Contents

Course description

The aim of the course is to enable the student to

- Apply formalisms of formal languages in order to formulate decision problems precisely
- Construct finite automata, regular expressions, push-down automata and context-free grammars as elements in an algorithmic solution of more complicated problems.
- Decide the complexity of new problems based on knowledge of the complexity of important examples of problems from the course.
- Judge whether a given problem may be solved by a computer or is undecidable.
- Argue that problems are NP-complete.
- Judge the possibility to develop an approximation algorithm for a given NP-hard optimization problem.
- Give lower bounds for the complexity of problem that are similar in nature to those studied in the course.

These competencies are important both when one wishes to develop new algorithms for a given problem and when one wants to judge whether a given problem may be possible to solve efficiently (possibly only approximately) by a computer.

The course builds on the knowledge acquired in the courses DM507 Algorithms and data structures and DM551 Algorithms and probability.

The course forms the basis for doing a bachelor project as well as elective candidate level courses containing one or more of the following elements: complexity of algorithms, approximation algorithms and computability.

Together with courses as above this course also provides a basis for doing a masters thesis on algorithmic and complexity theoretic subjects.

In relation to the competence profile of the degree it is the explicit focus of the course to:

- Give the competence to analyze complexity of (decision) problems.
- Give knowledge about the computational power of different models of computation.
- Enable the student to construct finite automata and regular expressions for simple languages.
- Enable the student to construct push-down automata and context-free grammars for simple languages.
- Equip the students with important tools to prove that a given language cannot be recognized by a finite automation, a push-down automaton or a Turing machine.
- Enable the student to prove lower bounds for the complexity of algorithms for a given problem.
- Enable the student to develop new approximation algorithms.
- Give the student important tools for proving that a given decision problem is NP-complete or undecidable.

Aims

- Judge the complexity of (decision) problems.
- Judge the computational power of various models of computation.
- Construct finite automata and regular expressions for simple languages.

- Construct push-down automata and context-free grammars for simple languages.
- Prove that a given language, which in nature resembles those from the course, cannot be recognized by a finite automaton, a push-down automaton or a Turing machine.
- Prove lower bounds for the complexity of algorithms for a given problem which in nature resembles those from the course.
- Design new approximation algorithms for a given problem which in nature resembles those from the course.
- Prove that a given decision problem which in nature resembles those from the course is NP-complete or undecidable.

Chapter 1

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 therefor uses extremely limited memory. Using this model we can recognize and formulate regular languages, This can be a simple tasks a finding a substring, or used as a tool for designing more complex systems.
A Finite atomata is defined as a tuple containing:

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

Regular languages

A regular languages is a sequence of letters in some alphabet defined by $\sum$ the empty alphabet is defined by the empty set $\varnothing$, and contains letters and the empty string ϵ. they are also closed under the union $\cup$, concatination $\cap$, and kleene star *, and the proceeding order is $^*, \cap, \cup$, A language is Regular is a Finite automata recognizes it.
Add definitions

Regular languages Closure under

TODO
Union
Concatenation
Intersection
Klein 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 convert into a DFA and the same goes the other way around, it is not a efficiency operation as the algorithm is exponential. there is however some benefits to both. where the non-deterministic preforms branching and could potentially benefit the execution wrt. size, runtime. or recourse consumption.
The correct term here would that that a DFA Can simulate a NFA

Pumping lemma for regular languages

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

$0^n1^n|n \geq 0$
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 most meet. the first case we can pump the language to have more 1's than 0s 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 0s and 1s,

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 compilers to parse a programming language and is a useful tool in language processing 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 as well as the languages within(regular languages).
The formal definition is:

- The set of states Q,
- The known alphabet Σ (Sigma)
- The stack alphabet Γ (Gamma)
- The transition function $\delta(delta) : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \longrightarrow Q$
- the start state q_0
- the set of reject states 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.
The formal definition is

- The finite set of variables V
- The finite set of terminals Σ disjoint from V
- The finite set of rules R
- The start symbol $S \in V$

Exsample

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

This above grammar should generate the string $0^n\#1^n \ n \in \mathbb{N}$.
A different examples that show ambiguity is the set of variables and terminals {num,A,-}
With the rules

- $A \to A - A$
- $A \to num$

for the above grammar if we apply it to the string
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 issue.

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 2n-1
The steps to convert a gramma 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 -> CNF

We start in the initial state
$A \to BAB|B|\epsilon$
$B \to 00|\epsilon$

From here we put a new start state S

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

From here we can eliminate the first ϵ

$S \to A$
$A \to BAB|B|\epsilon|BA|AB$
$B \to 00$ ~~$|\epsilon$~~
From here we can eliminate the ϵ in our 2nd rule,

~~$S \to |A$~~ $|BAB|$~~B~~$|BA|AB|\epsilon|CC$
$A \to BAB|$~~$B|\epsilon$~~$|BA|AB|CC$
$B \to 00|$~~ϵ~~$|CC$
$C- > 0$

Pumping lemma of non-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 $i \geq 0, uv^ixy^iz \in A$
- $|vy| > 0$
- $|vxy| \leq p$

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 most meet.
$a^nb^nc^n|n \geq 0$

- if vy only contain the same symbol the string connot 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

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
- $\sum$ is the input alphabet not containing the blank symbol
- τ is the tape alphabet where $\in \tau$ and $\sum \subseteq \tau$
- $\delta : Q \times \tau \to Q \times \tau \times \{L, R\}$ 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 ␣ 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. qlique, 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

Chapter 2

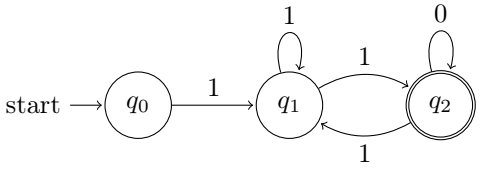
Exercises 2019

Week 1

page 84, question 1.7

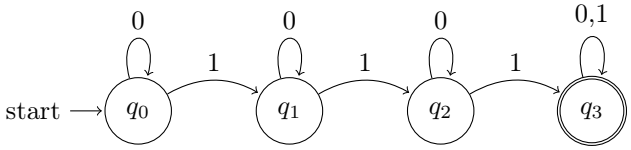
a

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



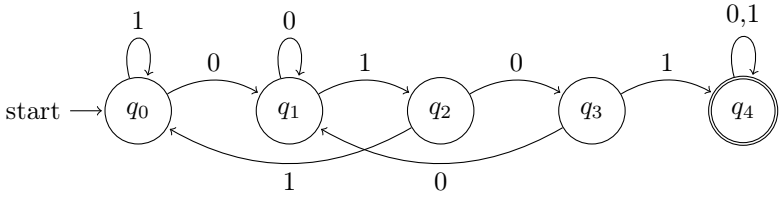
b

w|w contains at least 3 1's



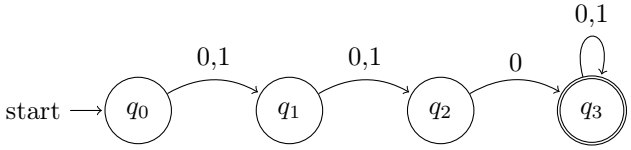
c

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



d

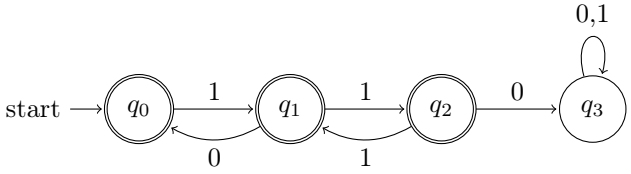
w|w bas at length of at least 3 and it's third symbol is 0



f

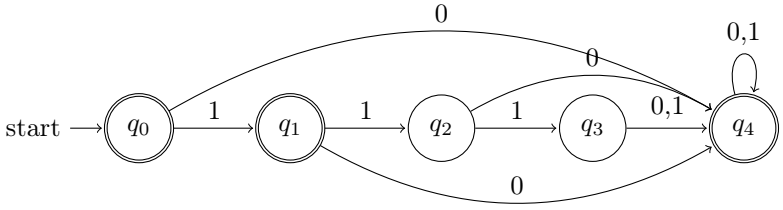
w|w doesn't contain the substring 001

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.



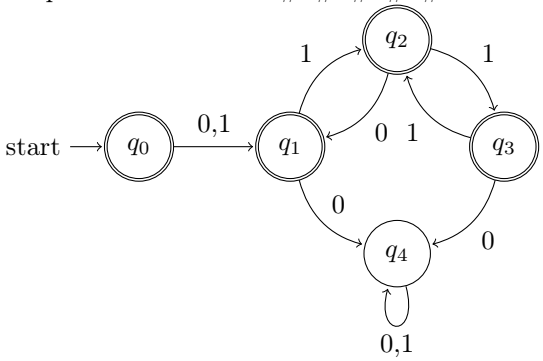
h

w|w is any string except 11 and 111
accepts any string that isn't 11, 111 including the empty string ϵ



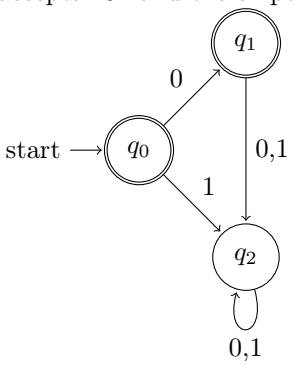
i

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

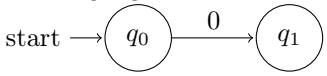
w|w is only the empty string and 0
accepts "0" and the empty string ϵ



page 84, question 1.7

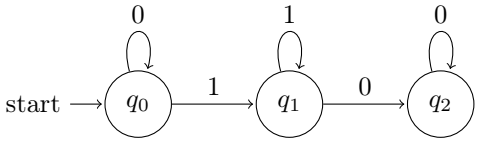
d

The language 0 with two states,



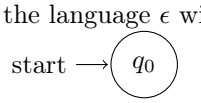
e

the language 0*1*0* with 3 states



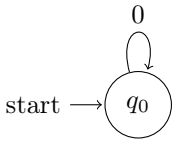
g

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 device 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.

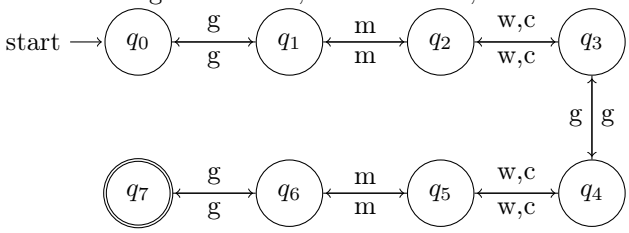
- First you carry the goat to the other side, and go back empty.
- The you ferry the wolf to the other side, and swap with the goat and bring the goat back.
- you then swap the goat with the cabbage and bring it to the other side.
- lastly you head back empty and bring the goat.
- you now have all the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

b

The string all of the valid moves are

$$\begin{aligned}(g(m(x(g(y(m(g)*)*)*)*)*)*\\ x, y = (w|c)\\ x \neq y\end{aligned} \tag{2.1}$$

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.



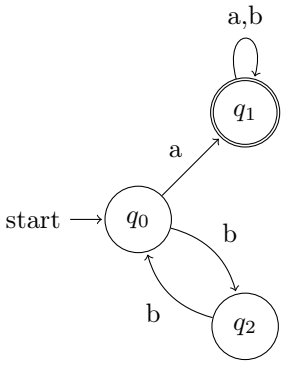
Week 2

page 86, question 1.16

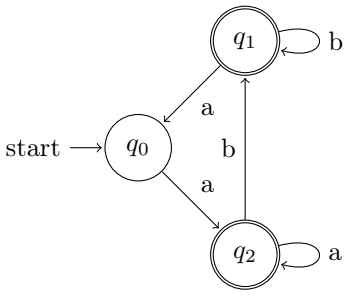
* is zero or more
! is one or more

a

it accepts (bb)*a!(a|b)*
the language 0*1*0* with 3 states



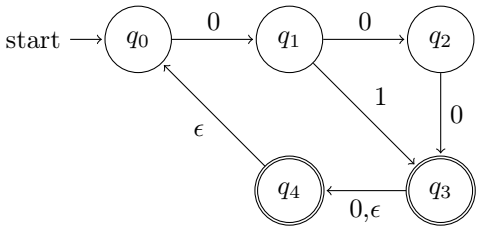
b



page 86, question 1.17

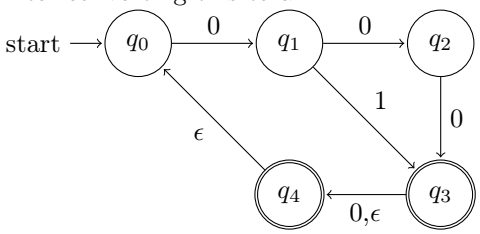
a

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



b

After converting this to a DFA



page 86, question 1.18

Predefined terms

$$x = (0,1)^* \tag{2.2}$$

$$y = (0|1) \tag{2.3}$$

a

$1x0$

b

$x1x1x1x$

c

$x0101x$

d

$yy0x$

e

$(0(yy)*)|1y(yy)^*)$

f

$0^*(1+10)^*$

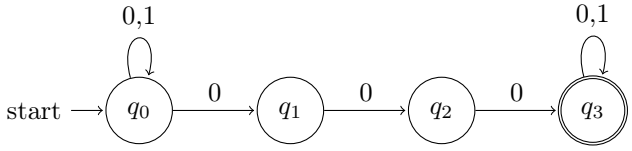
g

$yyyyy$

page 86, question 1.19 a

Convert the following regex to a NFA via lemma 1.55

$$(0 \cup 1)^*000(0 \cup 1)^* \tag{2.4}$$



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

$$a^*b^* \tag{2.5}$$

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

b

$$a(ba)^*b \tag{2.6}$$

member
abab, abababbabab,
not member
aaba baba

c

$$a^* \cup B^* \tag{2.7}$$

member
ab, aabb
not member
ba, aabba

d

$$(aaa)^* \tag{2.8}$$

member
aaa, aaaaaa
not member
a, aaaa

e

$$\sum^* a \sum^* b \sum^* a \sum^* \tag{2.9}$$

member
todo
not member
todo

f

$$aba \cup bab \tag{2.10}$$

member
todo
not member
todo

g

$$(\epsilon \cup a)b \tag{2.11}$$

member
ab, b
not member
abb, aab

h

$$(a \cup ba \cup bb) \sum^* \tag{2.12}$$

member
todo
not member
todo

page 86, question 1.21 b

The solution is

$$((a|b)(a, bb) * b(a)?)* \tag{2.13}$$

(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 sting

$$0^n 1^n 2^n \tag{2.14}$$

the first contradiction is that when we have a string eg. 000111222 and we split it so we have the flooowing, 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 then 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

$$a_2 = \{www | w \in \{a, b\}^*\} \tag{2.15}$$

But from my understanding is the * a kleen star? eg then one w and www are equivalent as {a,b} is all possibly strings in the library w. ? or is it understand such that w is equal to either a or b but any number of them eq $\{a|b\}^*$,
How i choose to intercept 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|1000|11111, This means that zyyx gives of a out of order 1 and we therefor 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 lang, the case of out of order fails as if we can get the string 00100111 ten we'll reach a contradiction but the case of

page 89, question 1.36

For the language w there exist a DFA that accepts if, for the revere 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 therefor 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. our 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 umping as this is not in the lang.

page 154. question 2.2

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

$$E \to E + T | T$$
$$T \to E \times F | F$$
$$F \to (E) | a$$

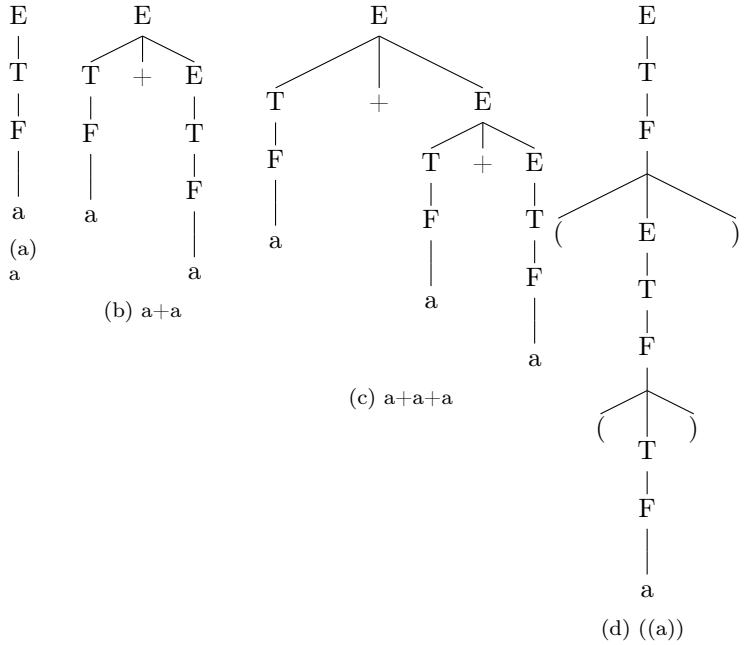


Figure 2.1: 2 Figures side by side

page 154. question 2.4

a

$S \to TTT$
$T \to E1E$
$E \to FTF$
$F \to 0|\epsilon$

b

$S \to 0F0|1F1$
$F \to F0F|F1F|\epsilon$

c

$S \to EFE$
$E \to FEF|EFF|FFE|\epsilon$
$F \to 0|1$

d

$S \to E0E$
$E \to FEF|\epsilon$
$F \to 0|1$

e

$S \to E$
$E \to FEF|GEG|\epsilon$
$F \to 0$ $G \to 1$

f

$S \to \epsilon$

2.6

Give the CFG that generates the following languages

b

The complment of the langauges $\{a^n b^n | n \geq 0\}$

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

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

2.14

We start in the initial state
$A \to BAB|B|\epsilon$
$B \to 00|\epsilon$

From here we put a new start state S

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

From here we can eliminate the first ϵ

$S \to A$
$A \to BAB|B|\epsilon|BA|AB$
$B \to 00 | \epsilon$
From here we can eliminate the ϵ in our 2nd rule,

$S \to | A |BAB|B|BA|AB|\epsilon|CC$
$A \to BAB|B| \epsilon |BA|AB|CC$
$B \to 00|\epsilon|CC$
$C -> 0$

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

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

Concatenation

This is simply shown via
Making s start symbol S, and have the two languages follow each other S_1 S_2
$S \to S_1 S_2$

Union

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

Star

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

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 langauge 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

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

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 it to a length not in this range,

c

i dont know

d

i dont know

week 4

2.58

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

a

Give a PDA that recognizes B
$S \to XTX|X1$
$T \to XT|X$
$X \to 1|0$

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 $\sum$ be a finite alphabet and let $L \subset \sum^+$ be a regular language, Define a new language L'as the language of all words w $\in \sum^*$ 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