СИНТЕЗ КОНЕЧНОГО АВТОМАТА ПО РЕГУЛЯРНЫМ ВЫРАЖЕНИЯМ

Дзнеладзе Г.Г, Картвелишвили О.М.
Грузинский Технический Университет

Резюме

Рассматривается абстрактный синтез конечного автомата, заданного на языке регулярных выражений. Указанная форма задания автомата является наиболее подходящей при разработке и реализации поведения в управляемых событиями программах, таких как сетевые драйверы и компиляторы. В компиляторах конечные автоматы используются для лексического анализа, при котором осуществляется распознавание и выделение лексем (последовательности допустимых символов языка программирования, имеющих смысл для компилятора) из входной последовательности символов. В данной работе предлагается построение конечного автомата, допускающего тот же язык, что и регулярное выражение. Используя построение Томпсона, для каждого регулярного выражения строится недетерминированный автомат как композиция из автоматов, соответствующих подвыражениям. На последующем этапе полученный недетерминированный автомат преобразуется в детерминированный. Минимизируется количество состояний с использованием поиска групп состояний, различимых входной строкой. Составлен алгоритм и программа на языке C++, которая была опробована на реальных примерах.

Ключевые слова: конечный автомат. Регулярные выражения. Абстрактный синтез.

1. Введение

Конечный автомат — 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 [1].

Конечные автоматы подразделяются на детерминированные и недетерминированные.

- 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 (не содержащей ни одного символа.);
2. Из одного состояния осуществляется несколько переходов, помеченных одним и тем же символом.

Как детерминированные, так и недетерминированные конечные автоматы способны к распознаванию регулярных множеств.

В данной работе рассматривается построение конечного автомата (как недетерминированного, так и детерминированного), допускающего тот же язык, что и заданное регулярное выражение.

Конечный автомат может быть задан с помощью пяти параметров: $M=(Q,T,\delta,q_0,F)$, где:

- **Q** — конечное *множество состояний* автомата;
- $\mathbf{q_0}$ — *начальное состояние* автомата ($q_0 \in Q$);
- **F** — множество *заключительных* (или *допускающих*) состояний, таких что $F \subseteq Q$;
- **T** — допустимый *входной алфавит* (конечное множество допустимых входных символов), из которого формируются строки, считываемые автоматом;
- **δ** — заданное отображение множества $Q \times T$ во множество $P(Q)$ подмножеств Q:

$$\delta : Q \times T \longrightarrow P(Q)$$

(иногда δ называют *функцией переходов автомата*, которая отображает пары входной символ-состояние на множество состояний).

Термин *алфавит*, или ***класс символов***, обозначает любое конечное множество символов. ***Строка*** над некоторым алфавитом — это конечная последовательность символов, взятых из алфавита. ***Пустая* строка**, обозначаемая как **e**, представляет собой специальную строку нулевой длины. **Термин *язык*** обозначает любое множество строк над некоторым фиксированным алфавитом [1]. Автомат начинает работу в состоянии $\mathbf{q_0}$, 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».

Для автомата можно определить множество слов в алфавите T, при вводе которых автомат переходит из начального состояния в одно из состояний множества F. Регулярное выражение в алфавите T и обозначаемое им регулярное множество в алфавите T определяются рекурсивно следующим образом:

1. $\varnothing$ регулярное выражение, обозначающее регулярное множество $\varnothing$;
2. {e} - регулярное выражение, обозначающее регулярное множество {e};
3. {a} - регулярное выражение, обозначающее регулярное множество {a};
4. если r_i и r_j –регулярные выражения, обозначающие регулярные множества A и B соответственно, то
   - $(r_i|r_j)$ - регулярное выражение, обозначающее регулярное множество AUB (объединение),
   - (r_ir_j)- регулярное выражение, обозначающее регулярное множество A·B (конкатенация),
   - (r^*) - регулярное выражение, обозначающее регулярное множество A^*(итерация);
5. ничто другое не является регулярным выражением в алфавите **T** [2].

Недетерминированный конечный автомат (НКА) - по определению есть пятерка M=(Q, T, D, q_0, F), где

D - функция переходов (отображающая множество $Q \times (T \cup \{e\})$ в множества Q), определяющая поведение конечного автомата.

Частным случаем недетерминированного конечного автомата является детерминированный конечный автомат, который на каждом такте работы имеет возможность перейти не более чем в одно состояние и не может делать переходы по **e**.

Будем называть **M'** детерминированным конечным автоматом (**ДКА**), если выполнены следующие два условия:

1. D(q, e) = $\varnothing$, для любого $q \in Q$, и
2. D(q, a) содержит не более одного элемента для любых $q \in Q$ и $a \in T$.

Конечный автомат может быть изображен графически в виде диаграммы, представляющей собой ориентированный граф, в котором каждому состоянию соответствует вершина, а дуга, помеченная символом $a \in T \cup \{e\}$, соединяет две вершины q_i и q_j. На диаграмме выделяются начальное и заключительные состояния (соответственно, входящей стрелкой и двойным контуром).

Рассмотрим в качестве примера недетерминированный конечный автомат ***M***, допускающий язык ***L***:

M = { {1, 2, 3, 4, 5}, {a, b}, D, 1, {4} },

где функция переходов **D** определяется как:

D (1, a) = {2} ,
D (2, b) = {3} ,
D (3, a) = {3, 4}

Диаграмма автомата приведена на рис. 1.

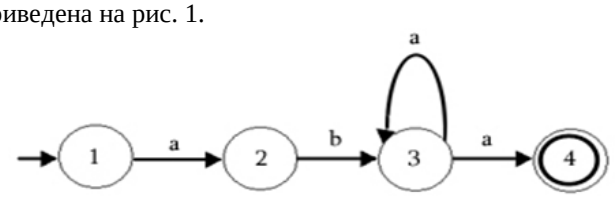


Рис. 1

Функция переходов НКА может быть реализована также *таблицей переходов*, в которой строки представляют состояния, а столбцы — входные символы (и, при необходимости, ***e***). Запись в строке ***i*** для символа ***a*** является множеством состояний, которые могут быть достигнуты переходом из состояния ***i*** при входном символе ***a***. Таблица переходов для НКА показана ниже.

Состояние	**Входной символ**	
	a	***b***
1	{2}	—
2	—	{3}
3	{3,4}	—

НКА *допускает*, или *принимает* входную строку ***s*** (а эта строка является *допустимой)* тогда и только тогда, когда в графе переходов существует некоторый путь от начального состояния к какому-либо из заключительных, такой, что метки дуг этого пути соответствуют строке **s**. Приведенный выше НКА допускает входные строки ***aba, abaa, abaaa***, **…** Например, ***abaa*** допускается по пути из состояния ***1*** вдоль дуги ***a*** в состояние ***2***, затем в состояния ***3*** вдоль дуги ***b***, затем в состояния ***3*** вдоль дуги ***a*** и затем в заключительное состояние ***4*** вдоль дуги ***a***.

Путь может быть представлен в виде последовательности переходов состояний, так называемых *перемещений*, или *ходов*. Следующая диаграмма показывает перемещения, выполненные для входной строки ***abaa****:*

1 → 2 → 3 → 3 → 4

Вообще говоря, в заключительное состояние может приводить более чем одна последовательность перемещений. Необходимо отметить, что для входной строки могут быть осуществлены и некоторые другие последовательности перемещений, не приводящие в заключительное состояние.

2. Основная часть

Целью настоящей работы является построение конечного автомата допускающего тот же язык что и заданное регулярное выражение. Рассмотрим алгоритм построения недетерминированного конечного автомата по регулярному выражению.

Входные данные: регулярное выражение ***r*** в алфавите ***T***.

Результат: НКА ***M***, такой что ***L(M) = L(r).***

Используем построение Томпсона.Автомат для выражения строится композицией автоматов, соответствующих подвыражениям. На каждом шаге построения строящийся автомат имеет в точности одно заключительное состояние, в начальное состояние нет переходов из других состояний и нет переходов из заключительного состояния в другие [2].

1. Автомат, для выражения ***e*** имеет вид:

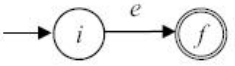


Рис. 2

2. Автомат, для выражения $a(a \in T)$ имеет вид:

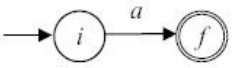


Рис. 3

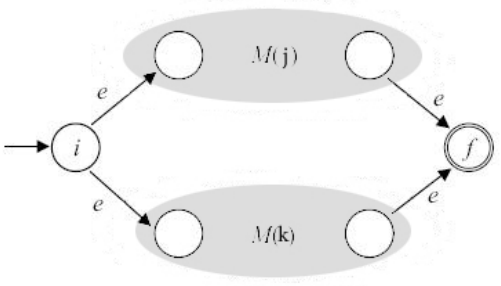


Рис. 4

3. Пусть **M(*j*)** и **M(*k*)** - НКА для регулярных выражений ***j*** и ***k*** соответственно.

- Для выражения ***j | k*** автомат ***M(j | k)*** строится в соответствии с показанным на рис.4. Здесь ***i*** - новое начальное состояние и ***f*** - новое заключительное состояние. Заметим, что имеет место переход по ***e*** из ***i*** в начальные состояния ***M(j)*** и ***M(k)*** и переход по ***e*** из заключительных состояний ***M(j)*** и ***M(k)*** в ***f***. Начальное и заключительное состояния автоматов ***M(j)*** и ***M(k)*** не являются таковыми для автомата ***M(j | k)***.

- Для выражения ***jk*** автомат ***M(jk)*** строится следующим образом:

Рис. 5

Начальное состояние автомата ***M(j)*** становится начальным для нового автомата, а заключительное состояние ***M(k)*** становится заключительным для нового автомата. Добавляется переход по ***e*** из заключительнго состояние ***M(j)*** в начальноое состояния ***M(k)***.

- Для выражения j^* автомат $M(j^*)$ строится как показано на рис.6. Здесь ***i*** - новое начальное состояние, а ***f*** - новое заключительное состояние.

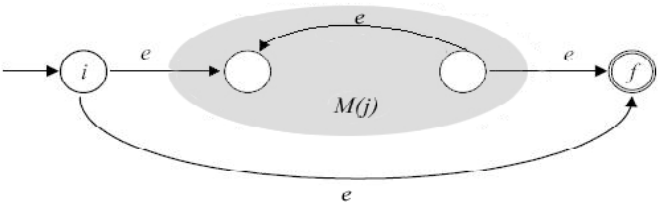


Рис. 6

Таким образом, построение НКА сводится к следующему:

1) Вначале мы разбираем ***r*** на составляющие подвыражения (составляем дерево разбора). Затем, используя приведенные выше правила (1) и (2), строим НКА для каждого базового символа в ***r*** (которые представляют собой символы входного алфавита, либо ***e***). Необходимо отметить, что если в ***r*** несколько раз встречается один и тот же символ, для каждого экземпляра этого символа создается отдельный НКА с собственными состояниями, при этом каждое новое состояние получает индивидуальное имя. Никакие состояния любого из компонентов НКА не могут иметь одно и то же имя;

2) Затем, следуя синтаксической структуре регулярного выражения ***r*** , комбинируем эти НКА с использованием правила (3), пока не будет получен НКА для всего выражения. Каждый промежуточный НКА соответствует своему подвыражению ***r*** и имеет ровно одно заключительное состояние. При этом ни одна дуга не входит в начальное состояние и ни одна не выходит из заключительного.

Рассмотрим алгоритм для преобразования НКА в ДКА, распознающий тот же язык, что и НКА. Этот алгоритм, часто называют *построением подмножества* [3] .

В таблице переходов НКА каждая запись представляет собой множество состояний; в таблице переходов ДКА — единственное состояние. Общая идея преобразования НКА в ДКА состоит в том, что каждое состояние ДКА соответствует множеству состояний НКА [4].

ДКА использует свои состояния для отслеживания всех возможных состояний, в которых НКА может находиться после чтения очередного входного символа. Таким образом, после чтения входного потока $a_1a_2...a_n$ ДКА находится в состоянии, которое представляет подмножество Q состояний НКА, достижимых из стартового состояния НКА по пути, помеченному как $a_1a_2...a_n$.

Входные данные: НКА ***M***.

Результат: ДКА $\mathbf{M}'$, допускающий тот же язык.

Целью работы данного алгоритма является построение таблицы переходов **Dtab** для $\mathbf{M}'$. Каждое состояние ДКА является множеством состояний НКА, и мы строим **Dtab** так, чтобы $\mathbf{M}'$ "параллельно" моделировал все возможные перемещения M по данной входной строке.

Для отслеживания множеств состояний НКА используются операции [5]:

1. **e-closure(q)** — результатом операции является множество состояний НКА, достижимых из состояния ***q*** только по **e**-переходам

2. **e-closure(R)** — результатом операции является множество состояний НКА, достижимых из какого-либо состояния q из R только по e-переходам

3. **move(R, a)** — результатом операции является множество состояний НКА, в которые имеется переход из какого- либо состояния ***q*** из ***R*** по входному символу ***a,*** *где* q— состояние НКА, а R— множество состояний НКА.

До считывания первого входного символа ***M*** может быть в любом из состояний множества ***eclosure(*** q_0 ***),*** где q_0 — начальное состояние ***M.***

Допустим, что состояния множества ***R,*** и только они, достижимы из q_0 после прочтения данной последовательности входных символов, и пусть ***a*** — следующий входной символ. Получив ***a, M*** может переместиться в любое состояние из множества ***move(R, a).*** Совершив из этих состояний все возможные e-переходы, ***M*** после обработки ***a*** может быть в любом состоянии из ***e-closure(move(R, a)).***

Множество состояний ***DS*** автомата M' и таблицу его переходов ***Dtab*** можно создать, исходя из нижеследующего. Каждое состояние $\boldsymbol{M}'$ соответствует множеству состояний НКА, в которых может находиться ***M*** после чтения некоторой последовательности входных символов, включая все возможные e-переходы до и после считанных символов.

Стартовое состояние $\boldsymbol{M}'$ — ***e-closure(*** q_0 ***).*** Состояние $\boldsymbol{M}'$ является допускающим, если оно представляет собой множество состояний НКА, содержащих хотя бы одно допускающее состояние ***M.***

Добавление состояний и переходов в $\mathbf{M}'$ происходит по следующему алгоритму:

```
DS = пустое множество.
Заносим в DS состояние e-closure( q0 ) которое непомечено;
пока в DS имеется непомеченное состояние R
{
пометить R;
для каждого входного символа a
{
   Temp = e-closure(move(R, a));
       если Temp ещё не занесён в DS
        {
            добавить Temp как непомеченное состояние в DS;
        }
```

Dtab(R, a) = Temp;
}
}
Конец

Каждое регулярное множество распознается единственным для данного множества ДКА с минимальным числом состояний. Рассмотрим построение такого ДКА уменьшением числа состояний данного ДКА до минимального без влияния на распознаваемый язык. Допустим: есть ДКА $\mathbf{M}'$ с множеством состояний Q и алфавитом входных символов **T**. Пусть данный ДКА всюду определен (т. е. каждое состояние имеет переход для каждого входного символа). Если исходный ДКА не является всюду определеннным, его можно преобразовать к всюду определённому ДКА следующим способом:

1) добавить новое мертвое состояние $\mathbf{q_k}$, с переходами из $\mathbf{q_k}$ в $\mathbf{q_k}$ для всех входных символов.

2) для всех состояний $\mathbf{M}'$ выполнить следующее: если у состояния $\mathbf{q_i}$ нет перехода по какому-то входному символу **a,** добавить состоянию q_i переход по символу a в состояние $\mathbf{q_k}$. Таким образом на входе алгоритм получает всюду опрределенный ДКА M'.

Введём понятие различимости состояний.

Строка **s** различает состояния q_i и q_j, если, ДКА M начиная работу в состоянии q_i и получая на вход строку **s**, заканчивает ее в заключительном состоянии, а начиная с q_j — в незаключительном состоянии (и наоборот).

Алгоритм минимизации количества состояний ДКА работает путем поиска всех групп состояний, различимых посредством некоторой входной строки [5]. Каждая группа состояний, которые не отличаются друг от друга, сливается в одно состояние. В алгоритме используется метод работы с разбиениями множеств состояний.

Каждая группа состояний в разбиении содержит состояния, которые еще не были разделены друг от друга, и в процессе работы выбираются пары состояний из разных групп, различаемые некоторой входной строкой.

Изначально разбиение состоит из двух групп: заключительные состояния и незаключительные. Основной шаг состоит в том, чтобы взять некоторую группу состояний, скажем $P = \{ p_1, p_2, \ldots p_k \}$, и некоторый входной символ *a* и посмотреть, какие переходы имеют состояния $p_1, p_2, \ldots p_k$ для этого входного символа. Если эти переходы представляют собой переходы в состояния, попадающие в две или более различные группы текущего разбиения, нужно разбить группу *P* на подгруппы так, чтобы для каждого из подмножеств *P* эти переходы ограничивались одной группой текущего разбиения. Допустим, что p_i и p_j переходят при входном символе *a* в состояния v_i и v_j, находящиеся в разных группах разбиения. Тогда необходимо разбить *P*, как минимум, на такие два подмножества, чтобы одно из них содержало p_i, а второе — p_j.

Процесс разбиения повторяется до тех пор, пока не останется группа, которая не подлежит разбиению...

На рис.7 представлены автоматы, построенные программой, разработанной в данной работе, по регулярному выражению **(a²b²c)*s(d²g²f)*t²m.**

3. Заключение

Рассматривается абстрактный синтез конечного автомата по регулярным выражениям, представляющим последовательность допустимых символов языка программирования, которые имеют смысл для компилятора. В процессе лексического анализа производится распознавание и выделение лексем из входной последовательности символов.

Разработанный автомат состоит из нескольких блоков.

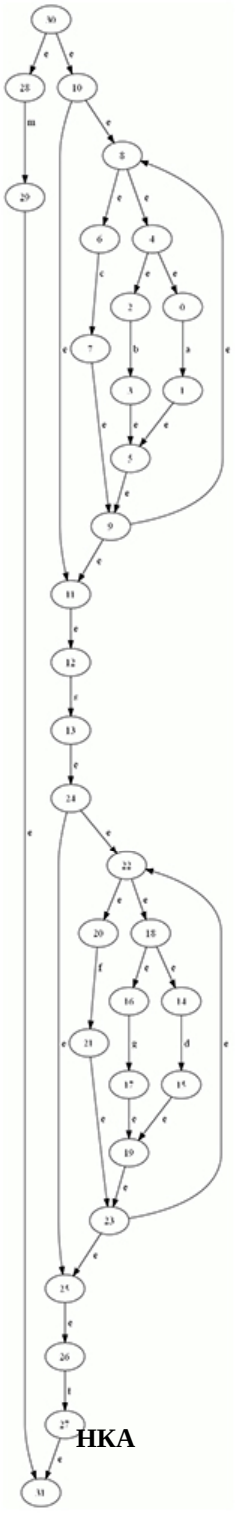
В первом из них строится недетерминированный автомат, учитывающий все его переходы; для этого используется построение Томпсона, при котором автомат строится, как композиция автоматов, соответствующих отдельным частям регулярного выражения.

Далее недетерминированный автомат преобразуется в детерминированный.

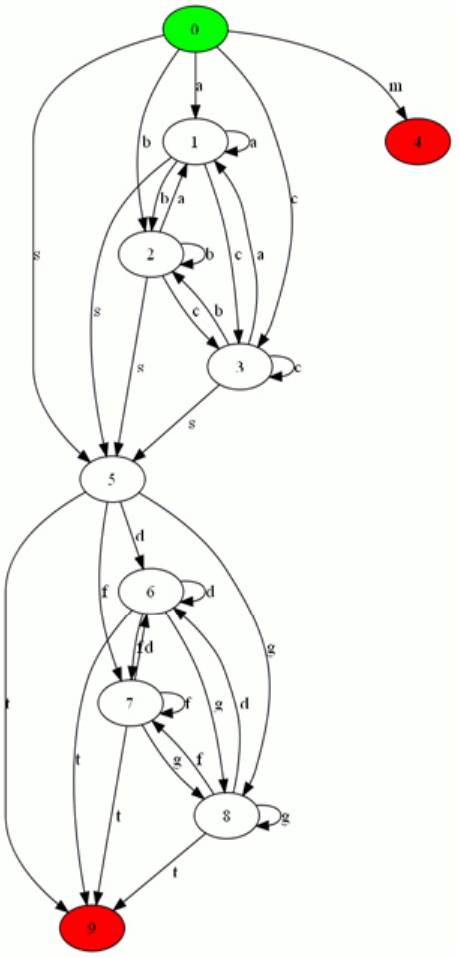
В заключительной части осуществляется минимизация состояний автомата.

Автомат , построенный по разработанному алгоритму позволяет:

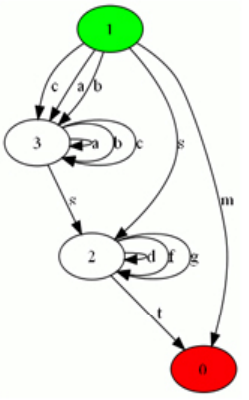
- проверить строку на соответствии шаблону;
- идентифицировать определенный текст в документе с целью его замены или удаления;
- извлекать из строки подстроку в соответствии с шаблоном.



НКА



ДКА



минимизирован ный ДКА

Рис.7

Литература:

1. Пентус А.Е.,Пентус М.Р. Математическая теория формальных языков. http//www.intuit.ru/department/algoritms/mathformlang
2. Серебряков., Галочкин., Гончар Д.Р., Фуругян. Теория и реализация языков программирования. http//www.intuit.ru/department
3. Aho Alfred V.,Seti Ravi., Uiiman Jeffrey D. Compillers: Principles,Techniques, and Tools.2007/sa/pltheory
4. Хопкрофт Д., Мотвини Раджив,Ульман Д. Введение в теорию автоматов, языков и вычислений. Москва ,из-во «Вильямс»,2002
5. Вавилов Е.Н., Портной Г.П.. Синтез схем электронных цифровых машин.Москва, «Сов.Радио» 1963

SYNTESIS OF FINITE AUTOMATA ACCORDING TO REGULAR EXPRESIONS

Dzneladze G. Kartvelishvili O.
Georgian Technical University

Summary

The present document is covering abstract synthesis of finite automata which is described using regular expression language. This type of automata presentation form is the most acceptable for the development and implementation of behavior in event-driven programs like network drivers and compilers. In personal computers finite automats are used for lexical analysis, which performs detection and selection of lexems (sequence of allowed characters of programming language, which have special meaning for the computer) from the input character sequence. This document proposes the finite automata structure, which accepts the same language as regular expressions. For each regular expression a non-deteministic automata is built using Thompson algorithm as a composition of automats corresponding with sub-expressions. During the subsequent stage the obtained non-deterministic automata is transformed into a deterministic one. The number of states is minimized using the search of state groups distinguishable by input string. In addition, corresponding algorithm and program were created and tested on real examples.

სასრულო ავტომატის სინთეზი რეგულარული გამოსახულების საფუძველზე

გ. ძნელაძე, ო. ქართველიშვილი
საქართველოს ტექნიკური უნივერსიტეტი

რეზიუმე

განხილულია რეგულარული გამოსახულების ენაზე მოცემული სასრულო ავტომატის აბსტრაქტული სინთეზი. ავტომატის აღნიშნული ფორმით მოცემა წარმოადგენს ყველაზე მიზანშეწონილს ისეთი პროგრამების დამუშავებასა და ქცევის რეალიზაციისათვის, რომლებიც იმართება ხდომილებით, როგორიცაა ქსელური დრაივერები და კომპილიატორები. კომპილიატორებში სასრულო ავტომატები გამოიყენება ლექსიკური ანალიზისათვის, რომლის დროსაც სრულდება ლექსემის (პროგრამირების ენის დასაშვები სიმბოლოების თანამიმდევრობა, რომელსაც აზრი აქვს კომპილიატორისათვის) გამოცნობა და გამოყოფა შემავალი სიმბოლოების მიმდევრობაში. მოცემულ ნაშრომში შემოთავაზებულია სასრულო ავტომატის აგების ალგორითმი, რომლისთვისაც დასაშვებია რეგულარული გამოსახულების ენა. ტომსონის აგების გამოყენებით ყოველი რეგულარული გამოსახულებისათვის აიგება არადეტერმინირებული ავტომატი, როგორც ცალკეული ქვეგამოსახულების შესაბამისი ავტომატების კომპოზიცია. შემდგომ ეტაპზე არადეტერმინირებული ავტომატი გადაისახება დეტერმინირებულად. სრულდება მდგომარეობათა რაოდენობის მინიმიზაცია, რომლისთვისაც გამოიყენება იმ მდგომარეობათა ჯგუფების მოძებნა, რომლებიც განსხვავებულია შემავალი სტრიქონებით. დამუშავებულია ალგორითმი და პროგრამა C++ ენაზე, რომელიც შემოწმებული იქნა რეალურ მაგალითებზე.