов в этой же строке следует тело макроса. Если тело макроса очень длинное, то его можно переносить на следующие строки, используя в качестве знака переноса символ «\» непосредственно перед концом строки. В отличие от языка C в теле макроса без необходимости не следует употреблять лишние круглые скобки, потому что они могут быть восприняты как обозначение кортежа. Допускаются вложенные макросы, но при обнаружении рекурсивного макровызова первый из них не заменяется макрорасширением. Это блокирует бесконечное распространение рекурсии и может (но не обязательно) повлечь выдачу синтаксических или семантических ошибок в период компиляции для предложения, в котором содержится макровызов рекурсивно определенного макроса. Каких-либо сообщений об ошибках в макросах или предупреждений, связанных с некорректным использованием механизма макросов, лексический анализатор не формирует.

Сканер El-транслятора реализует и функционал вложения текстовых файлов в транслируемую программу. В любой точке текста модуля в качестве отдельной строки может появиться инструкция:

include <путь_и_имя_файла>

Ключевое слово *include*, так же как и слово *macro*, не содержит символа «#» и должно быть записано в самом начале строки. Текст файла вставляется вместо всей этой строки и далее обрабатывается как часть транслируемой программы. В указанный файл могут включаться другие файлы, при этом глубина вложенности не ограничена. В случае обнаружения попытки рекурсивного включения она блокируется с выдачей пользователю предупреждающего сообщения. Если во включенном тексте обнаружены какие-либо ошибки, то транслятор формирует диагностическое сообщение, содержащее все указанные в предложениях *include* имена включенных файлов вплоть до точки обнаружения ошибки и номера строк в них. Отсутствие включаемого файла по указанному пути не рассматривается как ошибка, но транслятор выдаст предупреждение.

Синтаксический анализатор

Синтаксический анализатор El-транслятора построен пакетом Вебтранслаб путем преобразования грамматики языка в C++-программу. Грамматика относится к классу контекстно-свободных LL(1)-грамматик [9] и во внешнем пользовательском представлении содержит немногим более 120 правил. Во внутреннем представлении пакета Вебтранслаб после преобразования в форму без регулярных выражений она содержит более 320 правил. Фрагмент текущего пользовательского вида этой грамматики с использованием регулярных выражений выглядит так:

```
module : moduleName [ option ]? [ import ]* [ initialyzer ]* [ functionDef ]+
moduleName : "module" ident [ "." ident ]* ";"
option : "options" "(" options ")" ";"
import : "import" ident [ ("." | ",") ident ]* ";"
initialyzer : [ tuple [ guard ]? ]? block
initialyzer : "record" ident "(" options ")" ";"
functionDef : [access ]? ident argTuple bodyFunction
access : "public" | "private"
argTuple : "(" [ argTupleElem [ "," argTupleElem ]* ]? ")"
argTupleElem : varDef [ varType ]?  ident
argTupleElem : varType ident
argTupleElem : expression
bodyFunction : guard block restBody
bodyFunction : block
…
```

В это представление грамматики не включены действия по формированию промежуточного представления транслируемой программы и по генерации кода на языке ассемблера LLVM.

Синтаксический анализатор транслятора (парсер) реализован как нисходящий стековый автомат с несколькими состояниями [11], управляемый текущим входным токеном и полями клетки текущего состояния. Количество состояний автомата, построенного по приведенной выше грамматике языка El, составляет немногим более 950.

Управляющая таблица автомата является одномерной, каждая ее клетка соответствует одному состоянию и содержит поля, показанные на рисунке.

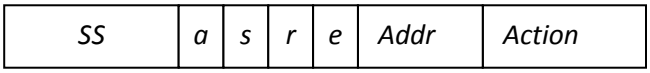

Рис. Состав клетки управляющей таблицы

Обозначения полей клетки имеют следующий смысл.

SS - множество выбора состояния. Это экземпляр вспомогательного класса *SelSet*, содержащий все токены, допустимые на входе парсера в данном состоянии. В этой реализации автомата множества выбора определены не для всех состояний. Они вычислены построителем Вебтранслаба только для состояний, которые соответствуют нетерминалам из левых частей грамматических правил. В состояниях, соответствующих терминальному символу (они могут находиться только в правых частях правил), текущий входной токен должен совпадать с этим символом. В противном случае парсер фиксирует синтаксическую ошибку. В состояниях, соответствующих нетерминальным символам из правых частей правил, множество выбора отсутствует, т.е. допустимость текущего входного токена не проверяется. Это будет выполнено в течение ближайших нескольких шагов работы автомата при переборе порождающих правил для данного нетерминала [11]. Такая особенность структуры управляющей таблицы автомата не влияет на способность парсера обнаруживать син-

таксические ошибки, но несколько ускоряет его работу.

a, s, r и *e* - флажки, управляющие поведением автомата. Их значения для каждого состояния сформированы преобразователем грамматики в автомат.

Флажок *a* (от английского *accept*) заставляет парсер вызвать сканер для чтения следующего токена. Этот флажок устанавливается в единицу в состояниях, соответствующих терминалам из правых частей правил.

Флажок *s* (от английского *stack*) обеспечивает сохранение номера текущего состояния, увеличенного на единицу, в стеке парсера. Построителем этот флажок устанавливается в единицу для всех состояний, соответствующих нетерминалам в правой части правила.

Флажок *r* (от английского *return*) приводит к переключению автомата в состояние, номер которого снимается с верхушки стека. Этот флажок устанавливается построителем в единицу в состояниях, соответствующих последним символам каждого правила. Если построитель обнаруживает, что в некотором состоянии оказались установлены в единицу и флажок *r* и флажок *s*, то он сбрасывает значение обоих флажков в ноль. Не имеет смысла заносить адрес возврата на верхушку стека только для того, чтобы немедленно удалить его из стека.

Флажок *e* (от английского *error*) означает необходимость подавления останова по ошибке. Кажущееся обнаружение синтаксической ошибки может возникнуть при переборе правил для разбора нетерминала в том случае, если парсер еще не дошел до нужного правила. Построителем автомата этот флажок устанавливается для всех состояний, соответствующих одному и тому же нетерминалу из левой части, кроме последнего правила.

Addr - адрес перехода. Это номер состояния, в которое должен перейти автомат, если в данном состоянии не установлен флажок *r*.

Action - указатель на функцию, телом которой является действие - фрагмент программы, записанной в правиле грамматики. Действия расширяют функциональность парсера или реализуют функционал семантического анализа, оптимизатора или генератора кода. Если значение *Action* не равно *NULL*, то эта функция вызывается, в противном случае ничего не делается.

Парсер, как стековый автомат, запускается, находясь в нулевом состоянии при пустом стеке. На его входе в этот момент находится первый токен, прочитанный из текста транслируемой программы. Последующая работа автомата описывается алгоритмом, включающим следующие шаги и выполняемым в каждом новом состоянии.

Шаг 1. Проверяется множество выбора текущего состояния *SS*. Если оно является пустым, то выполняется переход к шагу 2, иначе выполняется проверка, входит ли в это множество текущий токен. Если входит, то выполняется переход к шагу 2. В противном случае проверяется значение флажка *e*. Если он установлен в единицу, то автомат переключается в состояние, номер которого равен номеру текущего плюс 1. Для нового состояния заново выполняется весь алгоритм начиная с шага 1. Если же значение флажка *e* равно нулю, то фиксируется синтаксическая ошибка и формируется диагностическое сообщение. Затем парсер переходит в режим нейтрализации синтаксических ошибок, который будет рассмотрен в отдельной публикации.

Шаг 2. Проверяется значение флажка *a*. Если он установлен, то сравниваются текущий токен и токен терминала, для которого построено это состояние. Если они совпадают, то вызывается сканер для чтения следующего токена и выполняется переход к шагу 3. Иначе формируется диагностическое сообщение об ошибке, парсер переходит в режим нейтрализации синтаксических ошибок.

Шаг 3. Проверяется значение флажка *s*, и если он установлен, то на верхушку стека заносится номер текущего состояния, увеличенный на единицу. Проверяется значение флажка *r*, и если он установлен, то номер следующего состояния снимается с верхушки стека (если в этот момент стек оказался пустым, то выполняется переход к шагу 4). Иначе из текущей клетки выбирается адрес перехода в качестве номера следующего состояния. Автомат переключается в новое состояние, выполняется возврат к шагу 1.

Шаг 4. Автомат останавливается. Если текущий токен есть *EndOfFile* (т.е. к этому моменту прочитан весь входной текст), то транслируемая программа является правильной. Иначе формируется диагностическое сообщение о синтаксической ошибке, но переключение в режим нейтрализации синтаксических ошибок не производится, поскольку никаких новых ошибок после конца текста не может быть обнаружено.

Заключение

Описанные в данной работе алгоритмы проверяют синтаксическую (и лексическую - путем использования сканера) правильность входной программы на языке El. Ввиду того что разработка транслятора ведется с использованием средств автоматизации проектирования, для расширения функциональности синтаксического анализатора и реализации остальных этапов процесса трансляции достаточно просто расширять грамматику языка. В нее вставляются действия - фрагменты программ на языке C++, оформляемые построителем автоматов пакета Вебтранслаб в виде функций, указатели на которые размещаются в клетках управляющей таблицы автомата. При работе синтаксического анализатора вызовы этих функций будут выполняться по мере движения автомата от состояния к состоянию по ходу проверки правильности транслируемой программы. Тем самым одновременно с процессами лексического и синтаксического анализа реализуются все последующие этапы: преобразование входной программы в постфиксную форму записи и генерация псевдокода, семантический анализ, различные виды оптимизаций, генерация кода на языке ассемблера LLVM [12; 13].

Транслятор языка El в настоящее время разрабатывается на кафедре вычислительной техники Новосибирского государственного технического университета силами группы студентов и преподавателей и после завершения отладки будет представлен как свободное программное обеспечение с открытым исходным кодом.

СПИСОК ЛИТЕРАТУРЫ

1. Малявко, А.А. Функционально-императивный язык программирования El / А.А. Малявко // Научный вестник НГТУ. - 2018. - № 1 (70). - С. 117-136.
2. Wadler, P. Why no one uses functional languages / P. Wadler // ACM SIGPLAN Notices. - 1998.
3. Loder, W. Erlang and Elixir for Imperative Programmers / W. Loder. - Apress, 2016.
4. Кауфман, В.Ш. Языки программирования. Концепции и принципы / В.Ш. Кауфман. - М.: ДМК-Пресс, 2011. - 464 с.
5. Себеста, Р. Основные концепции языков программирования / Р. Себеста. - М.: Вильямс, 2001.
6. Armstrong, J. Programming Erlang: Software for a Concurrent World / J. Armstrong. - 2nd ed. - The Pragmatic Bookshelf, Dallas, USA, 2013.
7. Чезарини, Ф. Программирование в Erlang / Ф. Чезарини, С. Томпсон. - М.: ДМК-Пресс, 2012. - 487 с.
8. Лопес, Б. LLVM: инфраструктура для разработки компиляторов / Б. Лопес, Р. Аулер. - М.: ДМК-Пресс, 2015. - 342 с.
9. Ахо, А. Компиляторы. Принципы, технологии, инструменты / А. Ахо, Р. Сети, Д. Ульман. - СПб.: Вильямс, 2001. - 767 с.
10. Малявко, А.А. Использование веб-приложений и веб-технологий при разработке учебного программного обеспечения для изучения методов трансляции / А.А. Малявко // Современное образование: технические университеты в модернизации экономики России: материалы науч.-метод. конф. – Томск: Изд-во ТУСУР, 2011. - С. 45-47.
11. Малявко, А.А. Формальные языки и компиляторы: учеб. пособие для вузов / А.А. Малявко. - М.: Юрайт, 2017. - 429 с.
12. Сен, А. Создание действующего компилятора с помощью инфраструктуры LLVM. Ч. 1 / А. Сен. - URL: https://www.ibm.com/developerworks/ru/library/os-createcompilerllvm1/index.html (дата обращения: 04.12.2017).
13. Сен, А. Создание действующего компилятора с помощью инфраструктуры LLVM. Ч. 2 / А. Сен. - URL: https://www.ibm.com/developerworks/ru/library/os-createcompilerllvm1/index.html (дата обращения: 04.12.2017).

1. Malyavko, A.A. El- functional-imperative programming language / A.A. Malyavko // *Scientific Bulletin of NSTU.* – 2018. – No.1 (70). – pp. 117-136.
2. Wadler, P. Why no one uses functional languages / P. Wadler // ACM SIGPLAN Notices. - 1998.
3. Loder, W. Erlang and Elixir for Imperative Programmers / W. Loder. - Apress, 2016.
4. Kaufman, V.Sh. *Programming Languages. Concepts and Principles* / V.Sh. Kaufman. – M.: DMK-Press, 2011. – pp. 464.
5. Sebesta, R. *Basic Concepts of Programming Languages* / R. Sebesta. – M.: Williams, 2001.
6. Armstrong, J. Programming Erlang: Software for a Concurrent World / J. Armstrong. - 2nd ed. - The Pragmatic Bookshelf, Dallas, USA, 2013.
7. Chezarini, F. *Programming in Erlang* / F. Chezarini, S. Tompson. – M.: DMK-Press, 2012. – pp. 487.
8. Lopes, B. LLVM: *Infrastructure for Compiler Development* / B.Lopes, R. Auler. – M.: DMK-Press, 2015. - pp. 342.
9. Akho, A. *Compilers. Principles, Technologies, Tools* / A. Akho, R. Sety, D. Ulman. – S-Pb.: Williams, 2001. – pp. 767.
10. Malyavko, A.A. Use of Web-applications and web-technologies at development of training software to study translation medthods / A.A. Malyavko // *Modern Education: Engineering Universities in Russian Economy Updating: Proceedings of the Scientific Methodical Conf.* – Tomsk: Publishing House of TUSUR, 2011. – pp. 45-47.
11. Malyavko, A.A. *Formal Languages and Compilers*: manual for colleges / A.A. malyavko. – M.: Uright, 2017. – pp. 429.
12. Sen, A. *Formation of Functioning Compiler Using LLVM Infrastructure*. Part 1 / A. Sen. - URL: https://www.ibm.com/developerworks/ru/library/os-createcompilerllvm1/index.html (address date: 04.12.2017).
13. Sen, A. *Formation of Functioning Compiler Using LLVM Infrastructure*. Part 2 / A. Sen. - URL: https://www.ibm.com/developerworks/ru/library/os-createcompilerllvm1/index.html (address date: 04.12.2017).

Статья поступила в редколлегию 22.05.18.
Рецензент: д.т.н., профессор, зав. лабораторией синтеза ИВМиМГ СО РАН
Малышкин В.Э.

Сведения об авторах:

Малявко Александр Антонович, доцент Новосибирского государственного технического университета, тел. 8(383)346-04-92, e-mail: a.malyavko@corp.nstu.ru.

Malyavko Alexander Antonovich, Assistant Prof., Novosibirsk State Technical University, e-mail: a.malyavko@corp.nstu.ru.