

Contents

| | |
|---|-----------|
| Course description | 1 |
| 1 Exercises 2019 | 2 |
| Week 1 | 2 |
| page 84, question 1.7 | 2 |
| page 84, question 1.7 | 4 |
| Solve the following problem, | 4 |
| Week 2 | 5 |
| page 86, question 1.16 | 5 |
| page 86, question 1.17 | 5 |
| page 86, question 1.18 | 6 |
| page 86, question 1.19 a | 7 |
| page 86, question 1.20 | 7 |
| page 86, question 1.21 b | 7 |
| page 86, question 1.29 | 8 |
| page 88, question 1.30 | 8 |
| page 89, question 1.36 | 8 |
| week 3 | 8 |
| page 88, question 1.29(a),(b) and 1.30 | 8 |
| page 89. question 1.35 | 8 |
| page 91. question 1.51 | 8 |
| page 154. question 2.2 | 9 |
| page 154. question 2.4 | 9 |
| 2.6 | 10 |
| d | 10 |
| 2.14 | 10 |
| page 156, question 2.16 | 10 |
| page 158, question 2.32 | 11 |
| page 158, question 2.38 | 11 |
| page 158, question 2.42 | 11 |
| week 4 | 12 |
| 2.58 | 12 |
| 2002 program 2 | 12 |
| 2 Questions 2018 | 13 |
| 1. Finite automata and regular languages | 14 |
| Introduction | 14 |
| Regular languages Closure under | 14 |
| Deterministic And Non-deterministic | 14 |
| Pumping lemma for regular languages | 14 |
| Example -> Pumping lemma or convert NFA to DFA | 15 |
| 2. Pushdown automata and context-free languages | 16 |
| Introduction | 16 |
| Pushdown automata | 16 |
| Context free grammar | 16 |
| Pumping lemma of CFL | 16 |
| Chomsky normal Form | 17 |

| | |
|---|----|
| Example with CFG \rightarrow CNF | 17 |
| 3. Turing machines | 18 |
| Introduction | 18 |
| What is a turing machine | 18 |
| Different types of Turing machines | 18 |
| Enumerators | 18 |
| example/proof - Turing recognizable/decidable | 18 |
| 4. Decidability | 19 |
| 5. Reducibility | 20 |
| 6. NP-completeness proofs – examples. | 21 |
| 7. Proof that SATISFIABILITY is NP-complete (do not assume that there is a known NP- Complete problem — use the proof in Sipser’s book). | 22 |
| 8. Information-theoretic lower bounds (lower bounds proven by counting leaves in decision trees), especially the average case bounds for sorting by comparisons. | 23 |
| 9. Adversary arguments – technique, examples. | 24 |
| 10. Median problem – algorithm and lower bound. | 25 |
| 11. Approximation algorithms | 26 |

Course description

Chapter 1

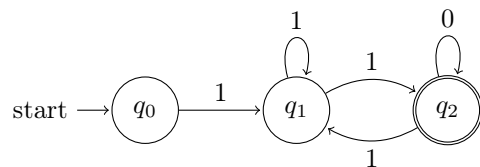
Exercises 2019

Week 1

page 84, question 1.7

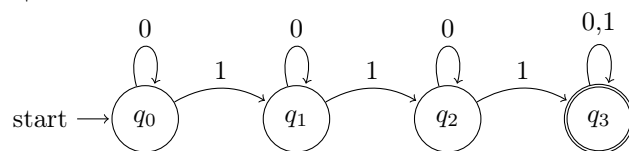
a

$w|w$ begins with a 1 and ends with a 0



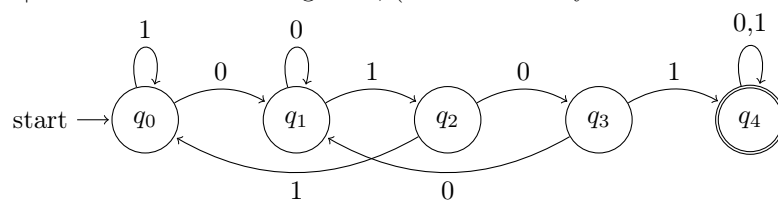
b

$w|w$ contains at least 3 1's



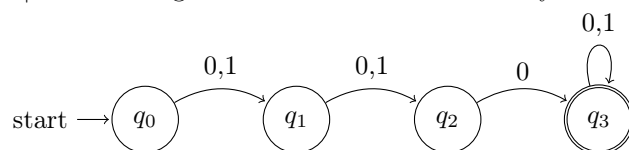
c

$w|w$ contains the substring 0101, (i.e. $w = x0101y$ for some x and y)



d

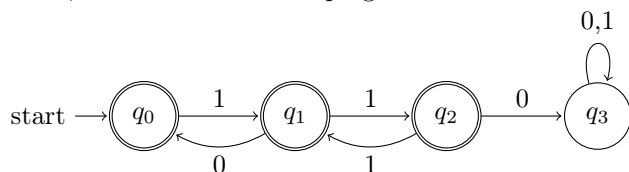
$w|w$ has at length of at least 3 and it's third symbol is 0



f

$w|w$ doesn't contain the substring 001

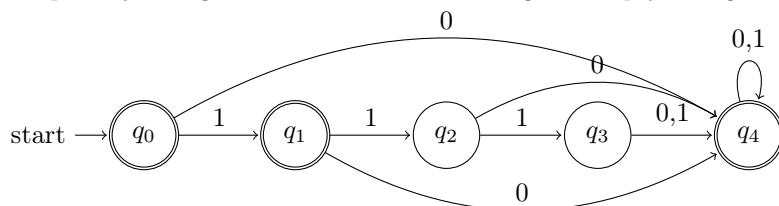
Accepts any string that doesn't contain the substring 001, and loops in rejecting state if this state is found, was unsure if the looping was needed but included it if it was the case.



h

$w|w$ is any string except 11 and 111

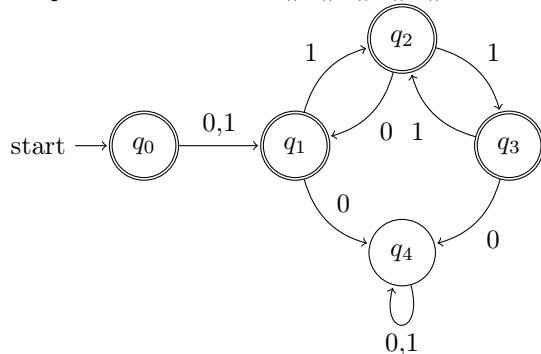
accepts any string that isn't 11, 111 including the empty string ϵ



i

$w|w$ every odd position of w is a 1

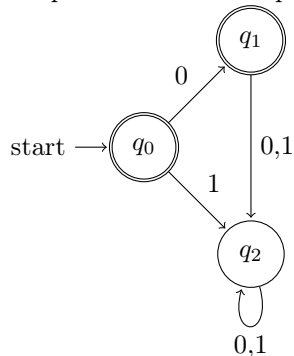
accepts all states where $\#1\#1\#1\#1\#1$ is followed also accepts the empty string ϵ



k

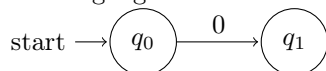
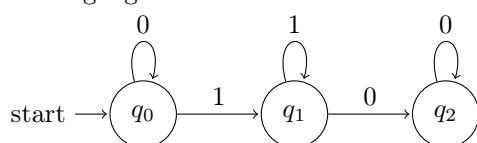
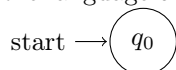
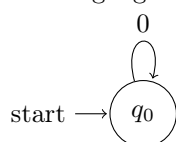
$w|w$ is only the empty string and 0

accepts "0" and the empty string ϵ



page 84, question 1.7**d**

The language 0 with two states,

**e**the language $0^*1^*0^*$ with 3 states**g**the language ϵ with 1 state**h**The language 0^* with 1 state**Solve the following problem,**

A man is travelling with a wolf (w) and a goat (g). He also brings along a nice big cabbage (c). He encounters a small river which he must cross to continue his travel. Fortunately, there is a small boat at the shore which he can use. However, the boat is so small that the man cannot bring more than himself and exactly one more item along (from w, g, c). The man knows that if left alone with the goat, the wolf will surely eat it and the goat if left alone with the cabbage will also surely eat that. The man's task is hence to devise a transportation scheme in which, at any time, at most one item from w, g, c is in the boat and the result is that they all crossed the river and can continue unharmed.

a

Describe a solution to the problem which satisfies the rules of the "game". You may use your answer to (b) to find a solution.

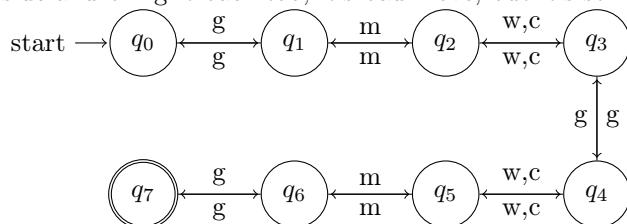
- First you carry the goat to the other side, and go back empty.
- The you ferry the wolf to the other side, and swap with the goat and bring the goat back.
- you then swap the goat with the cabbage and bring it to the other side.
- lastly you head back empty and bring the goat.
- you now have all the items on the other side of the river.

b

The string all of the valid moves are

$$\begin{aligned}
 &(g(m(x(g(y(m(g)*))*))*))* \\
 &\quad x, y = (w|c) \\
 &\quad x \neq y
 \end{aligned}
 \tag{1.1}$$

This is due to the fact that it's legal moves in the game, like the man can bring the goat to the other side and bring it back too, it's bad move, but it's still a valid move.



Week 2

page 86, question 1.16

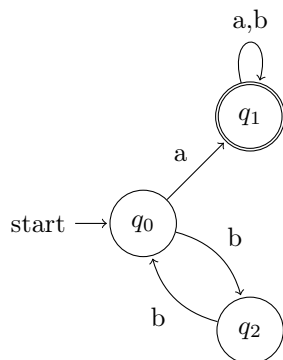
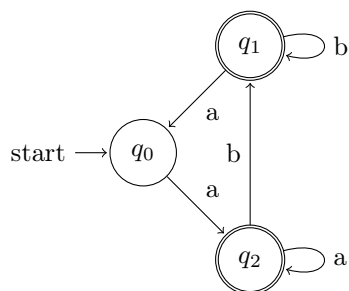
* is zero or more

! is one or more

a

it accepts $(bb)^*a!(a|b)^*$

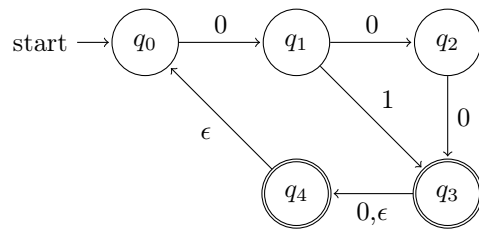
the language $0^*1^*0^*$ with 3 states

**b**

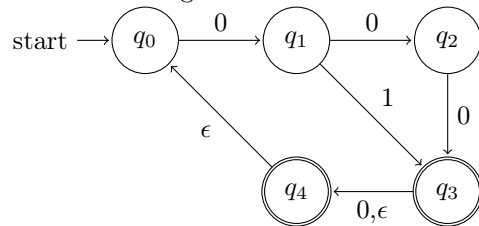
page 86, question 1.17

a

The first task is to make a NFA recognizing $(01u001u010)^*$

**b**

After converting this to a DFA

**page 86, question 1.18**

Predefined terms

$$x = (0, 1)^* \quad (1.2)$$

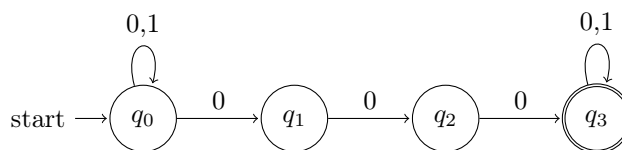
$$y = (0|1) \quad (1.3)$$

a $1x0$ **b** $x1x1x1x$ **c** $x0101x$ **d** $yy0x$ **e** $(0(yy)^*|1y(yy)^*)$ **f** $0^*(1+10)^*$ **g** $yyyyyy$

page 86, question 1.19 a

Convert the following regex to a NFA via lemma 1.55

$$(0 \cup 1)^* 000(0 \cup 1)^* \quad (1.4)$$

**page 86, question 1.20**

For each of the following expressions give two strings that are and two that are not in the languages. assume that the alphabet is $\{a,b\}$

a

$$a^* b^* \quad (1.5)$$

member
aa, ab, aabb, aab, abbbb
not member
ba, abab,

e

$$\sum^* a \sum^* b \sum^* a \sum^* \quad (1.9)$$

member
todo
not member
todo

b

$$a(ba)^* b \quad (1.6)$$

member
abab, ababababab,
not member
aaba baba

f

$$aba \cup bab \quad (1.10)$$

member
todo
not member
todo

c

$$a^* \cup B^* \quad (1.7)$$

member
ab, aabb
not member
ba, aabba

g

$$(\epsilon \cup a)b \quad (1.11)$$

member
ab, b
not member
abb, aab

d

$$(aaa)^* \quad (1.8)$$

member
aaa, aaaaaa
not member
a, aaaa

h

$$(a \cup ba \cup bb)^* \quad (1.12)$$

member
todo
not member
todo

page 86, question 1.21 b

The solution is

$$((a|b)(a,bb)^* b(a)?)^* \quad (1.13)$$

(a or b, followed by any number of (a,bb)* followed by a single b (as it matches uneven numbers of b and followed by 0 or 1 a*

page 86, question 1.29**a**

For the task a we have the string

$$0^n 1^n 2^n \quad (1.14)$$

the first contradiction is that when we have a string eg. 000111222 and we split it so we have the following, $x = 000$, $y = 111$, $z = 222$ and we pump y so we have the string $xyyz$ this violates the first condition of the pumping lemma, as we'll have more 1s than 0s and 2s.

the string y only contains 1s which also causes a contradiction.

and the 3rd case if we have the string $x = 00$, $y = 011$ and $z = 1222$ where we'll get out of order letters so we'll again reach a contradiction.

b

For assignment b i find it a bit odd, i may misunderstand the exercise but w/e

We want to pump the language

$$a_2 = \{www | w \in \{a, b\}^*\} \quad (1.15)$$

But from my understanding is the $*$ a Kleene star? eg then one w and www are equivalent as $\{a, b\}^*$ is all possible strings in the language w . ? or is it understood such that w is equal to either a or b but any number of them eq $\{a|b\}^*$,

How i choose to interpret it for now is that w is equal to either the string a^* or b^* and if this is the case then some of the same argumentation as for task a is valid.

www can give us the string 111000111 and we can split it as follows. 111|000|11111, This means that $zyyx$ gives of a out of order 1 and we therefore get a contradiction wrt. to the pumping lemma.

page 88, question 1.30

The error There's a few things here, the first case from 1:73 fail right away as the string 0001111 is in the language, the case of out of order fails as if we can get the string 00100111 then we'll reach a contradiction but the case of

page 89, question 1.36

For the language w there exist a DFA that accepts it, for the reverse language w^r the DFA which a reversed edges and the start state is now our accept state.

week 3**page 88, question 1.29(a),(b) and 1.30**

Already done, see above.

page 89. question 1.35

As B is in bijection of A therefore it's a regular.

page 91. question 1.51**a**

This can be proved via the pumping lemma.
where you can get out of order 1 and/or 0s and that causes a contradiction.

b

Same as a wrt. out of order.

c

This is this is either the unbounded string 0 or 1.

d

this is again wrt. out of order 0.1 eg. if we can get the string 01010 by pumping as this is not in the language.

page 154. question 2.2

Give parse trees and derivations for each string using the following Context free grammar (CFG)

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow E \times F \mid F \\ F &\rightarrow (E) \mid a \end{aligned}$$

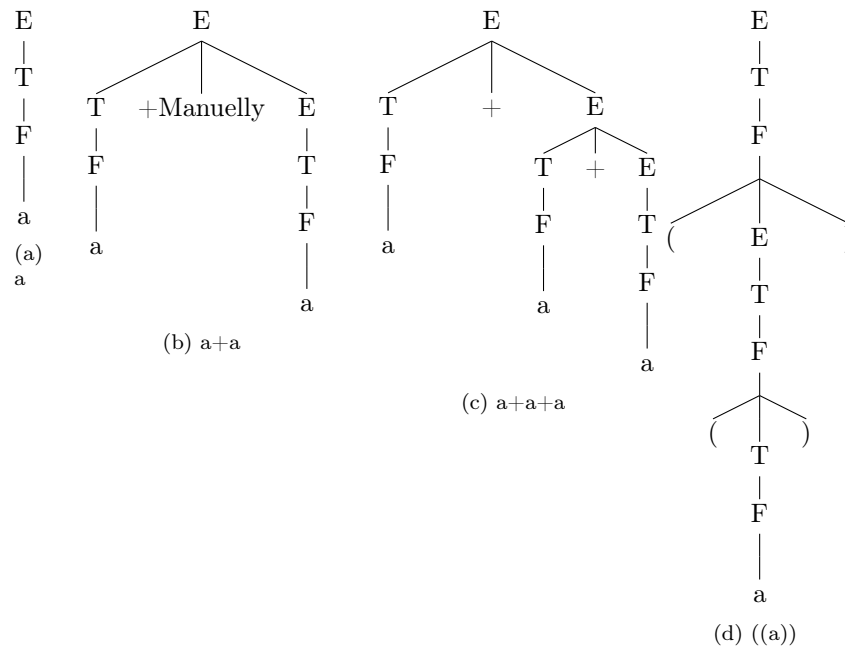


Figure 1.1: 2 Figures side by side

page 154. question 2.4**a**

$$\begin{aligned} S &\rightarrow TTT \\ T &\rightarrow E1E \\ E &\rightarrow FTF \\ F &\rightarrow 0 \mid \epsilon \end{aligned}$$

b

$$\begin{aligned} S &\rightarrow 0F0 \mid 1F1 \\ F &\rightarrow F0F \mid F1F \mid \epsilon \end{aligned}$$

c

$$\begin{aligned} S &\rightarrow EFE \\ E &\rightarrow FEF \mid EFF \mid FFE \mid \epsilon \\ F &\rightarrow 0 \mid 1 \end{aligned}$$

d

$$\begin{aligned} S &\rightarrow E0E \\ E &\rightarrow FEF \mid \epsilon \\ F &\rightarrow 0 \mid 1 \end{aligned}$$

e

$$\begin{aligned} S &\rightarrow E \\ E &\rightarrow FEF \mid GEG \mid \epsilon \\ F &\rightarrow 0 \quad G \rightarrow 1 \end{aligned}$$

f

$$S \rightarrow \epsilon$$

2.6

Give the CFG that generates the following languages

b

The complement of the languages $\{a^n b^n | n \geq 0\}$

$S \rightarrow FG$
 $G \rightarrow GbG$
 $F \rightarrow FaF$

d

For the third one it's a unbounded string with up to k alternations of $0^*1^*0^*1^*0^*$

This can simply be defined using the following grammar.

$S \rightarrow FG$
 $E \rightarrow GE|FE|\epsilon$
 $G \rightarrow 1G|\epsilon$
 $F \rightarrow 0F|\epsilon$

2.14

We start in the initial state

$A \rightarrow BAB|B|\epsilon$
 $B \rightarrow 00|\epsilon$

From here we put a new start state S

$S \rightarrow A$
 $A \rightarrow BAB|B|\epsilon$
 $B \rightarrow 00|\epsilon$

From here we can eliminate the first ϵ

$S \rightarrow A$
 $A \rightarrow BAB|B|\epsilon|BA|AB$
 $B \rightarrow 00|\epsilon$

From here we can eliminate the ϵ in our 2nd rule,

$S \rightarrow A|BAB|B|BA|AB|\epsilon|CC$
 $A \rightarrow BAB|B|\epsilon|BA|AB|CC$
 $B \rightarrow 00|\epsilon|CC$
 $C \rightarrow 0$

page 156, question 2.16

Show that the class of context free languages are closed under the regular operations Concatenation, union and Star

Using the following grammar

- $S_1 \rightarrow aS_1b$
- $S_1 \rightarrow \epsilon$
- $S_2 \rightarrow cS_2d$
- $S_2 \rightarrow \epsilon$

Concatenation

This is simply shown via

Making s start symbol S , and have the two languages follow each other $S_1 S_2$

$S \rightarrow S_1 S_2$

Union

This is simply shown via
 Making s start symbol S,
 $S \rightarrow S_1 | S_2$

Star

This is simply shown via
 Making s start symbol S,
 $S \rightarrow S_1 S$

page 158, question 2.32

Let $A/B = \{w | wx \in A \text{ for some } x \in B\}$, Show that if A is context free and B is regular, then A/B is context free.

"Proof. We can augment the memory of the PDA recognizing the CFL with the DFA recognizing the regular language and run both machines in parallel. We accept iff both machines accept.??
 note the intersection of two CFL's and not necessarily a CFL.!

page 158, question 2.38

As each step has a most 2 sub rules we can calculate that at step 1 we increasing the string length by $2n-1$ and as all rules do this our end case is $2n-1$

page 158, question 2.42

• 2.42 Hint for (d): first intersect the language with a suitably chosen regular language and then prove that the language you obtain is not context-free.

a

it to a length not in this range,

A break wrt. out of order letters, and not n
 counts of some letter

c

i dont know

b

For the lang b is has to be in the size of wrt.
 $n + n^2 + n^3$ eg 3,14, 39, 84, 155 and you can pump

d

i dont know

week 4**2.58**

let $\Sigma = \{0, 1\}$ and let B be the collection of strings that contain at least one 1 in their second half, in other words, $B = \{uv \mid u \in \Sigma^*, v \in \Sigma^* 1 \Sigma^* \text{ and } |u| \geq |v|\}$

a

Give a PDA that recognizes B

$$S \rightarrow XT X | X1$$

$$T \rightarrow XT | X$$

$$X \rightarrow 1 | 0$$
b

Give a CFG that generates B Read the input from end to front, pushing every 0 you meet onto the stack. if a 1 is found you continue parsing but instead pop a letter from the stack until it's empty, if you reach the beginning of the input before the stack is empty you reject.

2002 program 2

Let Σ be a finite alphabet and let $L \subset \Sigma^+$ be a regular language, Define a new language L' as the language of all words $w \in \Sigma^*$ such that w is obtained from a word in L by deleting the first letter, is the language L' regular as well? prove your answer

Chapter 2

Questions 2018

1. Finite automata and regular languages

Introduction

I'm going to talk about Finite automata and regular languages.

Finite automata

Finite automata is the simplest computational model that works via states and transitions, and therefore uses extremely limited memory. Using this model we can recognize and formulate regular languages. This can be a simple task such as finding a substring, or used as a tool for designing more complex systems. A Finite automata is defined as a tuple containing:

- The set of states Q ,
- The known alphabet Σ ,
- The transition function $\delta : Q \times \Sigma \rightarrow Q$
- the start state q_1
- and the set of accept states F .

Regular languages

A regular language is a sequence of letters in some alphabet defined by Σ the empty alphabet is defined by the empty set \emptyset , and contains letters and the empty string ϵ . They are also closed under the union \cup , concatenation \cap , and Kleene star $*$, and the preceding order is $*, \cap, \cup$. A language is Regular if a Finite automata recognizes it.

Add definitions

Regular languages Closure under

TODO

Union

Concatenation

Intersection

Kleene star

Deterministic And Non-deterministic

The difference between deterministic and non-deterministic is the way they operate. They recognize the same class of languages as a NFA can be converted into a DFA and the same goes the other way around, it is not an efficiency operation as the algorithm is exponential. There are, however, some benefits to both, where the non-deterministic performs branching and could potentially benefit the execution wrt. size, runtime, or resource consumption.

The correct term here would be that a DFA can simulate a NFA

Pumping lemma for regular languages

The pumping lemma is a way for us to prove if a language is regular. The theorem for the pumping lemma states that the 3 conditions for the lemma are

- for each $i \geq 0, xy^iz \in A$
- $|y| > 0$
- $|xy| \leq p$ where p is the pumping length.

$0^n 1^n | n \geq 0$

The way you do this is to assume it's regular and make a counter argument. With the pumping lemma we have 3 conditions that our counter argument must meet. The first case we can pump the language to have more 1's than 0's or (2) the other way around.. The third (3) is that we can get out of order letters if we pump a string containing both 0's and 1's,

Example -> Pumping lemma or convert NFA to DFA

TODO exam with pumping lemma

2. Pushdown automata and context-free languages

Introduction

I'm going to talk about Pushdown automata and context free languages. These topics are used in modern compilers to parse a programming language and is a useful tool in language processing wrt. human language as well as when interpreting one language to an other.

Pushdown automata

A pushdown automata is a type of automaton that employs a stack to help with the computation, this allows the PDA to recognize and run Context free grammars and all the languages below it (Regular languages).

- The set of states Q ,
- The known alphabet Σ ,
- Γ the stack alphabet
- The transition function $\delta : Q \times \Sigma_{\epsilon} \times \Gamma_{\epsilon} \rightarrow Q$
- the start state q_0
- the set of reject states F

Context free grammar

A CFG consist of a collection of substitution rules, a CFG operate on a set of variables, terminals and a designated start symbol. an exsample of such a rule is:

- $A \rightarrow 0A1$
- $A \rightarrow B$
- $B \rightarrow \#$

This above grammar should generate the string $0^n \# 1^n$ $n \in \mathbb{N}$.

The formal definition of a CFG is a 4 tuple consisting of the set of variables V , the set of terminals Σ , the set of rules R and the start symbol S that is in V

A more advanced example could be the over the set of variables and terminals $\{\text{num}, A, -\}$

Lets say we have the following rules

$A \rightarrow A - A$

$A \rightarrow \text{num}$

for the above grammar if we apply it to the grammar

1-2-3 we can get the output 2 or -4 depending on how the grammar is parsed. (1-2)-3 vs 1-(2-3)

We can handle this ambiguity in two manners either we introduce the terminals (and) to the language or we can rewrite or rules as to not have this bias.

Pumping lemma of CFL

The pumping lemma is a way for us to proof if a language is regular. The theorem for the pumping lemma states that the 3 conditions for the lemma is

- for each $i \geq 0, uv^i xy^i z \in A$
- $|vy| > 0$
- $|vxy| \leq p$

example

The way you do this is to assume it's regular and make a counter argument. with the pumping lemma we have 3 conditions that our counter argument must meet.

$a^n b^n c^n | n \geq 0$

- if vy only contain the same symbol the string cannot contain an equal number of letters as required
- when either v or y contains more than 1 type of symbol we get an out of order contradiction

Chomsky normal Form

Chomsky normal form is a simplified form that we can convert our grammars into which enables to decide things like if a string is generated by a grammar in polynomial time.

The steps to convert a grammar to CNF is to:

1. eliminate all ϵ productions.
2. Eliminate all productions where RHS is one variable
3. Eliminate all productions longer than 2 variables
4. move all terminals to productions where RHS is one terminal.

Example with CFG \rightarrow CNF

We start in the initial state

$$A \rightarrow BAB|B|\epsilon$$

$$B \rightarrow 00|\epsilon$$

From here we put a new start state S

$$S \rightarrow A$$

$$A \rightarrow BAB|B|\epsilon$$

$$B \rightarrow 00|\epsilon$$

From here we can eliminate the first ϵ

$$S \rightarrow A$$

$$A \rightarrow BAB|B|\epsilon|BA|AB$$

$$B \rightarrow 00|\epsilon$$

From here we can eliminate the ϵ in our 2nd rule,

$$S \rightarrow A|BAB|B|BA|AB|\epsilon|CC$$

$$A \rightarrow BAB|B|\epsilon|BA|AB|CC$$

$$B \rightarrow 00|\epsilon|CC$$

$$C \rightarrow 0$$

3. Turing machines

Introduction

In this talk i'll talk about turning machines and their use in computer science, A turning machine is a theoretical form of computer in the same sense as the other models but it's the closets one to modern computers and we can use it for decide things wet. to run time, and compatibility of problems and algorithms.

What is a turing machine

What is a turing machine, The model itself is of a tape and a read/write head that can move left or right on the tape.

- Q is the set of states
- Σ is the input alphabet not containing the blank symbol
- τ is the tape alphabet where $\epsilon \in \tau$ and $\Sigma \subseteq \tau$
- $\delta : Q \times \tau \rightarrow Q \times \tau \times \{L, R\}$ is the transition function.
- $q_0 \in Q$ is the starte states
- $q_{accept} \in Q$ is the accept state
- $q_{reject} \in Q$ is the reject state where $q_{reject} \neq q_{accept}$

Different types of Turing machines

All turning machines are equal in power, show why.

Multitape

The idea is to show that a multitape turing machine TM_m can be simulated on a equivalent single tape turing machine. this is done by storing the tapes on a single tape and discriminating them via the letter #,

And we use dot to mark the position of the read head on the underlying tapes. If we reach the condition where one of the simulated readheads reaches a # symbol on the right side of the simulated tape we write a \sqcup and shift the contents of the tape by 1,

Nondetermanistic turning machine

A ND turing machine can be simulated by running a tree search to find accepting state for the desired configuration. and we'll want to approach this in a breath first search as using a dept first can result in following a infinite branch that never reaches a accept state, therefor it's better running a breath first as we're guaranteed to find a accept state should one exist but the issue of looping still exists. We further call a non deterministic turing machine a decider if all branches halt on all input.

Enumerators

Is a turing machine attached to a printer, basically it generates all possible outputs for a set configuration.

Theorem 3.21

A language is only turing recognizable if and only if some enumerator enumerates it.

example/proof - Turing recognizable/decidable

TODO

4. Decidability

Introduction If a language defined by a DFA is decidable.

Examples of Decidability and undecidability

5. Reducibility

What is it and how to use it.

6. NP-completeness proofs – examples.

what is np_completeness

Why we use it to reduce,

Proff of np complete. clique, subset sum, Hamiltonian circuit,

7. Proof that SATISFIABILITY is NP-complete (do not assume that there is a known NP-Complete problem — use the proof in Sipser's book).

Cook-levin theorem - in sipset, presentation use slides on homepage.

8. Information-theoretic lower bounds (lower bounds proven by counting leaves in decision trees), especially the average case bounds for sorting by comparisons.

As is average case.

9. Adversary arguments – technique, examples.

10. Median problem – algorithm and lower bound.

11. Approximation algorithms

vertex cover