NFA to Minimized DFA Project Report

Michael

1030300036

November 16, 2012

HKBU-BNU United International College

COMP3100: Principle of Programming Lanuages

For: Professor Weifeng Su

Abstract

This paper is about the research of converting Non-deterministic Finite Automata (NFA) to minimized Deterministic Finite Automata (DFA). The algorithm of converting NFA to DFA and converting DFA to minimized DFA will be covered in this project. The python code will also be used to implement this algorithm in almost all kinds of platform of electronic devices.

1.Introduction

In this project a NFA statement should be converted to minimized DFA automatically. The python code should get the input of NFA and output a minimized DFA statement.

2. Background

2.1 Finite automata

Finite automata are recognizers, they simply say "yes" or "no" about each possible input string. Finite automata come in two flavors: Deterministic finite automaton (DFA) and Nondeterministic finite

automaton (NFA).

2.2 Nondeterministic finite automaton (DFA)

Nondeterministic finite automata (NFA) have no restrictions on the labels of their edges. A symbol can label several edges out of the same state, and E, the empty string, is a possible label.

A nondeterministic finite automaton (NFA) consists of:

1. A finite set of states S.
2. A set of input symbols Σε, the input alphabet. We assume that ε, which stands for the empty string, is never a member of Σ.
3. A transition function that gives, for each state, and for each symbol inΣ U {ε} a set of next states.
4. A state 80 from S that is distinguished as the start state (or initial state) .
5. A set of states F, a subset of S, that is distinguished as the accepting states (or final states).

We can represent either an NFA or DFA by a transition graph, where the nodes are states and the labeled edges represent the transition function. There is an edge labeled a from state s to state t if and only if t is one of the next states for state 8 and input a. This graph is very much like a transition diagram, except:

a) The same symbol can label edges from one state to several different

states, and

b) An edge may be labeled byε, the empty string, instead of, or in addition to, symbols from the input alphabet.

2.3 Deterministic finite automaton (DFA)

Deterministic finite automata (DFA) have, for each state, and for each symbol of its input alphabet exactly one edge with that symbol leaving that state.

A deterministic finite automaton (DFA) is a special case of an NFA where:

1. There are no moves on inputε, and

2. For each state S and input symbol a, there is exactly one edge out of s labeled a.

2.4 Minimizing the Number of States of a DFA

There can be many DFA's that recognize the same language. For instance, note that the DFA's of Figs. Not only do these automata have states with different names, but they don't even have the same number of states. If we implement a lexical analyzer as a DFA, we would generally prefer a DFA with as few states as possible, since each state requires entries in the table that describes the lexical analyzer. It turns out that there is always a unique (up to state names) minimum state DFA for any

regular language. Moreover, this minimum-state DFA can be constructed from any DFA for the same language by grouping sets of equivalent states.

2.5 Python

Python is a general-purpose, high-level programming language whose design philosophy emphasizes code readability. Python's syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C, and the language provides constructs intended to enable clear programs on both a small and large scale.
Python supports multiple programming paradigms,
including object-oriented, imperative and functional programmingstyles.
It features a fully dynamic type system and automatic memory management, similar to that of Scheme, Ruby, Perl and Tclm and has a large and comprehensive standard library.
Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts.
Using third-party tools, Python code can be packaged into standalone executable programs. Python interpreters are available for many operating systems.

3. Methodology

This project will be processed under python 2.7.3 Mountain Lion operating system on MacBook Pro 2012 late version. The data will be input in IDLE, which is a shell of python programming. And the output will also be shown in IDLE.

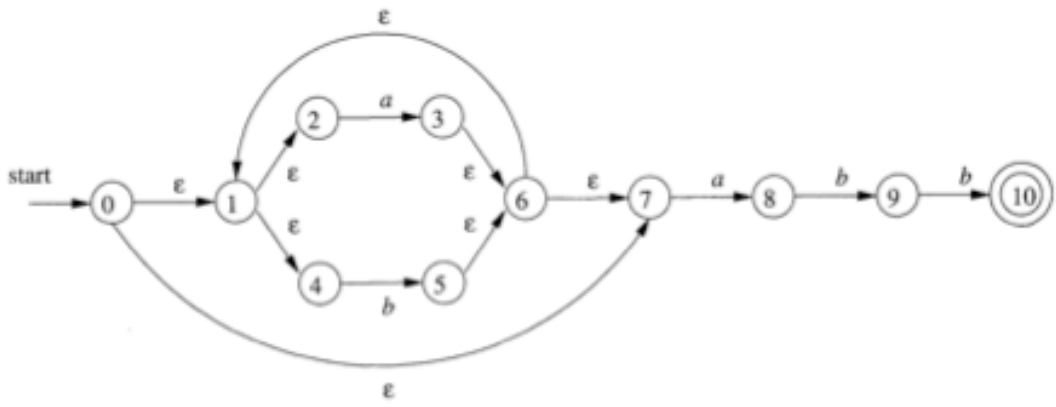
3.1 NFA to DFA

The general idea behind the subset construction is that each state of the constructed DFA corresponds to a set of NFA states. After reading input al a2 . . . an , the DFA is in that state which corresponds to the set of states that the NFA can reach, from its start state, following paths labeled al a2 . . . an.

It is possible that the number of DFA states is exponential in the number of NFA states, which could lead to difficulties when we try to implement this DFA. However, part of the power of the automaton-based approach to lexical analysis is that for real languages, the NFA and DFA have approximately the same number of states, and the exponential behavior is not seen.

OPERATION	DESCRIPTION
ϵ-*closure*(s)	Set of NFA states reachable from NFA state s on ϵ-transitions alone.
ϵ-*closure*(T)	Set of NFA states reachable from some NFA state s in set T on ϵ-transitions alone; $= \cup_{s \text{ in } T}$ ϵ-*closure*(s).
move(T, a)	Set of NFA states to which there is a transition on input symbol a from some state s in T.

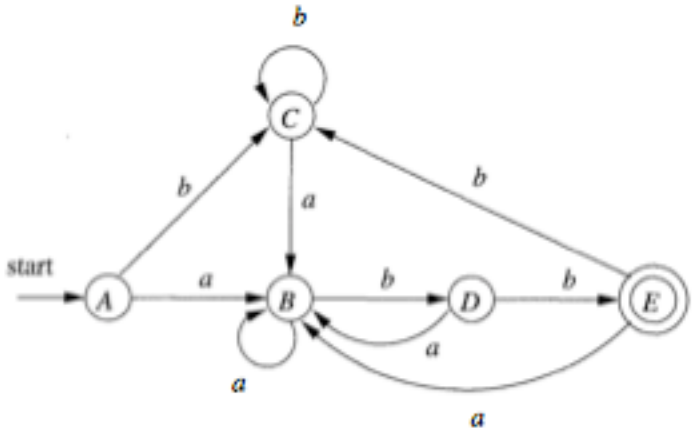
For a NFA function as following:



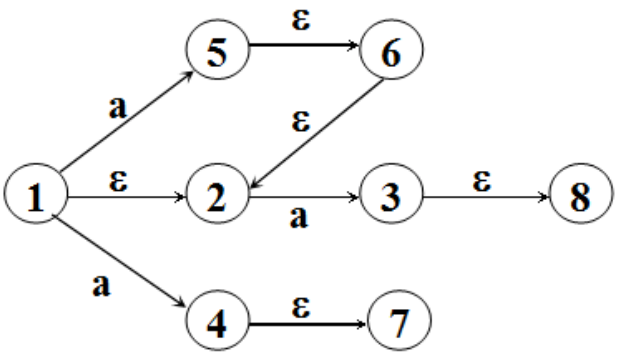
It should be transfer to a transition table first according to computation of ε-closure (S):

NFA STATE	DFA STATE	a	b
{0, 1, 2, 4, 7}	A	B	C
{1, 2, 3, 4, 6, 7, 8}	B	B	D
{1, 2, 4, 5, 6, 7}	C	B	C
{1, 2, 4, 5, 6, 7, 9}	D	B	E
{1, 2, 3, 5, 6, 7, 10}	E	B	C

the state will then be drawn in a new DFA diagram:

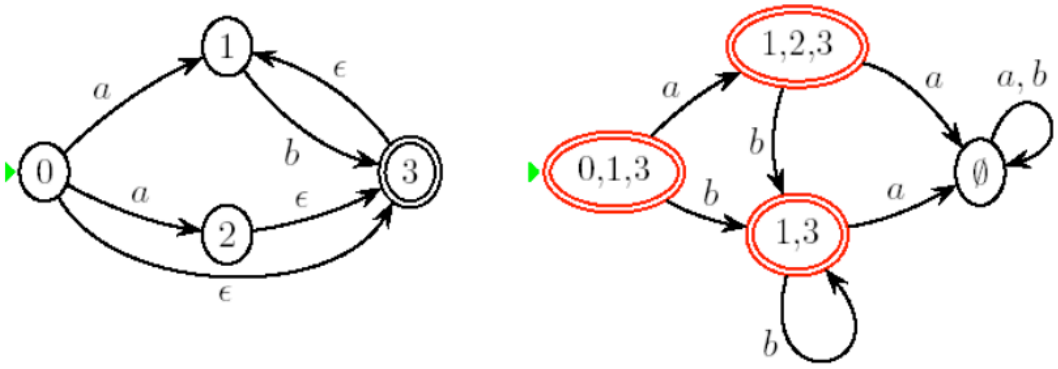


For set of states S, ε-closure(S) is the set of states that can be reached from S without consuming any input.



ε-closure ({1})={1，2}=I　　Ia=ε-closure ({5,4,3})

Example:



3.2 Minimize DFA

It turns out that there is always a unique (up to state names) minimum state DFA for any regular language. Moreover, this minimum-state DFA can be constructed from any DFA for the same language by grouping sets of equivalent states.

To minimize a DFA, we should firstly separate states into final states and nonfinal states. Then check for different sigma, will those nonfinal states reach the final states, and divide them according to their behavior.

4. Data

Data form used in this program is set.

4.1 Input Data:

```
NFA = [[[['q0'],'0',['q1']],
        [['q0'],'3',['q5']],
        [['q1'],'1',['q4']],
        [['q1'],'3',['q5']],
        [['q2'],'3',['q0']],
        [['q2'],'0',['q4']],
        [['q3'],'0',['q5']],
        [['q3'],'0',['q2']],
        [['q4'],'3',['q5']]],['q0'],['q5'],['1','2','3'],['q0','q1','q2','q3','q4','q5']]
```

The input data should be input to IDLE by keyboard, its form is a big set. The output data is also a set, the first part is transition function, the second part is start state, the third part is final state, the forth part is sigma, and the fifth part is all states.

4.2 Output Data:

```
MDFA = [[[['q0','q3'],'0',['q1','q2','q4']],
         [['q0','q3'],'1',['q0','q1','q2','q5']],
         [['q1','q2','q3','q4'],'0',['q0','q1','q4','q5']],
         [['q1','q2','q3','q4'],'1',['q1','q5']]],
        ['q0','q3'],
        [['q0','q1','q2','q5'],['q0','q1','q4','q5'],['q1','q5']],
        ['1','2','3'],
        [['q0','q1','q2','q5'],['q0','q1','q4','q5'],['q1','q5'],['q1','q2','q3','q4'],['q0','q3']
```

The output data is also a set, and its form is the same to the input data. The first part is transition function, the second part is start state, the third part is final state, the forth part is sigma, the fifth part is all states.

5. Analysis

After the minimization of DFA, the number of transition function

reduced and the structure of the finite automata seems much more clear. It reduced complexity and redundancy of NFA.

6. Conclusions

In this project we learned what is Deterministic finite automaton (DFA) and Nondeterministic finite automaton (NFA). We get a simple view of how programming language and finite automata designed. We also learned how to programming in python, especially the grammar.

Principle of programming language is really a important course.

7. References

Compilers: Principles, Techniques, and Tools, 2nd ed,

by Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman

Addison-Wesley, September 2006.

http://en.wikipedia.org/wiki/Deterministic_finite_automaton

http://en.wikipedia.org/wiki/Nondeterministic_finite_automaton