**Содержание**

Введение

1. Постановка задачи

2. Математические и алгоритмические основы решения задачи

2.1 Понятие конечного автомата

2.2 Способы описания

2.3 Детерминированность

2.4 Автоматы и регулярные языки

3. Функциональные модели и блок-схемы решения задачи

4. Программная реализация решения задачи

5. Пример выполнения программы

Заключение

Список использованных источников и литературы

**Введение**

В качестве примера, хорошо поясняющего сущность конечного автомата, часто приводят автомат для продажи каких-то товаров. Опускаемые в автомат монеты соответствуют записываемым входным символам. Сам автомат хранит в памяти данные о полученной до настоящего момента общей сумме, к которой будет добавляться новая сумма, состоящая из только что опущенных в автомат монет. Эта хранящаяся в памяти прежняя сумма соответствует состоянию конечного автомата. Когда в автомат опускаются деньги, составляющие цену билета, порции кофе и т.д., срабатывает механическое устройство, выдающее покупателю товар вместе со сдачей, и опущенная в прорезь автомата сумма сбрасывается в Нуль.

Современные автоматы не просто продают какой-то один вид товара. В них сделаны специальные окошечки, в которых высвечивается опущенная в прорезь автомата сумма. Указываемая в окошке сумма соответствует состоянию, набор монет соответствует входному символу, а выдаваемый автоматом товар вместе со сдачей – выходному символу. Если даже бросить монеты в автомат наугад, но сумма превысит определенную величину, товар будет выдан вместе со сдачей. Брошенная в автомат сумма должна иметь верхний предел, то есть не может быть неограниченно большой. Таким образом, число состояний, число входных символов (число комбинаций монет) и число выходных символов (число выдаваемых разновидностей товара вместе со сдачей) конечны.

Как только автомат установлен, он включается в сеть обслуживания, которая следит за регулярным изъятием денег и пополнением автомата товаром. Поэтому время работы автомата можно считать неограниченным. Пока автомат исправен, он находится на учете, и полный срок его службы в общем непредсказуем.

Устройство выполнения операций, устройство управления и оперативное (основное) запоминающее устройство, входящие в состав машины Тьюринга, делают ее машиной с ограниченным числом состояний, то есть самым настоящим конечным автоматом. Другими словами, при запоминании конечного числа каких-то элементов можно характеристики этих элементов представить в виде набора состояний конечного автомата. Бесконечное число состояний или, говоря языком логики, сколь угодно большое наперед заданное число состояний запомнить невозможно.

В конечном автомате нельзя запомнить число состояний, большее числа, заранее заданного для этого автомата, которое определяется параметрами рабочих лент. Хотя число разновидностей символов, используемых на лентах, является конечным, число ячеек может быть бесконечным, как бесконечен сам натуральный ряд чисел. Как в двоичной, так и в десятичной системах счисления на рабочих лентах можно записать последовательность цифр с требуемым числом разрядов и, таким образом, запомнить любое самое большое число данной разрядности.

Целью данной курсовой работы является ЛИСП-реализация конечных автоматов.

**1. Постановка задачи**

Конечный автомат – автомат, проверяющий допустимость слова на ленте, и возвращающий True / False (в данном случае Correct / Incorrect).

Конечный автомат может двигаться по ленте только в одном направлении.

Требуется написать функцию, реализующую конечный автомат.

На вход ей подается начальное состояние, конечные состояния, функция смены состояний и содержимое ленты.

Возвращаемое значение – ответ на вопрос, допустимо ли данное слово данным конечным автоматом.

Пример 1.

Таблица 1 – Таблица переходов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| char | a | b | c | c | - |
| cur | qb | qb | q1 | q2 | - |
| q1 | q1 | q2 | qe | qe | q0 |

qb – начальное состояние автомата;

qe – множество заключительных состояний;

a, b, c – входной алфавит, из которого формируются строки, считываемые автоматом;

cc – строка, считываемая автоматом.

Проверим допустимо ли слово на ленте для данного автомата.

Согласно таблице переходов получаем:

с qb → q0

с q0 → q0.

Так как q0 не соответствует множеству заключительных состояний, следовательно данное слово cc не допустимо.

Пример 2.

Таблица 2 – Таблица переходов

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| char | a | b | c | a | b | с |
| cur | qb | qb | qb | q1 | q2 | q3 |
| state | q1 | q2 | q3 | q1 | q2 | q3 |

q1 – начальное состояние автомата;

q1, q2, q3 – множество заключительных состояний;

a, b, c – входной алфавит, из которого формируются строки, считываемые автоматом;

aaaaaa – строка, считываемая автоматом.

Проверим допустимо ли слово на ленте для данного автомата.

Согласно таблице переходов получаем:

a q1 → q1

a q1 → q1

a q1 → q1

a q1 → q1

a q1 → q1

a q1 → q1

Так как q1 соответствует множеству заключительных состояний, следовательно данное слово aaaaaa допустимо для данного автомата.

**2. Математические и алгоритмические основы решения задачи**

**2.1 Понятие конечного автомата**

Конечный автомат – в теории алгоритмов математическая абстракция, позволяющая описывать пути изменения состояния объекта в зависимости от его текущего состояния и входных данных, при условии, что общее возможное количество состояний конечно. Конечный автомат является частным случаем абстрактного автомата.

Существуют различные варианты задания конечного автомата. Например, конечный автомат может быть задан с помощью пяти параметров:

где:



Q – конечное множество состояний автомата;

q0 – начальное состояние автомата ();



F – множество заключительных (или допускающих) состояний, таких что ;



Σ – допустимый входной алфавит (конечное множество допустимых входных символов), из которого формируются строки, считываемые автоматом;

δ – заданное отображение множества во множество подмножеств Q:



(иногда δ называют функцией переходов автомата).

Автомат начинает работу в состоянии q0, считывая по одному символу входной строки. Считанный символ переводит автомат в новое состояние из Q в соответствии с функцией переходов. Если по завершении считывания входного слова (цепочки символов) автомат оказывается в одном из допускающих состояний, то слово «принимается» автоматом. В этом случае говорят, что оно принадлежит языку данного автомата. В противном случае слово «отвергается».

Конечные автоматы широко используются на практике, например в синтаксических, лексических анализаторах, и тестировании программного обеспечения на основе моделей.

**2.2 Способы описания**

Диаграмма состояний (или иногда граф переходов) – графическое представление множества состояний и функции переходов. Представляет собой нагруженный однонаправленный граф, вершины которого – состояния конечного автомата, дуги – переходы из одного состояния в другое, а нагрузка – символы, при которых осуществляется данный переход. Если переход из состояния q1 в q2 может быть осуществлен при появлении одного из нескольких символов, то над дугой должны быть надписаны все они.

Таблица переходов – табличное представление функции δ. Обычно в такой таблице каждой строке соответствует одно состояние, а столбцу – один допустимый входной символ. В ячейке на пересечении строки и столбца записывается действие, которое должен выполнить автомат, если в ситуации, когда он находился в данном состоянии на входе он получил данный символ.

**2.3 Детерминированность**

Конечные автоматы подразделяются на детерминированные и недетерминированные.

http://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:%D0%94%D0%9A%D0%90.jpg

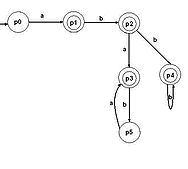


Рисунок 1 – Детерминированный http://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:%D0%94%D0%9A%D0%90.jpgконечный автомат

Детерминированным конечным автоматом (ДКА) называется такой автомат, в котором для каждой последовательности входных символов существует лишь одно состояние, в которое автомат может перейти из текущего.

Недетерминированный конечный автомат (НКА) является обобщением детерминированного. Недетерминированность автоматов достигается двумя способами.

1. Существуют переходы, помеченные пустой цепочкой ε (рисунок 2).

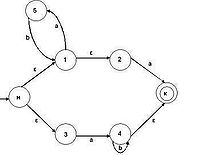


Рисунок 2 – Недетерминированный конечный автомат с пустыми переходами

2. Из одного состояния выходит несколько переходов, помеченных одним и тем же символом (рисунок 3).

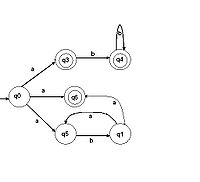


Рисунок 3 – Недетерминированный конечный автомат с несколькими переходами

Существует теорема, гласящая, что «Любой недетерминированный конечный автомат может быть преобразован в детерминированный так, чтобы их языки совпадали» (такие автоматы называются эквивалентными). Однако, поскольку количество состояний в эквивалентном детерминированном конечном автомате в худшем случае растёт экспоненциально с ростом количества состояний исходного недетерминированного конечного автомата, на практике подобная детерминизация не всегда возможна. Кроме того, конечные автоматы с выходом в общем случае не поддаются детерминизации.

В силу последних двух замечаний, несмотря на большую сложность недетерминированных конечных автоматов, для задач, связанных с обработкой текста, преимущественно применяются именно недетерминированные конечные автоматы.

**2.4 Автоматы и регулярные языки**

Для автомата можно определить язык (множество слов) в алфавите Σ, который он представляет – так называется множество слов, при вводе которых автомат переходит из начального состояния в одно из состояний множества F.

Теорема Клини гласит, что класс языков, представимых конечными автоматами, совпадает с классом регулярных языков. Кроме того, этот класс совпадает с классом языков, задаваемых регулярными грамматиками.

**3. Функциональные модели и блок-схемы решения задачи**

Функциональные модели и блок-схемы решения задачи представлены на рисунках 4 – 7.

Условные обозначения:

* cur – текущее слово;
* char – текущий символ;
* text – входное слово;
* funct – функция смены состояний;
* start – начальное состояние;
* end – список конечных состояний.



Рисунок 4 – Функциональная модель решения задачи для функции KA



Рисунок 5 – Функциональная модель решения задачи для функции function1



Рисунок 6 – Функциональная модель решения задачи для функции function2



Рисунок 7 – Функциональная модель решения задачи для функции isOneof

**4. Программная реализация решения задачи**

*; Тестовый конечный автомат – функция, преобразуюцая состояние*

*; Аргументы: 'cur' – текущее состояние*

*; 'char' – текущий символ*

*; Возвращаемое значение: новое состояние*

(**defun** **function1** (cur **char**)

(**cond**

((**and** (**eq** **char** `a) (**eq** cur `qb)) `q1)

((**and** (**eq** **char** `b) (**eq** cur `qb)) `q2)

((**and** (**eq** **char** `c) (**eq** cur `q1)) `qe)

((**and** (**eq** **char** `c) (**eq** cur `q2)) `qe)

(**t** `q0)

)

)

*; Тестовый конечный автомат – функция, преобразуюцая состояние*

*; Аргументы: 'cur' – текущее состояние*

*; 'char' – текущий символ*

*; Возвращаемое значение: новое состояние*

(**defun** **function2** (cur **char**)

(**cond**

((**and** (**eq** **char** `a) (**eq** cur `qb)) `q1)

((**and** (**eq** **char** `b) (**eq** cur `qb)) `q2)

((**and** (**eq** **char** `c) (**eq** cur `qb)) `q3)

((**and** (**eq** **char** `a) (**eq** cur `q1)) `q1)

((**and** (**eq** **char** `b) (**eq** cur `q2)) `q2)

((**and** (**eq** **char** `c) (**eq** cur `q3)) `q3)

(**T** **nil**)

)

)

*; Функция проверки, является ли 'char' элементом 'set' (необходима для остановки)*

*; Алгоритм проверки:*

*; 1. 'set' пусто => нет*

*; 2. 'char' совпадает с головой 'set' => да*

*; 3. 'char' является злементом хвоста 'set' => да*

*; 4. 'set' – не список => нет*

(**defun** **isOneOf** (**set** **char**)

(**cond**

((**eq** **set** **nil**) **nil**)

((**eq** **char** (**car** **set**)) **T**)

((isOneOf (**cdr** **set**) **char**) **T**)

(**T** **nil**)

)

)

*; Непосредственно конечный автомат*

*; Аргументы: 'begin' – начальное состояние*

*; 'end' – список конечных состояний*

*; 'move' – функция смены состояний*

*; 'text' – входное слово*

*; Возвращаемое значение: 'Correct' – входное слово допустимо*

*; 'Incorrect' – входное слово недопустимо*

*; Алгоритм:*

*; 1. Лента пуста и*

*; а. текущее состояние финальное => слово допустимо*

*; б. текущее состояние не финальное => слово недопустимо*

*; 2. Текущий символ допустим и лента не пуста => движемся дальше*

(**defun** **KA** (begin end move text)

(**cond**

((**eq** text **nil**)

(**cond**

((isOneOf end begin) `Correct)

(**T** `Incorrect)

)

)

(**T** (KA (**funcall** move begin (**car** text)) end move (**cdr** text)))

)

)

(**setq** input\_stream (**open** *«d:\\text.txt»*:direction:input))

*; входное слово*

(**setq** text (**read** input\_stream))

*; функция смены состояний*

(**setq** funct (**read** input\_stream))

*; начальное состояние*

(**setq** start (**read** input\_stream))

*; список конечных состояний*

(**setq** end (**read** input\_stream))

(**close** input\_stream)

(**setq** output\_stream (**open** *«d:\\KA.txt»*:direction:output))

(**print** (KA start end funct text) output\_stream)

(**terpri** output\_stream)

(**close** output\_stream)

**5. Пример выполнения программы**

Пример 1

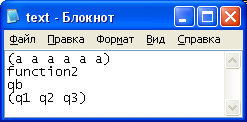


Рисунок 8 – Входные данные

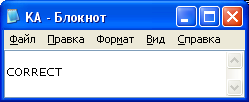


Рисунок 9 – Выходные данные

Пример 2

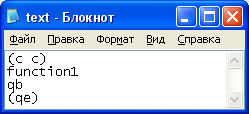


Рисунок 10 – Входные данные

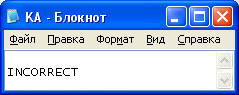


Рисунок 11 – Выходные данные

Пример 3.

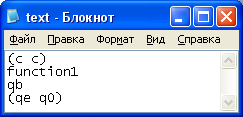


Рисунок 12 – Входные данные

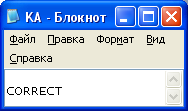


Рисунок 13 – Выходные данные

Заключение

Мышление в терминах конечных автоматов (то есть разбиение исполнения программы на шаги автомата и передача информации от шага к шагу через состояние) необходимо при построении событийно-ориентированных приложений. В этом случае программирование в стиле конечных автоматов оказывается единственной альтернативой порождению множества процессов или потоков управления.

Часто понятие состояний и машин состояний используется для спецификации программ. Так, при проектировании программного обеспечения с помощью UML для описания поведения объектов используются диаграммы состояний. Кроме того, явное выделение состояний используется в описании сетевых протоколов.

Итогом работы можно считать созданную функциональную модель реализации конечных автоматов. Созданная функциональная модель и ее программная реализация могут служить органической частью решения более сложных задач.

**Список использованных источников и литературы**

1. Бронштейн, И.Н. Справочник по математике для инженеров и учащихся втузов [Текст] / И.Н. Бронштейн, К.А. Семендяев. – М.: Наука, 2007. – 708 с.
2. Дехтярь, М.И. Введение в схемы, автоматы и алгоритмы. [Электронный ресурс] / М.И. Дехтярь. – М.: Наука, 2002. С. 642.
3. Конечный автомат [Электронный ресурс] – Режим доступа: http://ru/wikipedia.org/wiki/Конечный\_автомат.
4. Мозговой, М.В. Классика программирования: алгоритмы, языки, автоматы, компиляторы. Практический подход. / М.В. Мозговой. – М.: Наука и Техника, 2006. С. 320.
5. Семакин, И.Г. Основы программирования. [Текст] / И.Г. Семакин, А.П. Шестаков. – М.: Мир, 2006. C. 346.
6. Симанков, В.С. Основы функционального программирования [Текст] / В.С. Симанков, Т.Т. Зангиев, И.В. Зайцев. – Краснодар: КубГТУ, 2002. – 160 с.
7. Степанов, П.А. Функциональное программирование на языке Lisp. [Электронный ресурс] / П.А. Степанов, А.В. Бржезовский. – М.: ГУАП, 2003. С. 79.
8. Хювенен Э. Мир Лиспа [Текст] / Э. Хювенен, Й. Сеппянен. – М.: Мир, 1990. – 460 с.