УДК 004.4’422

Компилируемый язык описания программ для эмулятора машины Тьюринга

Л.О. Воробьёв
Донецкий национальный технический университет
lev4411@gmail.com

Воробьёв Л.О., Компилируемый язык описания программ для эмулятора машины Тьюринга. *В статье рассматривается язык программирования для описания алгоритмов на машине Тьюринга. Проанализированы существующие способы записи программы для машины Тьюринга. Описывается синтаксис разработанного языка описания программ и обоснована эффективность его применения в разработке программного обеспечения.*

Ключевые слова: *синтаксический анализ, компиляция, эмулятор, машина Тьюринга, форма Бэкуса-Наура*

Введение

Цель работы: создание специализированного компилируемого языка описания программ для многоленточных и одноленточных машин Тьюринга, последующая разработка и реализация компилятора для созданного языка и написание эффективных алгоритмов интерпретации скомпилированных программ.

Машина Тьюринга — абстрактная модель вычислительной машины, состоящая из бесконечной ленты и головки, которая двигается вдоль ленты, считывает и записывает содержимое ячеек, и может принимать различные состояния [1, с. 83]. В общем случае, существуют также многоленточные машины Тьюринга (в англоязычной литературе — multitape Turing machines [2, с. 126]), которые состоят из более чем одной ленты, перемещающихся независимо друг от друга.

Программой для машины Тьюринга называется конечный набор команд [1, с. 84], математическая запись которых имеет вид:

$$q_s a_i \to q_t a_j d \tag{1}$$

где q_s, q_t — номера состояний; a_i, a_j — содержимое ячейки; d — направление движения ленты.

Команда, записанная в формуле (1), означает, что если головка машины Тьюринга находится в состоянии q_s, а на ленте записан символ a_i, то головка меняет свое состояние на q_t, на ленту записывается символ a_j, и лента перемещается в направлении d.

Программы для многоленточных машин Тьюринга содержат команды, которые имеют вид:

$$q_s\,(a_{i1},\, a_{i2},\, \ldots a_{in}) \to q_t\,(a_{j1},\, a_{j2},\, \ldots a_{jn})\,d \tag{2}$$

где: a_{ik}, a_{jk} — содержимое ячеек на ленте под номером k; n — количество лент.

Машины Тьюринга используется во множестве теоретических и практических исследованиях. К примеру, в статье [3] описано использование машины Тьюринга для реализации алгоритмов симметричного шифрования.

Предлагаемая программная система может использоваться, как средство эмуляции многоленточных и одноленточных машин Тьюринга путем написания программ на специализированном языке программирования. Разработанный эмулятор машины Тьюринга имеет открытый API-интерфейс для языков C и C++, что позволяет использовать созданный язык программирования в разработке прикладного программного обеспечения.

Использование нового компилируемого языка программирования позволит ускорить работу программного обеспечения, использующего механизм машины Тьюринга.

Анализ существующих решений

Программная система, которая выполняет команды вида (1) и (2) называется эмулятором машины Тьюринга. Математической моделью такой системы можно считать универсальную машину Тьюринга [1, с. 87].

На сегодняшний день существует большое количество эмуляторов машины Тьюринга, которые используют различный синтаксис текстовых файлов для представления программ.

Программа [4], написанная на языке Паскаль, имитирует работу одноленточной машины Тьюринга. При этом программы для эмуляции задаются

в виде таблицы переходов, каждая клетка которой состоит из трех частей:

1. символ из заданного алфавита или знак "_"для обозначения «пустого» символа;
2. направление перемещения: > (вправо), < (влево) или . (на месте);
3. новое состояние автомата [4].

Существует онлайн-сервис [5], предоставляющий возможность выполнения программ для одноленточной машины Тьюринга. Программа задается списком команд, наиболее близких к математической записи вида (1) и (2).

Программа JFLAP позволяет имитировать работу однолетночной и многоленточной машины Тьюринга [2, с. 126]. При ближайшем рассмотрении можно заметить, что программы для машины Тьюринга задаются в виде графа переходов, который вводится пользователем в интерактивном режиме.

Постановка задачи

В данной работе необходимо спроектировать синтаксис языка описания программ для многоленточных и одноленточных машин Тьюринга. Полученную грамматику следует использовать для построения компилятора, транслирующего программу на формальном языке в инструкции для эмулятора машины Тьюринга.

Разработка синтаксиса для нового формального языка представляет собой неординарную задачу. Следует учесть несколько факторов, которые влияют на качество проектируемого синтаксиса:

- стандартизация — синтаксис языка должен унаследовать стандартные конструкции из других языков программирования [6, с. 48];
- ясность, простота единообразие понятий языка [6, с. 38];
- обеспечение простоты чтения и написания программы [6, с. 94].

Новый синтаксис должен решить ряд проблем, связанных с описанием программы для машины Тьюринга:

1. Команды могут не использовать запись на ленту нового значения, могут не перемещать ленту, не использовать переход в другое состояние.
2. Программа может содержать несколько состояний, для которых команды отличаются лишь несколькими значениями.
3. Программа может содержать команды, выполняющие одно и то же действие при различных наборах значений на лентах.

Решение данных проблем позволит сократить описание программ для машины Тьюринга.

Синтаксис и семантика языка

Описание синтаксиса языка можно представить при помощи БНФ — формы Бэкуса-Наура [7, с. 17]. Язык описания программ для машины Тьюринга имеет синтаксис, заданный в виде БНФ на рис. 1.

Ключевые слова `turing`, `template`, `begin`, `end`, `state`, `dir`, `char`, `nul`, `left`, `right`, `and`, `or`, `write`, `move`, `goto` являются нечувствительными к регистру.

В качестве идентификатора (`<identifier>`) может выступать любая последовательность букв, цифр, и знаков «_». Начало идентификатора не может быть цифрой или знаком «$».

Строка (`<string>`) — любая последовательность символов, заключенная в двойные кавычки.

Символьная константа (`<character-constant>`) — символ или его шестнадцатеричное представление, заключенное в одинарные кавычки. Запись символа в шестнадцатеричном представлении имеет вид: `'\xHH'`, где `HH` — шестнадцатеричное число. Символ «\» Задается с помощью конструкции `'\\'`. Последовательности `'\"'`, `'\''` позволяют задавать символы кавычек внутри строк и символьных констант.

Номер ленты (`<tape-number>`) — последовательность цифр, записанных после знака «$».

Программа для машины Тьюринга, записанная с помощью заданного синтаксиса представляет собой определение символьных множеств (`<set-declaration>`) и описание правил перехода (`<transition-rule>`), сгруппированных в состояния (`<state-declaration>`). Описание правил перехода состоят из условия (`<conditional-expression>`) и спаиска операторов (`<statement>`).

Если состояния имеют похожие правила перехода (проблема 2, см. стр. 2), тогда можно использовать шаблоны (`<template-declaration>`), которые описивают правила перехода используя параметры из списка (`<parameter-list>`). В таком случае определение состояния будет содержать символ «`:`», идентификатор шабона и список фактических параметров (`<initialiser-list>`).

Проблема, когда при различных наборах значений на лентах автомат выполняет одинаковые действия (проблема 3, см. стр. 2), решается путем использования условных выражений (`<confitional-expression>`). К примеру, если одно и то же правило перехода используется, когда на двух лентах записаны одинаковые значения из множества `alpha`, то условное выражение может иметь вид: `(alpha, alpha) and $2 == $1`.

В случае, если команда не меняет значение на ленте, не передвигает ленту или не переводит автомат в новое состояние (проблема 1, см. стр. 2), то соответствующий оператор (`<statement>`) может быть опущен.

```
<translation-unit> ::= {<external-declaration>}*
<external-declaration> ::= <turing-definition>
<turing-definition> ::= turing <identifier> { {<set-declaration>}* {<turing-declaration>}+ }
<set-declaration> ::= <identifier> = <set-constant>
<set-constant> ::= <string>
<turing-declaration> ::= <state-declaration>
                       | <template-declaration>
<template-declaration> ::= template <identifier> ( <parameter-list> ) {<transition-rule>}+
<state-declaration> ::= {begin}? state <identifier> {<transition-rule>}+
                      | {begin}? state <identifier> : <identifier> ( <initializer-list> )
<parameter-list> ::= <parameter-declaration>
                   | <parameter-list> , <parameter-declaration>
<parameter-declaration> ::= <type-specifier> <identifier>
<type-specifier> ::= dir | char | state
<initializer-list> ::= <initializer>
                     | <initializer-list> , <initializer>
<initializer> ::= <character-expression>
                | <direction-expression>
                | <state-expression>
<character-expression> ::= <character-constant>
                         | <number>
                         | <identifier>
                         | <tape-number>
                         | nul
<set-expression> ::= <set-constant>
                   | <identifier>
<direction-expression> ::= <direction-constant>
                         | <identifier>
<direction-constant> ::= left | right
<state-expression> ::= <identifier> | end
<conditional-expression> ::= <or-expression>
<or-expression> ::= <and-expression>
                  | <or-expression> or <and-expression>
<and-expression> ::= <equality-expression>
                   | <and-expression> and <equality-expression>
<equality-expression> ::= <relational-expression>
                        | <tape-number> == <character-expression>
                        | <tape-number> != <character-expression>
<relational-expression> ::= <primary-expression>
                          | <tape-number> < <character-expression>
                          | <tape-number> > <character-expression>
                          | <tape-number> <= <character-expression>
                          | <tape-number> >= <character-expression>
<primary-expression> ::= <tape-expression>
                       | ( <tape-list> )
<tape-list> ::= <tape-expression>
              | <tape-list> , <tape-expression>
<tape-expression> ::= <character-expression>
                    | <set-expression>
<transition-rule> ::= <conditional-expression> : {<statement>}*
<statement> ::= <write-statement>
              | <move-statement>
              | <goto-statement>
<write-statement> ::= write <character-expression> ;
                    | <tape-number> . write <character-expression> ;
<move-statement> ::= move <direction-expression> ;
                   | <tape-number> . move <direction-expression> ;
<goto-statement> ::= goto <state-expression> ;
```

Рисунок 1 — Синтаксис языка описания машины Тьюринга в форме Бэкуса — Наура

Описание структуры разработанной системы

Разработанная система состоит из компилятора `mtrc`, написанного на языке C++, и интерпретатора правил перехода, выполненного в виде статической библиотеки (`statestep.lib`), написанной частично на языке C и на Ассемблере.

Компилятор `mtrc` — это программа, которая считывает текст программы, записанный на языке описания машин Тьюринга, и транслирует (переводит) его в эквивалентную программу для интерпретатора правил перехода (рис. 2). В процессе трансляции, компилятор может сообщать пользователю о налиции ошибок в исходной программе [8].

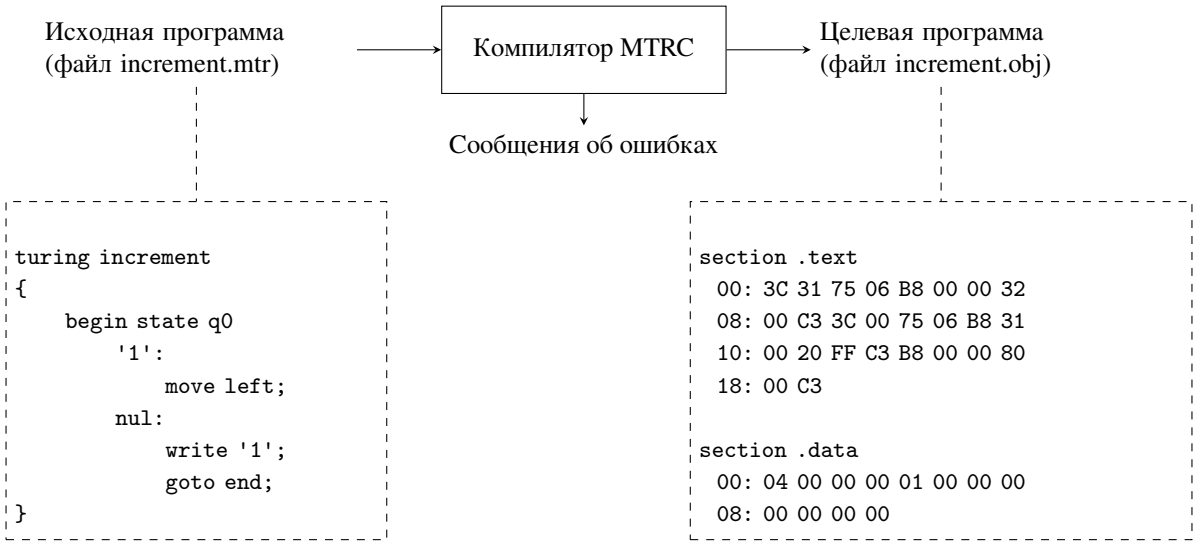

Рисунок 2 — Компилятор `mtrc`

В качестве примера для исходной программы использована машина Тьюринга, которая прибавляет 1 к числу в унарном коде. Математическая запись команд этой машины Тьюринга имеет вид:

$$\begin{aligned} q_1 1 &\to q_1 1 L; \\ q_1 \lambda &\to q_z 1 N. \end{aligned} \tag{3}$$

Компилятор создает объектный файл, содержащий секции `.data` и `.text`. Секция `.data` содержит таблицу состояний, которая представляет собой массив указателей на процедуры из секции `.text`.

В данном примере компилятор создал одну процедуру для состояния, которая содержится в секции `.text`. Семантика этой процедуры может быть описана с помощью ассемблерных команд и языка C (табл. 1).

Таблица 1 — Дизассемблирование скомпилированного объектного модуля

Машинный код	Ассемблер x86	Язык C

00: 3C 31
02: 75 06
04: B8 00 00 32 00
09: C3
0A: 3C 00
0C: 75 06
0E: B8 31 00 20 FF
13: C3
14: B8 00 00 80 00
19: C3
	cmp al, 31h
	jne .e1
	mov eax, 00320000h
	ret
	.e1: cmp al, 0
	jne .e2
	mov eax, 0FF200031h
	ret
	.e2: mov eax,00800000h
	ret
		switch(al) {
		case '1':
		return 0x00320000;
		case 0:
		return 0xFF200031;
		default:
		return 0x00800000;
		};

Процедура, созданная компилятором, возвращает правило перехода, которое зависит от значения регистра AX. В этот регистр интерпретатор должен записывать символы, содержащийся на лентах в текущем состоянии машины.

В реализации эмулятора машины Тьюринга использованы эффективные алгоритмы, записанные на языке Ассемблера [9], который существенно повышают скорость выполнения программы [10].

Для использования программы, написанной на данном языке программирования необходимо написать простую программу на языке C, использующую эмулятор машины Тьюринга.

Выводы

Разработанный язык описания программ для эмулятора машины Тьюринга не является полноценным языком программирования, однако он может быть использован в программных разработках вместе с языками C/C++, C#, и другими.

Применение разработанного языка позволит ускорить работу программного обеспечения, которое использует механизм машины Тьюринга.

Литература

1. А. К. Гуц. Математическая логика и теория алгоритмов: Учебное пособие. [Текст] / А. К. Гуц. — Омск : Издательство Наследие. Даилог-Сибирь, 2003. — 108 с.
2. Rodger, S. H. JFLAP: An Interactive Formal Languages and Automata Package. [Text] / S. H. Rodger, T. W. Finley. — [S. l.] : Jones and Bartlett, 2006. — 192 p.
3. М. М. Чернушко. Применение машины Тьюринга для реализации алгоритмов шифрования [Текст] / М. М. Чернушко // Технические науки: теория и практика: материалы II междунар. науч. конф. (г. Чита, январь 2014 г.). — [Б. м.] : Издательство Молодой ученый, 2014. — С. 19–22.
4. К. Поляков. Учебная модель компьютера «Машина Тьюринга» [Текст]. — [Б. м. : б. и.], 2016. — Режим доступа: http://kpolyakov.spb.ru/prog/turing.htm (дата обращения: 06.10.2016).
5. А. С. Балюк. Детерминированная машина Тьюринга [Текст]. — [Б. м. : б. и.], 2007. — Режим доступа: http://matinf.igpu.ru/simulator/tm.html (дата обращения: 06.10.2016).
6. Пратт Т. Языки программирования: разработка и реализация [Текст] / Пратт Т., Зелковиц М. ; Под ред. А. Матросова. — СПб. : Питер, 2002. — 688 с.
7. В. А. Серебряков. Основы конструирования компиляторов [Текст] / В. А. Серебряков, М. П. Галочкин. — [Б. м.] : Едиториал УРСС, 2001. — 192 с.
8. А. Ахо. Компиляторы: принципы, технологии и инструменты [Текст] / А. Ахо, Р. Сети, Дж. Ульман. — М. : Издательский дом «Вильямс», 2003. — 768 с.
9. Ю. С. Магда. Ассемблер для процессоров Intel Pentium. [Текст] / Ю. С. Магда. — СПб. : Питер, 2006. — 410 с.
10. Ю. С. Магда. Ассемблер. Разработка и оптимизация Windows-приложений. [Текст] / Ю. С. Магда. — СПб. : БХВ-Петербург, 2003. — 544 с.

Воробьёв Л.О., Компилируемый язык описания программ для эмулятора машины Тьюринга. *В статье рассматривается язык программирования для описания алгоритмов на машине Тьюринга. Проанализированы существующие способы записи программы для машины Тьюринга. Описывается синтаксис разработанного языка и обоснована эффективность его применения в разработке программного обеспечения.*

Ключевые слова: *синтаксический анализ, компиляция, эмулятор, машина Тьюринга, форма Бэкуса-Наура*

Vorobyov L.O., The compiled programming language for emulator of Turing mashines. *The article deals with the programming language to describe the algorithm on a Turing machine. Analyzed the existing methods of recording a program for a Turing machine. It describes the syntax of the developed language and proved its efficiency in software development*

Keywords: *parsing, compilation, emulator, the Turing machine, Backus-Naur Form.*