CPSC-402 Report
Compiler Construction

Sebastian Brumm
Chapman University

April 18, 2022

Abstract

Contents

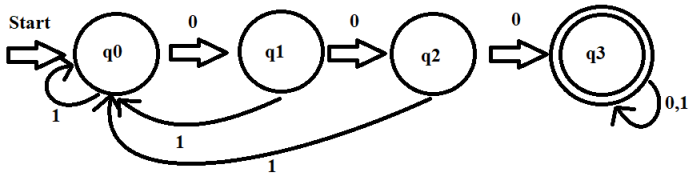
1 Introduction

1.0.1 Definitons, Examples, Theorems, Etc

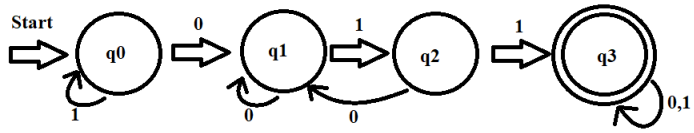
2 Homework

2.1 Week 1

Exercise 2.2.4:
a. The set of all strings with three consecutive 0's



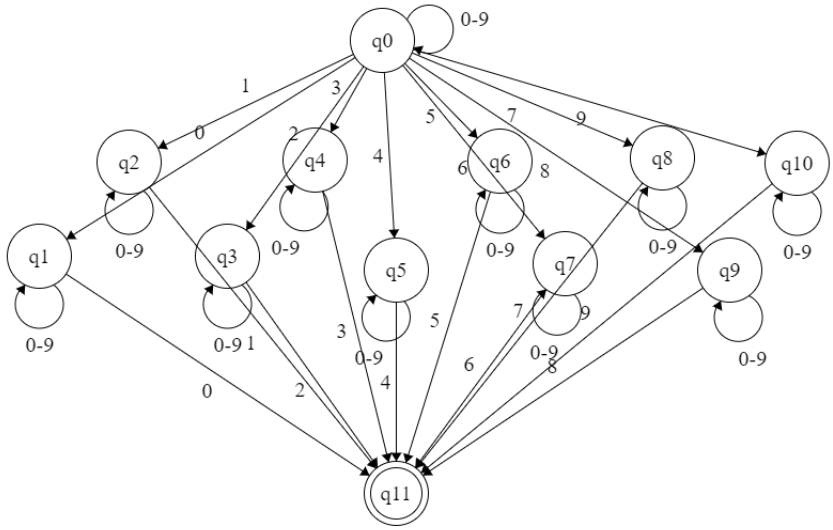
b. The set of all strings with 001 as a substring



2.2 Week 2

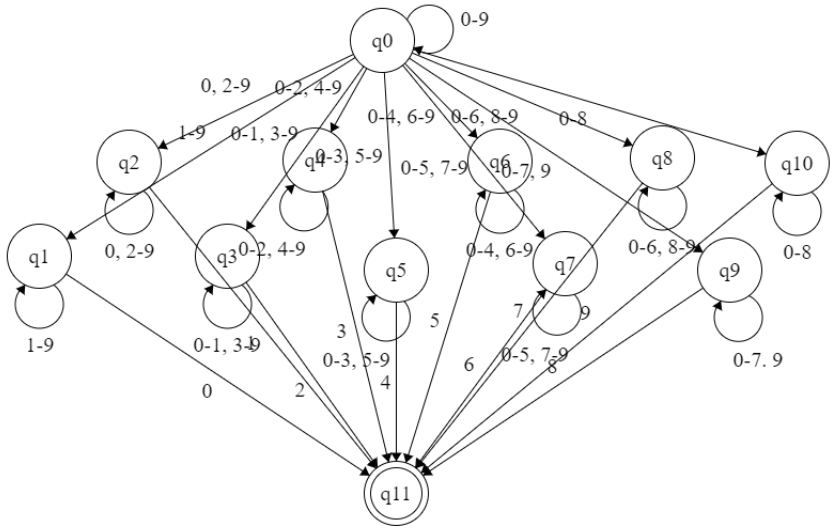
2.2.1 2.3.4-a: The set of all strings 0-9 such that the final digit has appeared before

Regular Expression: $\Sigma^*0\Sigma^*0 + ...$ add sigma def



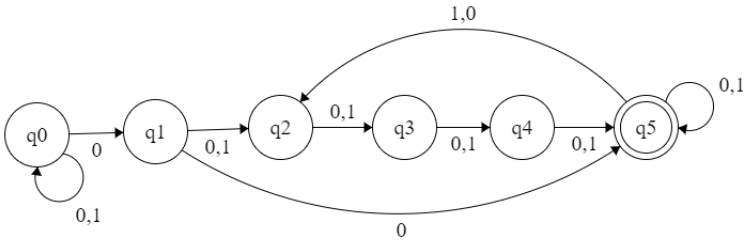
2.2.2 2.3.4-b: The set of all strings 0-9 such that the final digit has not appeared before

Regular Expression: $\{1-9\}0+\{0,2-9\}1...$



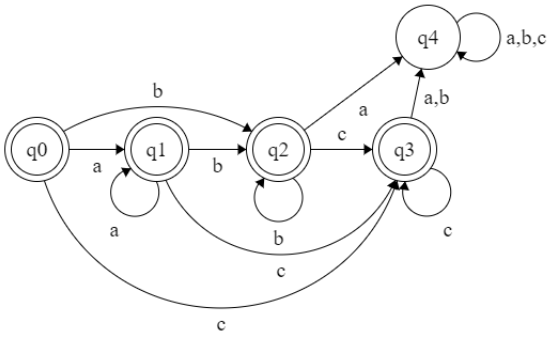
2.2.3 2.3.4-c: The set of all strings 0,1 such that there are two 0's separated by a number of positions that is a multiple of 4, including 0

Regular Expression: $\Sigma^*0\Sigma^{4*}0\Sigma^*$:wrong automata, add another state after q5 with a zero transition and make it accepting



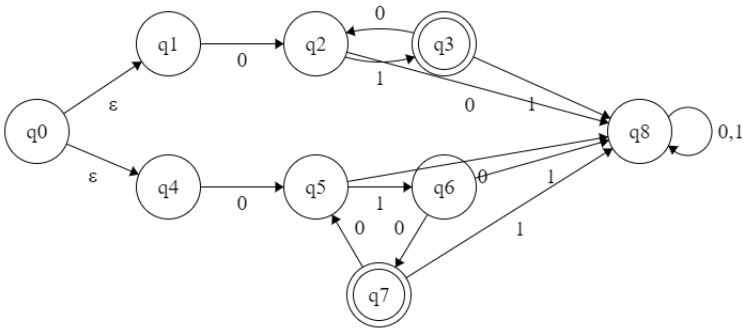
2.2.4 2.5.3-a: The set of all strings a,b,c consisting of zero or more a's followed by zero or more b's followed by zero or more c's

Regular Expression: $a^*b^*c^*$



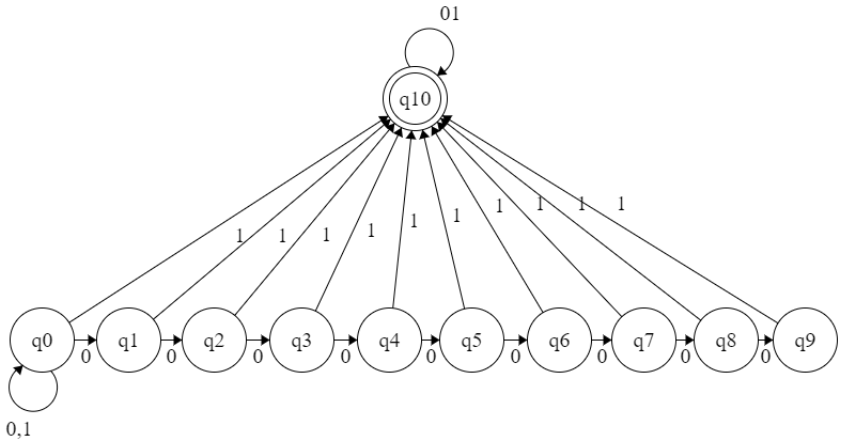
2.2.5 2.5.3-b: The set of all strings consisting of either 01 repeated one or more times or 010 repeated one or more times

Regular Expression: $(01)^+ + (010)^+$



2.2.6 2.5.3-c: The set of all strings 0,1 where at least one of the last ten positions is a 1

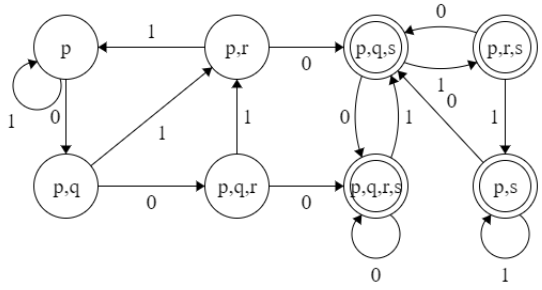
Regular Expression: $\Sigma^*1\Sigma^9 + \Sigma^*1\Sigma^8 ... + \Sigma^*1$: remove loop on the accept state



2.3 Week 3

Exercise 2.3.1 : Convert to a DFA the following NFA:

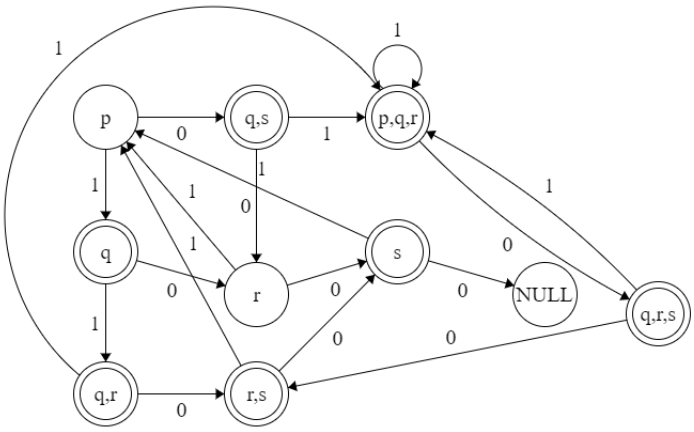
	0	1
$\rightarrow p$	$\{p, q\}$	$\{p\}$
q	$\{r\}$	$\{r\}$
r	$\{s\}$	$\emptyset$
$*s$	$\{s\}$	$\{s\}$



: change it so that pqrs -¿ prs on 1

Exercise 2.3.2 : Convert to a DFA the following NFA:

	0	1
$\rightarrow p$	$\{q, s\}$	$\{q\}$
$*q$	$\{r\}$	$\{q, r\}$
r	$\{s\}$	$\{p\}$
$*s$	$\emptyset$	$\{p\}$

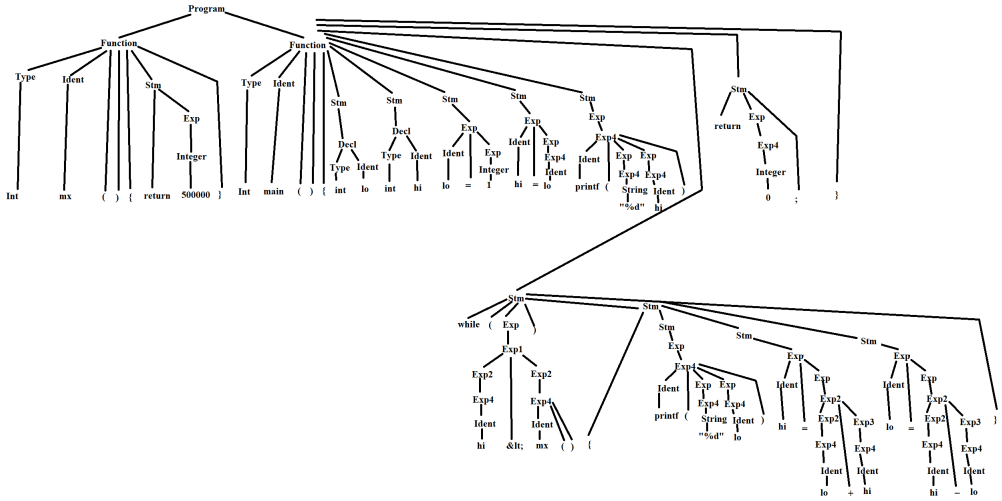



```
-- convert an NFA to a DFA
nfa2dfa :: NFA s -> DFA [s]
nfa2dfa nfa = DFA {
  -- exercise: correct the next three definitions
  dfa_initial = [(nfa_initial nfa)],
  dfa_final = let
    final qs = disjunction (map (nfa_final nfa) qs)
    in final,
  dfa_delta = let
    delta [] c = []
    delta (q:qs) c = concat [nfa_delta nfa q c, delta qs c]
    in delta }
```

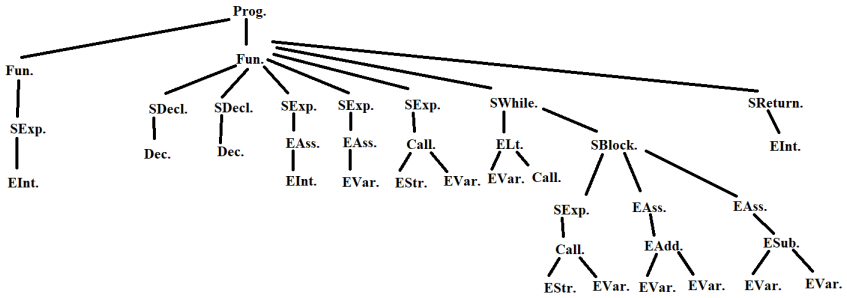
The code above is my modifications to the automata05.hs code that converts a non epsilon NFA to a DFA. I have run it with several example strings and have found it to work. The first function dfa_initial is simple because there are no epsilon transitions here. Therefore, we can simply set the initial state of the NFA to the initial state of the DFA. If there were epsilon transitions, we would also have to account for any possible states that connect to the initial state with an epsilon transition. For the dfa_final function, we just need to find all states that contain one of the final states from the NFA. This is done using the disjunction and map functions to find if the list of states contains an accepting state. Finally, for the dfa_delta function, we need to recursively loop through the states that result from an input given the starting state. Both the input and output are a list of states, so we must use the concat function to keep the list as a list of states and not a list of lists.

2.4 Week 4

Concrete syntax tree for Fibonacci function:



Abstract syntax tree:



2.5 Week 10

Proof tree for the code below that the interpreter will generate:

```
int x ;
    {   x = 2 ;
        bool x = false && x ;
        y = y++ + ++y ; }
    x = y ;
```

γ.[] ⇒ 2 ⇓ <2, γ.[]>	γ.[] ⇒ false ⇓ <false, γ.[]>	γ.[] ⇒ x ⇓ <2, γ.[]>
γ.[] ⇒ x=2 ⇓ <2, [x=2].[]>	γ.[] ⇒ false && x ⇓ <2, γ.[]>	
	γ.[] ⇒ bool x = false && x ⇓ <x=2, γ.[]>	γ.[] ⇒ y ⇓ <3, γ.[]>
		γ.[] ⇒ y++ ⇓ <3, γ.[3+1]>
		γ.[] ⇒ y++ + ++y ⇓ <3,4, γ.[3+1+1+3]>
	[x=NULL] ⇒{ x=2; bool x = false && x; y=y++ + ++y;} ⇓<x=2,y=9>	γ.[] ⇒ y=y++ + ++y ⇓ <3,4, γ.[(3+1)+1+1+3]>
	Int x ⇒{ x=2; bool x = false && x; y=y++ + ++y;} ⇓<x=2,y=9>	γ.[] ⇒ y ⇓ <9, γ.[]>
		γ.[] ⇒ x=y ⇓ x=9,y=9.[]
	⇒int x;{ x=2; bool x = false && x; y=y++ + ++y;} x=y;⇓<x=9,y=9>	

3 Project

3.1 Introduction

For the project, I decided to look at how C++ compiles into assembly code using the gcc compiler. I will be using an old data structures assignment with linked lists as an example and the website godbolt.org to compile into assembly. The repository for the example code can be found here. As our first example, we can look at the ListNode.h header file and compile it into assembly.

```
__static_initialization_and_destruction_0(int, int):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        mov     DWORD PTR [rbp-8], esi
        cmp     DWORD PTR [rbp-4], 1
        jne     .L3
        cmp     DWORD PTR [rbp-8], 65535
        jne     .L3
        mov     edi, OFFSET FLAT:_ZStL8__ioinit
        call    std::ios_base::Init::Init() [complete object constructor]
        mov     edx, OFFSET FLAT:__dso_handle
        mov     esi, OFFSET FLAT:_ZStL8__ioinit
        mov     edi, OFFSET FLAT:_ZNSt8ios_base4InitD1Ev
        call    __cxa_atexit
.L3:
        nop
        leave
        ret
_GLOBAL__sub_I_example.cpp:
```

```
push   rbp
mov    rbp, rsp
mov    esi, 65535
mov    edi, 1
call   __static_initialization_and_destruction_0(int, int)
pop    rbp
ret
```

This is much less code than would typically be seen in a .cpp file because a header file is just meant to declare constructors and functions of a given class. This one in particular is shorter because there are no functions being implemented here. This class is just supposed to represent the nodes of a linked list and are not very complex.

4 Conclusions

References

[HMU] John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman: Introduction to automata theory, languages, and computation, 3rd Edition. Pearson international edition, Addison-Wesley 2007