DM553 - Kompleksitet og Beregnlighed

Grundige noter til kursets indhold og opgaver

Kevin Vinther

2024

Indhold

1

Endelige Automater

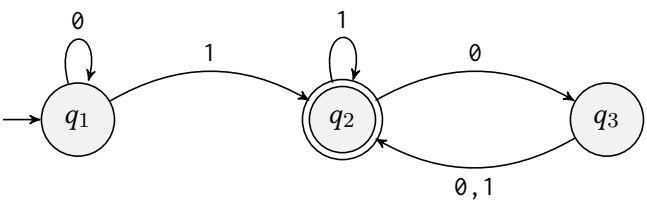
Endelige automater er en type abstrakt maskine med endelig hukommelse, som kan bruges til for eksempel at matche mønstre med.

1.1 Deterministiske Endelige Automater

I en deterministisk endelig automat (DFA) vil man altid vide hvilken state er den næste man havner i, efter et givet symbol. Dette bestemmes ud fra *transitionsfunktionen*, som tager en state, et symbol, og giver en ny state som output. Denne nye state er automatens næste state.

Vi kan beskrive en DFA grafisk ved hjælp af *state diagrams*. Et eksempel på et sådan diagram kan ses i Figur 1.1. Som kan ses på denne figur består den af nogle cirkler, `states`, nogle pile, `transition pile`, og en underlig dobbelt-cirkel, `accept state`. States er en form for hukommelse som en DFA kan være i på et givet tidspunkt. Efter at have læst tre karakterer af en streng, *kan* den eksempelvis være i q_3, eller q_{90}, eller q_{fisk}, eller bare $fisk$. Pointen er at navnet er ligegyldig, og der er

ikke en nødvendig rækkefølge for *alle* strenge. Den første pil, som peger ind til q_1 er startpilen, som indikerer *initial state*, altså den state der startes i når strengen skal læses. q_2, som er den state med dobbeltcirkler, er den *accepterende state*, og betyder at hvis en DFA ender i denne state når den er færdig med at læse en streng, er strengen accepteret. Hvis den ikke ender i denne state, er strengen ikke accepteret. Det skal siges at det er tilladt at have mere end en accept state.



Figur 1.1: Et eksempel på et state diagram for en automat M_1

Spørgsmålet om hvor mange accept states der kan være, samt hvordan transitionspilene fungerer, samt mere, kan forklares ved hjælp af en matematisk definition. Vi introducerer nu den formelle definition af en deterministisk endelig automat:

Definition 1.1
En deterministisk endelig automat er en 5-tuple $(Q, \Sigma, \delta, q_0, F)$, hvor:

1. Q er et endeligt sæt af states (hukommelse)
2. Σ er et endeligt sæt af symboler kaldet alfabetet
3. δ er *transitionsfunktionen*
4. q_0 er den state maskinen starter i (start staten)

5. F er et endeligt sæt af *accept* states ($F \subseteq Q$)

For at give et eksempel beskriver vi M_1 i Figur 1.1 formelt.
$M_1 = (Q, \Sigma, \delta, q_1, F)$, hvor

1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma = \{\texttt{0}, \texttt{1}\}$
3. δ beskrives som en tabel:

	`0`	`1`
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

4. q_1 er start staten
5. $F = \{q_2\}$

Vi siger at, hvis A er sættet af alle strenge som M accepterer, så er A sproget af maskinen M, eller $L(M) = A$. Ligeledes siger vi at M genkender A, eller M accepterer A. Dog er "genkender" mest brugt, da accept har andre betydninger her.

Læg mærke til at **hvis en maskine ikke accepterer nogen strenge**, så genkender den stadig et sprog: det tomme sprog, $\emptyset$.

Vi vil gerne have en bedre forståelse af hvordan en DFA egentlig *komputerer*, altså, hvordan den kommer fra A til B. Vi introducerer her en formel definition af komputering:

Definition 1.2 (Komputering for en DFA)
Lad $M = (Q, \Sigma, \delta, q_0, F)$ være en endelig automat, og lad $w = w_1 w_2 \cdots w_n$

være en streng hvor alle w_i er et symbol som er medlem af alfabetet Σ. M accepterer her w hvis en sekvens af states $r_0, r_1, \ldots, r_n$ i Q eksisterer med tre betingelser:

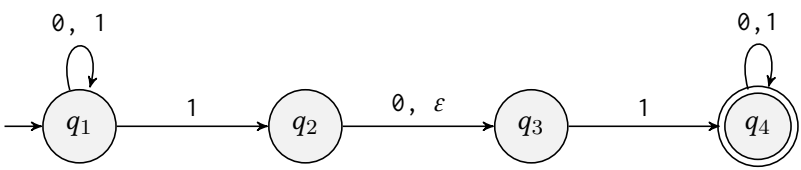
1. $r_0 = q_0$
2. $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n-1$
3. $r_n \in F$

Det vil altså sige at **betingelse 1** siger at maskinen starter i den initielle state. **Betingelse 2** siger at ud fra reglerne af transitionsfunktionen går maskinen fra state til state. **Betingelse 3** siger at maskinen accepterer dets input hvis den ender i en accept state.

Definition 1.3
Et sprog kaldes et **regulært sprog** hvis en endelig automat genkender det.

1.2 Nondeterministiske Endelige Automater

Som beskrevet i sektionen om deterministiske endelige automater, vil man altid vide, givet en state og et symbol, hvad den næste state er. Dette er hvad der gør DFA'er anderledes fra NFA'er. I nondeterministiske endelige automater er der ikke kun én, men flere mulige veje som den kan gå, givet en eller flere symboler. I Figur 1.2 ses et eksempel på et state diagram af en NFA.



Figur 1.2: En NFA N_1

Der springer med det samme nogle idéer ud af N_1 som ikke er set før. Specielt med den ekstra information at ε (den tomme streng) **ikke** er en del af alfabetet! Allerede fra den første state kan du se at der er to pile der har symbolet `1` på sig. I en DFA ville dette være ulovligt, men i en NFA er det lovligt. Lad os gå state diagrammet igennem, og forstå hvad der sker. Vi har en initial state, q_1 som har pile både til sig selv, og ud til q_2. I isolation ligner pilene noget vi kunne se fra en DFA, men givet at `1` er en del af to pile, kan det godt fovrirre lidt. Det der sker her, er at maskinen laver *kopier* af sig selv. Frem for kun at gå til q_2 eller q_1 går den til **både** q_2 *og* q_1. Det vil altså sige, at næste gang den skal læse et symbol, vil den både være i state q_1 og q_2. Her, hvis vi får `0` som input, vil vi blive i q_1 i den maskine der self-loopede, og vi vil gå til q_3 i den anden maskine. Men faktisk sker der noget før vi overhovedet kigger på det næste input symbol. Når en maskine når til en state med

et epsilon symbol ε, så laver den en kopi til den state som epsilon peger til. Det vil sige, at i start staten, hvis vi læser et `1`, så vil vi have ikke kun to, men tre kopier! En i q_1, en i q_2 og en i q_3. Lad os så forblive på idéen med at vi læser et `0`. Så er det jo klart hvad der sker for de maskiner der er i hhv. q_1 og q_2, men hvad med den i q_3, da der jo ingen pil med et `0` er? Den bliver destrueret! Vi er fuldstændig ligeglade med den fra nu af, da den ikke eksisterer længere. Det vil sige at vi kun gider at kigge på q_1 og q_3.

Generelt set, gælder det for en NFA at, givet et symbol, kan det have nul, en, eller mange pile der kommer fra en state.

En vigtigt ting med NFA'ere, er at det er en *generalisering* af en DFA. Det vil sige, at den kan ikke noget som en DFA ikke kan. Det virker måske sådan lige nu, men vi vil senere se en måde hvorpå en NFA kan konverteres til en DFA med nogle simple udregninger (dog ikke så simple, at de tager kort tid.)

Hvis vi kigger på den formelle, matematiske, definition af en NFA vil vi se at forskellen ligger i *transitionsfunktionen*. Den formelle definition beskrives nu.

Definition 1.4

En ***nondeterministisk endelig automat*** er en $5-$tuple $(Q, \Sigma, \delta, q_0, F)$, hvor

1. Q er et endeligt sæt af states
2. Σ er et endeligt alfabet
3. $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ er transitionsfunktionen
4. $q_0 \in Q$ er startstaten
5. $F \subseteq Q$ er sættet af accept states

Vi kan her få svaret på nogle spørgsmål omkring transitionsfunktionen. Først læg mærke til at Σ_ε blot er $\Sigma \cup \{\varepsilon\}$, altså alfabetet plus den tomme streng. I definition for en DFA gik den til én Q, her går det til *potensmængden* (eng. power set) af alle states. Det vil sige, at hvis $Q = \{1, 2, 3\}$ så er $\mathcal{P}(Q) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$.

For et eksempel på hvordan en NFA kan beskrives formelt, kigger vi tilbage på Figur 1.2, som vi nu vil beskrive formelt.

$N_1 = (Q, \Sigma, \delta, q_1, F)$, hvor

1. $Q = \{q_1, q_2, q_3, q_4\}$
2. $\Sigma = \{0, 1\}$
3. δ vises som en tabel:

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

4. q_0 er start staten
5. $F = \{q_4\}$

Husk på at den tomme streng følges uanset input, trods det måske ikke er bemærket i tabellen.

Vi vil nu kigge på definitionen af komputering for en NFA.

Definition 1.5 (Komputering for en NFA)
Lad $N = (Q, \Sigma, \delta, q_0, F)$ være en NFA og w være en streng over alfabetet

Σ. Vi siger at N **accepterer** w hvis vi kan skrive w som $w = y_1 y_2 \cdots y_m$, hvor hvert y_i er et medlem af Σ_ε og en sekvens af states $r_0, r_1, \ldots, r_m$ eksisterer i Q med tre betingelser:

1. $r_0 = q_0$
2. $r_{i+1} \in \delta(r_i, y_{i+1}),$ for $i = 0, \ldots, m-1$
3. $r_m \in F$

Vi er kendte med alle betingelser fra Definition 1.2. Dog er det vigtigt at bemærke at i betingelse 2 her gælder det at $\delta(r_i, y_{i+1})$ er **sættet** af tilladte næste states, r_{i+1} er blot én af disse.

1.3 Ækvivalens af NFA'er og DFA'er

Der er tidligere beskrevet at en NFA blot er en generalisering af en DFA. Da dette gælder, må det betyde at det er muligt at konvertere en NFA til en DFA. Måden vi vil bevise på, at NFA'er og DFA'er er ækvivalente er lige præcis ved dette: konvertering af en NFA til en DFA. Bemærk at en DFA automatisk også er en NFA, og derfor er der ingen konvertering nødvendig.

Teorem 1.6
Hver nondeterministisk endelig automat har en ækvivalent deterministisk endelig automat.

Vi siger at to maskiner er **ækvivalente** hvis de genkender samme sprog.

Bevis:
Lad $N = (Q, \Sigma, \delta, q_0, F)$ være en NFA der genkender et sprog A. Vi konstruerer DFA $M = (Q', \Sigma', \delta', q_0', F')$ til at genkende A. Først kigger vi på tilfældet hvor der ingen ε pile er tilstæde, for at simplificere processen. Efter dette udvider vi så ε pile også tages i betragtning.

1. $Q' = \mathcal{P}(Q)$.
   Husk fra den formelle definition for en NFA at den går til potensmængden. Derfor er det vigtigt at kunne have alle disse muligheder med i DFA'en, selv hvis de ikke bliver brugt.

2. For $R \in Q'$ og $a \in \Sigma$, lad $\delta'(R, a) = \{q \in Q | q \in \delta(r, a)$ for en $r \in R\}$.

Dette kan også skrives således:

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q |\ R \text{ indeholder en accept state af } N\}$

For at tage ε pile i betragtning introducerer vi $E(R)$, som værende samlingen af states der kan nåes fra medlemmer af R kun ved brug af ε pile. Altså, for $R \subseteq Q$ lad

$$E(R) = \{q |\ q \text{ kan nåes fra } R \text{ ved at rejse langs 0 eller flere } \varepsilon \text{ pile}\}$$

Med denne nye samling af sæt, skal vi erstatte både startstaten og transitionsfunktionen. Startstaten er simpel, da vi bare ændrer q_0' til $E(q_0)$. I transitionsfunktionen erstatter vi $\delta(R, a)$ med $E(\delta(r, a))$, således: $\delta'(R, a) = \{q \in Q | q \in E(\delta(r, a)) \text{ for en } r \in R\}$ □

1.4 Regulære Operationer

Frem for at have dette i DFA sektionen som der bliver gjort i Sipser, mener jeg at det giver mere mening at have den til sidst, hvor man kender til alle koncepterne. De regulære operationer er operationer der kan bruges på sprog, som resulterer i et nyt regulært sprog. Vi vil kigge på følgende regulære operationer:

Definition 1.7
Lad A og B være sprog. Vi definerer de regulære operationer **fællesmængde** (union), **sammenkædning** (concatenation) og **stjerne** (star, kleene star), som følger:

1. **Fællesmængde**: $A \cup B = \{x |\ x \in A \text{ eller } x \in B\}$.
2. **Sammenkædning**: $A \circ B = \{xy |\ x \in A \text{ og } y \in B\}$.
3. **Stjerne**: $A^* = \{x_1 x_2 \dots x_k |\ k \geq 0 \text{ og hvert } x_i \in A\}$.

Vi kigger på et eksempel for at demonstrere bruget af alle operationer.

Eksempel 1.8
Lad alfabetet Σ være det danske alfabet $\{\mathtt{a}, \mathtt{b}, \dots, \mathtt{å}\}$. Hvis $A = \{\mathtt{god}, \mathtt{dårlig}\}$ og $B = \{\mathtt{bog}, \mathtt{video}\}$, så:

$$A \cup B = \{\mathtt{god}, \mathtt{dårlig}, \mathtt{bog}, \mathtt{video}\}$$
$$A \circ B = \{\mathtt{godvideo}, \mathtt{godbog}, \mathtt{dårligvideo}, \mathtt{dårligbog}\}$$
$$A^* = \{\varepsilon, \mathtt{god}, \mathtt{dårlig}, \mathtt{godgod}, \mathtt{dårligdårlig}, \mathtt{goddårlig}, \mathtt{dårliggod}, \mathtt{godgodgod}, \mathtt{godgoddårlig}, \mathtt{goddårliggod}, \dots\}$$

Tidligere blev det nævnt at, givet to regulære sprog (eller et, i tilfældet af stjerne-operationen), så vil de producere et nyt, regulært sprog.

Dette vil vi gerne for hver af disse operationer, først kigger vi på den eneste vi kan bevise med DFA'er, og efter går vi videre til dem som lettere kan bevises med NFA'er. Vi kalder det lukket, når en operation der tager to argumenter af samme klasse, giver et output i samme klasse. Eksempelvis er multiplikation og addition lukket under positive heltal, men hverken subtraktion eller division er[1].

Teorem 1.9
Klassen af regulære sprog er lukket under fællesmængde operationen.

Vi beviser dette ved at konsturere en DFA der holder styr på alle de mulige states den kan være i givet begge sprog.

Bevis:
Lad M_1 genkende A_1, hvor $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ og M_2 genkende A_2, hvor $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Vi konsturerer en ny DFA, M som genkender $A_1 \cup A_2$, hvor $M = (Q, \Sigma, \delta, q_0, F)$, hvor:

1. $Q = \{(r_1, r_2 |\ r_1 \in Q_1 \text{ og } r_2 \in Q_2)\}$.
   Dette sæt er det ***Kartesiske Produkt*** af sættene Q_1 og Q_2 og er skrevet $Q_1 \times Q_2$. Det er altså sættet af alle par af states hvor den første af fra Q_1 og den anden fra Q_2.

2. Σ alfabetet, er det samme som i både M_1 og M_2. Hvis det ikke er ville $\Sigma = \Sigma_1 \cup \Sigma_2$.

3. δ er transitionsfunktionen som defineres som følgende. For hvert $(r_1, r_2) \in Q$ og hvert $a \in \Sigma$, lad

$$\delta\left((r_1, r_2), a\right) = (\delta_1(r_1, a), \delta_2(r_2, a))$$

[1]Eksempelvis giver $1 - 100$ et negativt tal, og $1/2$ et rationelt tal.

Altså er den næste state parret af de næste states for hver af funktionerne.

4. q_0 er parret (q_1, q_2)

5. F er sættet af par hvori hvert medlem enten er en accept state af M_1 eller M_2:

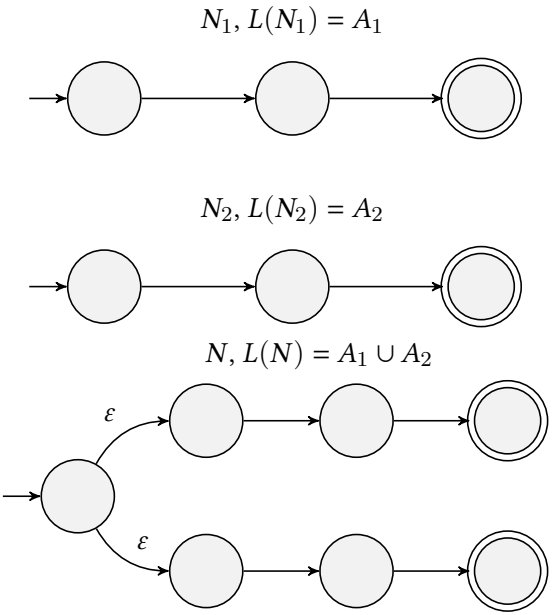
$$F = \{(r_1, r_2) | r_1 \in F_1 \text{ eller } r_2 \in F_2\}$$

Dette er det samme som $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$, men **ikke** det samme som $F_1 \times F_2$ som ville give snittet (intersection) af to sprog.

□

Vi vil nu se på et bevis på at fællesmængden er lukket under regulære sprog, hvor vi beviser det ved hjælp fra NFA'er.

Bevis:
Hvis vi har to sprog, A_1 og A_2, som hver har en NFA der genkender deres sprog, vil vi konstruere en ny NFA, N, som genkender fællesmængden af disse to sprog. En simpel måde at gøre dette på er ved at introducere en ny start state, som har en epsilonpil pegende til hver af de to NFA'ers gamle start states. Med dette har vi en NFA der genkender fællesmængden af de to sprog. En visual idé af dette kan ses i Figur 1.3.



Figur 1.3: NFA N der genkender $A_1 \cup A_2$

Lad os nu få det beskrevet formelt. Lad $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ genkende A_1 og lad $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ genkende A_2.

Konstruér $N = (Q, \Sigma, \delta, q_0, F)$ til at genkende $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
   Vi tilføjer her den nye start state, q_0.

2. q_0

3. $F = F_1 \cup F_2$

4. For en $q \in Q$ og en $a \in \Sigma_\varepsilon$:

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ og } a = \varepsilon \\ \emptyset & q = q_0 \text{ og } a \neq \varepsilon \end{cases}$$

□

Teorem 1.10
Klassen af regulære sprog er lukket under sammenkædningsoperationen.

Et visuelt bevis er her undladt, trods at det er meget deskriptivt, fordi jeg er for doven. Det kommer på min todo liste. Hvis jeg ender med ikke at inkludere det, kan det ses på side 61 i sipser.

Bevis:
Lad $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ genkende A_1, og $N_2 = (Q_2, \Sigma, \delta_2, F_2)$ genkende A_2.

Vi konstruérer en NFA til at genkende $A_1 \circ A_2$, $N = (Q, \Sigma, \delta, q_1, F_2)$.

1. $Q = Q_1 \cup Q_2$
2. q_1 fra N_1
3. $F = F_2$ fra N_2
4. δ for alle $q \in Q$ og alle $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ og } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ og } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \text{ og } a = \varepsilon \\ \delta_2(q,a) & q \in Q_2 \end{cases}$$

□

Teorem 1.11
Klassen af regulære sprog er lukket under stjerneoperationen.

Igen undlader jeg det simple visuelle bevis af dovenskab. Denne gang kan du finde det på side 62 af Sipser.

Bevis:
Lad $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ genkende A_1. Konstruér $N = (Q, \Sigma, \delta, q_0, F)$ som genkender A_1^*.

1. $Q = \{q_0\} \cup Q_1$.
2. q_0 er en ny start state
3. $F = \{q_0\} \cup F_1$.
4. δ så for alle $q \in Q$ og alle $a \in \Sigma$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{hvis } q \in Q_1 \text{ og } q \notin F_1 \\ \delta_1(q,a) & \text{hvis } q \in F_1 \text{ og } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & \text{hvis } q \in F_1 \text{ og } a = \varepsilon \\ \{q_1\} & \text{hvis } q = q_0 \text{ og } a = \varepsilon \\ \emptyset & \text{hvis } q = q_0 \text{ og } a \neq \varepsilon \end{cases}$$

□

1.5 Pumpelemmaet og ikke-regulære sprog

Der findes mange sprog der er regulære, men det er også vigtigt at vide om et givet sprog er regulært. Et klassisk eksempel på et sprog der ikke er regulært er sproget der beskriver om paranteser er balanceret, altså $L = \{w | w$ hver (er (på et tidspunkt) efterfulgt af et $)\}$. Dette gælder strenge såsom `(((((()))))(())` og `((((((((((...))))))))))`. Bemærk at alle endelige sprog er regulære, da en endelig automat vil kunne genkende alle endelige strenge, dog kan det i mange tilfælde resultere i kæmpe maskiner.

Et hovedteorem i at finde ud af om et sprog, som for eksempel sproget beskrevet før, er regulært, er *pumpelemmaet*. Ifølge pumpelemmaet gælder det at alle regulære sprog har en specifik egenskab, og dermed, hvis et sprog ikke har denne egenskab er det ikke et regulært sprog.

Teorem 1.12 (Pumpelemmaet)
Lad A være et regulært sprog, og p (**pumpelængden**) være et tal, hvor hvis s er en streng i A af længde mindst p, så kan s brydes ned i mindre stykker, $s = xyz$, som opfylder de følgende betingelser:

1. $(\forall i \geq 0)(xy^i z \in A)$
2. $|y| > 0$
3. $|xy| \leq p$

Betingelse 1 siger at en del af strengen, y skal kunne "pumpes" et uendeligt antal gange (hvor $y^0 = \varepsilon$.) Det vil sige at hvis strengen "abcdef" er en del af et regulært sprog, og $y = cde$, så vil "abf" også være en del af sproget, ligesom "abcdecdecdecdef". **Betingelse 2** siger at $y \neq \varepsilon$, altså må y ikke være en tom streng. Et enkelt symbol er nok. **Betingelse**

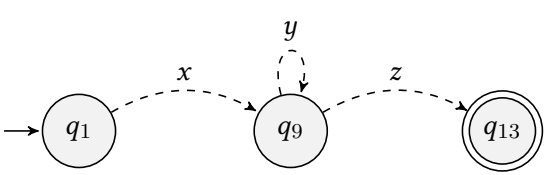
3 siger at første del af strengen, uden z-delen, skal være mindre end eller lig med p (lig med hvis $z = \varepsilon$.)

Læg mærke til i betingelserne at det er tilladt for enten x eller z at være lig ε, men ikke y, da denne skal kunne "pumpes".

Bevis:
Lad $M = (Q, \Sigma, \delta, q_1, F)$ være en DFA som genkender sproget A. Lad *pumpelængden* p være antallet af states i M, altså $|Q|$.

Lad $s = s_1 s_2 \cdots s_n$ være en streng i A af længde n, hvor $n \geq p$. Lad $r_1, \ldots, r_{n+1}$ være sekvensen af states som M går i når den gennemgår s. Dermed er $r_{i+1} = \delta(r_i, s_i)$ for $1 \leq i \leq n$. Siden sekvensen har længde $n + 1$ må mindst to states være ens.[2] Vi kalder den første r_j og den anden r_l. Fordi r_l. Da $n + 1 \geq p + 1$, gennemgåes r_l indenfor de første $p + 1$ states, i en sekvens der starter ved r_1, har vi at $l \leq p + 1$. Lad nu $x = s_1 \cdots s_{j-1}, y = s_j \cdots s_{l-1}$ og $z = s_l \cdots s_n$. Altså har vi splittet strengen op i sekvenser af states.

Da s_j og s_l er ens, betyder det at y kan gentages et uendeligt antal gange, og opfylder dermed betingelse 1. Siden $j \neq l$ er betingelse 2 også opfyldt, og $l \leq p + 1$ betyder at betingelse 3 også er opfyldt. □



Figur 1.4: Hvordan en endelig automat kan deles op ud fra beviset.

Figur 1.4 viser en simpel version af hvordan x, y og z defineres gra-

[2] Af dueslagsprincippet

fisk ud fra M. Altså har du første del af strengen, som ikke er den der kan gentages. Denne del, x bliver genkendt indtil vi kommer til staten der genkender starten af y, som kan gentages flere gange. Til sidst kommer vi til de states der genkender z. Læg mærke til at navnene på statesne er arbitrære, og er valgt for at vise et eksempel på den mulige længde mellem states.

Vi vil nu vise to eksempler til hvordan pumpelemmaet kan bruges til at bevise nonregulæritet af sprog.

Eksempel 1.13 (Sipser Eksempel 1.76)
Vi vil gerne vise at sproget $D = \{1^{n^2} | \ n \geq 0\}$ ikke er regulært. D er sproget af strenge med 1-taller af længde n^2. Vi beviser nonregulæritet ved bevis ved modstrid.
Lad $s = 1^{p^2}$. Fordi s er et medlem af D og har længde mindst p, så siger pumpelemmaet, at vi kan splitte det op så $s = xyz$, og for alle $i \geq 0$ er xy^iz også en del af sproget. For at bevise det, lad os kigge på to strenge: xyz og xy^2z. Forskellen på disse to strenge er et enkelt y. Fra betingelse 3 gælder det at $|xy| \leq p$, og dermed $|y| \leq p$ (da $|xy| \leq |y|$.) Vi ved at $xyz = p^2$, og dermed må $xy^2z \leq p^2 + p$ men vi vil gerne have $(p + 1)^2 = p^2 + 2p + 1$. Fordi betingelse 2 siger at $p \neq \varepsilon$ gælder det at $|xy^2z| > p^2$, og derfor er xy^2z imellem to andengradspotenser, og dermed er $xy^2z \notin D$ og D er dermed ikke regulært.

Eksempel 1.14 (Sipser Eksempel 1.77)
Vi bruger pumpelemmaet til at vise at sproget $E = \{0^i1^j | \ i > j\}$ ikke er regulært. Vi beviser ved modstrid.
Antag at E er regulært. Lad p være pumpelængden af E. Lad $s = 0^{p+1}1^p$. s kan blive opdelt i xyz. Hvis $y = 0$, så af betingelse 1 må xy^0z også gælde. Men da $xy^0z = xz$ er der én mindre nul, og dermed lige mange 0'er og 1'er. Dette er en modstrid, så E er **ikke** et regulært sprog.

1.6 Hovedpointer

Disse hovedpointer er taget fra ugeseddel 2, og indeholder de vigtigste ting om endelige automater.
Automater er matematiske modeller af computere.

- Automater er teoretiske modeller, brugt til at forstå principperne af komputering.
- Der er simple automater, såsom endelige automater (DFA, NFA, RE), men der er også mere komplicerede automater såsom Turing Maskiner, som kan simulere alle typer komputering.
- Automatteori hjælper med at udforske komputerings kapaciteter og begræsninger gennem flere sammenhæng.

Endelige Automater (FA) bruger en konstant mængde hukommelse og bestemmer om en given streng er en del af et sprog. Bevægelserne af en DFA er bestemt udelukkende af dens start state og input streng. En DFA M accepterer en streng w hvis den (unikke!) *walk*[3] som starter i start staten, og "staver" w slutter i en accept state af M.

- Endelige Automater er designet til at bestemme om en streng tilhører et specifikt sprog.
- FA'en finder ud af dette ved at processere strengen symbol-by-symbol, ved at følge reglerne specificeret i transitionsfunktionen.
- Ved hvert nyt symbol, ved DFA'en **altid** hvad det næste state skal være.

[3] Graf-teoretisk for en måde at gå fra A til B

- Hvis strengen w, $|w| = n$ går igennem en sekvens af states $q_0 \cdots q_n$ og $q_n \in F$, så er strengen en del af sproget som DFA'en accepterer. Ellers er den ikke.

En NFA kan vælge imellem flere alternative bevægelser, og har muligheden for at gætte bevægelserne der går til en ønsket state. En NFA M accepterer en string w hvis der eksisterer en walk fra start staten til en af de accepterende states, som "skriver" præcis w

- En NFA har, ikke ligesom en DFA, flere mulige næste states for et givent symbol, inklusiv ingen states.
- Denne nondeterminisme tillader NFA'en at udforske flere veje igennem automaten på samme tid, som om den klonede sig selv til at følge alle mulige transitions til hvert skridt.
- Hvis nogen af disse kopier, eller veje, ender i en accept state, er strengen accepteret.

Dog har NFA'er ikke en større komputationel kraft en DFA'er, da vi kan konvertere en arbitrær NFA M til en ækvivalent DFA M' (hvor $L(M') = L(M)$). Bemærk at denne konversion *ikke* er en polynomiel algoritme, siden M' kan have eksponentielt mange states i forhold til M!

- En NFA er blot en generalisering af en DFA, og kan derfor konverteres til en DFA med præcis samme deskriptive kraft (altså kan den forstå de samme sprog) som den givne NFA.
- Algoritmen der konverterer en NFA til en DFA er ikke polynomiel, da der kan være eksponentielt mange states i den konverterede DFA.

Et regulært udtryk (RE) er en formel specifikation af et regulært sprog.

- Et regulært udtryk er en sekvens af karakterer der definerer et søgemønster.
- De kan kun genkende mønstre der er en del af regulære udtryk

Et sprog er accepteret af en endelig automat hvis og kun hvis den er genereret af et regulært udtryk, siden en ækvivalent endelig automat kan blive konstrueret fra et givent regulært udtryk og omvendt.

- Ligesom ved en NFA, kan et regulært udtryk blive konvereteret.
- Dermed har et regulært udtryk samme deskriptive kraft som endelige automater.

Klassen af regulære sprog er lukket under sammenkædning, komplement, fællesmængde, stjerne, og snit. Følgende holder:

Lad L, L' være regulære sprog, så er følgende også regulære:

1. $L \cup L'$.
2. Komplementet $\overline{L}$ af L.
3. $L \cap L'$
4. $L \setminus L'$.
5. LL'.
6. L^*

For at forstå at $L \cap L'$ er regulært, er det nok at bemærke at $L \cap L' = \overline{\overline{L} \cup \overline{L'}}$

- Jeg har ikke så meget at tilføje her. Den vigtigste er nok snittet:

- Hvis vi har $\overline{L} \cup \overline{L'}$ får vi sproget der indeholder hvad der er ens mellem komplementet af de to sprog. Dette kan også beskrives som alt andet, end hvad der er ens for de to sprog (ikke komplementet!) Dermed, hvis vi tager komplementet på dette, får vi hvad der er ens for begge sprog, altså snittet.

Hvert endeligt sprog er regulært. Vi kan sagtens lave en NFA $M(w)$ som accepterer præcis strengen w, så hvis L består af strengene $w_1, w_2, \ldots, w_k$ for et endeligt heltal k, så lav NFA'erne $M(w_i), i = 1, 2, \ldots, k$ og byg (i $k - 1$ fællesmængde skridt) en NFA hvis sprog er præcis L. Så hvert non-regulært sprog indeholder abritrært lange strenge.

- Alle endelige sprog er regulære sprog.
- Vi kan se dette som at et sprog er en liste af (endelige) strenge.
- Hvis vi laver en NFA for hver af disse strenge, skal vi *bare* lave fællesmængde-NFA'en der genkender dem allesammen tilsammen.
- Dermed har vi en NFA der genkender et arbitrært endeligt sprog.

At anvende pumpelemmaet er som en 2-personersspil mellem dig og en modstander: For at bevise at en uendeligt sprog L *ikke* er regulært, gør du som følger:

- **Antag (for modstrid) at $L = L(M)$ for en DFA M, og lad p være antallet af states i M. Du kan også tænkte på det som at få p fra modstanderen, som siger at M eksisterer.**
- ***Du* bestemmer en streng $s \in L$ hvor $|s| \geq p$**
- **Nu må modstandered vælger strenge x, y, z over Σ således at**

1. $s = xyz$
2. $xy^i z \in L$ for alle $i \geq 0$
3. $|y| > 0$ og $|xy| \leq p$

Ifølge pumpelemmaet kan de gøre det kun hvis L er regulært.

- **Så vælger *du* en $i \geq 0$ og viser at $xy^i z \notin L$, som er en modstrid**
- **Siden modstriden kom fra antagelsen at L var regulært, følger det at L *ikke* er et regulært sprog.**

1.7 Opgaver

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 e 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 on 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 in (b) to find a solution.

Givet at ulven og geden ikke må være alene (ulven spiser geden), og geden og kålen må ikke være alene (da geden spiser kålen), er der en mulighed tilbage ($\binom{3}{2} = 3$, og vi udelukker 2.) Derfor skal vi først tage geden, da ulven og kålen ingen virkning har på hinanden. Efter dette skal vi tage tilbage, tage kålen, og så skal vi have geden med tilbage, så den ikke spiser kålen. Når vi er tilbage smider vi geden af, og tager ulven med så den er med kålen. Denne gang tager vi ikke noget tilbage, da ulven og kålen er gode venner, så vi henter geden og kommer tilbage til begge to uden der er sket noget.

(b) Consider strings over the alphabet $\Sigma = \{m, w, g, c\}$ and interpret these as follows: The symbol m means that the man crosses the river alone, w means that he brings the wolf, etc.

Design a finite automaton which accepts precisely those strings over Σ which correspond to a transportation sequence where everybody survives and is legal in the sense that the man can only bring an item (e.g. w) back across the river if it was actually on the shore where the boat just left from. For example, gmcg is a legal string (it is not a solution) whereas gc is not legal.

1.7.1 Sipser 1.9

Use the construction in the proof of Theorem 1.47 to give the state diagrams of NFAs recognizing the concatenation of the languages described in

a. **Exercises 1.6g and 1.6i**

1.6g

$$\{w \mid \text{ the length of } w \text{ is at most } 5\}$$

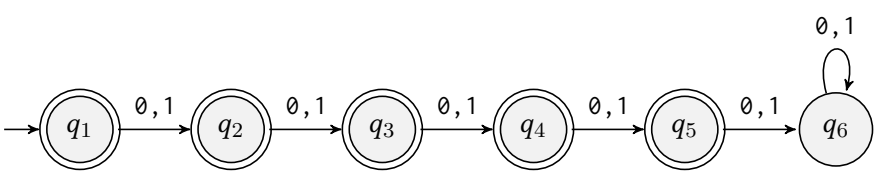
1.6i

$$\{w \mid \text{ every odd position of } w \text{ is a } 1\}$$

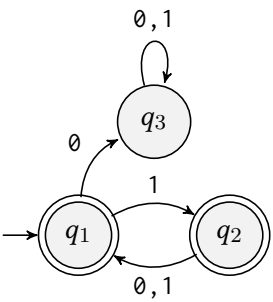
Alfabetet i begge sprog er $\{0, 1\}$

Teorem 1.47 er i disse noter Teorem 1.10. Først laver vi et state diagram af begge NFA'er, og derefter laver vi sammenkædning.

$$M_1 = \{w \mid \text{ the length of } w \text{ is at most } 5\}$$



$M_2 = \{w \mid \text{ every odd position of } w \text{ is a } 1\}$



Lad os nu beskrive begge disse maskiner formelt.
$M_1 = (Q, \Sigma, \delta, q_1, F)$ hvor

1. $Q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$
2. $\Sigma = \{0, 1\}$
3. $\delta =$

	0	1
q_1	q_2	q_2
q_2	q_3	q_3
q_3	q_4	q_4
q_4	q_5	q_5
q_5	q_6	q_6
q_6	q_6	q_6

4. q_1

5. $F = \{q_1, q_2, q_3, q_4, q_5\}$

$M_2 = (Q, \Sigma, \delta, q_1, F)$, hvor

1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. $\delta =$

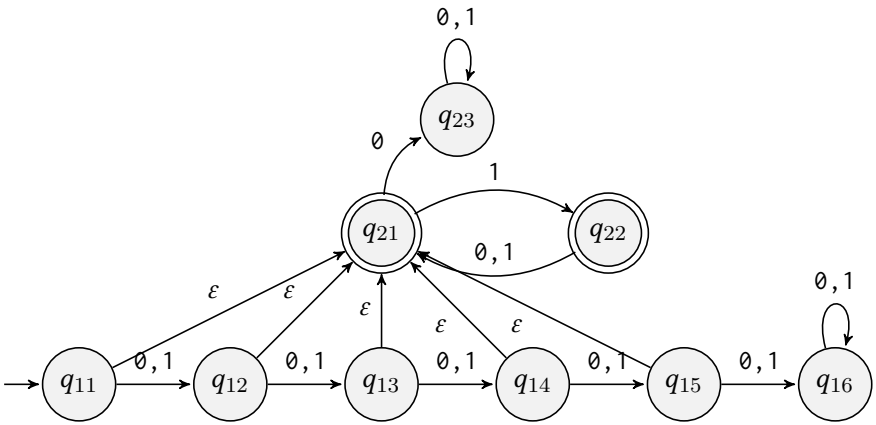
	0	1
q_1	q_3	q_2
q_2	q_1	q_1
q_3	q_3	q_3

4. q_1
5. $F = \{q_1, q_2\}$

Vi bruger nu Teorem 1.10 til at lave sammenkædningen. Vi kalder vores nye maskine M_3. $M_3 = (Q, \Sigma, \delta, q_0, F)$, hvor:

1. $Q = \{q_{11}, q_{12}, q_{13}, q_{14}, q_{15}, q_{16}\} \cup \{q_{21}, q_{22}, q_{23}\}$
2. $\Sigma = \{0, 1\}$
3. δ undlades
4. q_{11}
5. $F = \{q_{21}, q_{22}\}$

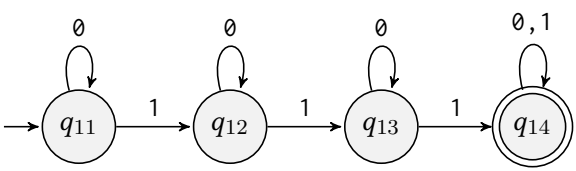
Vi laver det om til et state diagram:



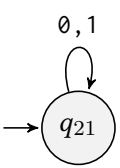
b. **Exercises 1.6b and 1.6m**

Nu når vi kender rutinen bliver denne langt mindre formel:

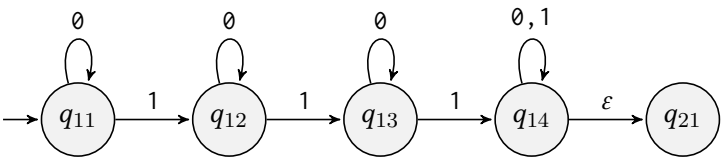
$$M_1 = \{w \mid w \text{ contains at least three } 1s\}$$



$M_2 =$ the empty set



Den var da kedelig, og sammenkædningen bliver endnu kedeligere!



1.7.2 Sipser 1.11

Prove that every NFA can be converted to an equivalent one that has a single accept state.

Det her spørgsmål er heldigvis meget simpelt, og jeg gider ikke bruge mere tid på at tegne diagrammer i TikZ. Måden du konverterer er ved at lave en ny accept state. Så skal du tage alle gamle accept states og konvertere dem til ikke accept states. Derefter laver du ε pile fra alle de gamle accept states til den nye accept state. Der gør bare NFA'en mere kompleks, men hey! Det virker!

1.7.3 Sipser 1.12

Let

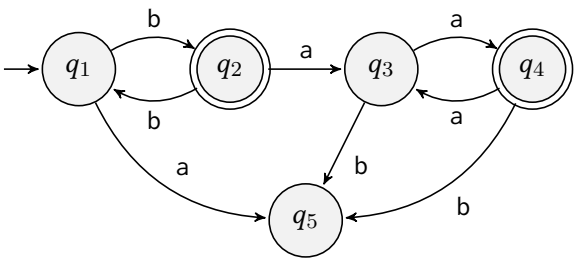
$$D = \{w \mid w \text{ contains an even number of } a\text{'s and an odd number of } b\text{'s and does not contain substring } ab\}.$$

Give a DFA with five states that recognizes D and a regular expression that generates D. (Suggestion: Describe D more simply)

Først simplificerer vi. Siden ab er ulovligt, kan vi ændre simplificere og sige at alle a'er må komme efter b'er. Vi starter med det regulære udtryk.

$$b(bb) * (aa)*$$

Og her kommer state diagrammet:



1.7.4 Sipser 1.31

For any string $w = w_1 w_2 \cdots w_n$, the *reverse* of w, written $w^{\mathcal{R}}$, is the string w in reverse order, $w_n \cdots w_2 w_1$. For any language A, let $A^{\mathcal{R}} = \{w^{\mathcal{R}} | w \in A\}$. Show that if A is regular, so is $A^{\mathcal{R}}$

Alle regulære sprog kan defineres af en FA. Derfor antager vi at det omvendte sprog også kan defineres af en FA. Givet en NFA $M = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ definerer vi en reverse NFA således $M_{\mathcal{R}} = (Q, \Sigma, \delta, q_0, F)$ hvor

1. $Q = Q_1 \cup \{q_0\}$
2. $\Sigma = \Sigma_1$
3. q_0 er en nyintroduceret state
4. $F = \{q_1\}$

5.

$$\delta = \begin{cases} F_1 & q = q_0 \text{ og } a = \varepsilon \\ \emptyset & q = q_0 \text{ og } a \neq \varepsilon \\ \delta(q,a)^{-1} & \text{for resten} \end{cases}$$

Hvor $\delta(q,a)^{-1}$ betyder at omvendde pilen fra hvad den var tidligere, i.e., hvis $\delta_1(q_{100}, a) = q_{101}$, så $\delta(q_{101}, a) = q_{100}$, hvor δ_1 er fra den originale automat, og δ fra reverse. Dette skal fortsætte, så vi får en "omvendt" FA, og dermed reverse. QED.

1.7.5 Sipser 1.29(b)

Use the pumping lemma to show that the following languages are not regular.

$A_2 = \{www | w \in \{\mathsf{a}, \mathsf{b}\}^*\}$

Bevis:
Bevis ved modstrid.
Vi antager at A_2 er regulært. For at være regulært skal det opfylde følgende kriterier (Fra Teorem 1.12):

1. $(\forall i \geq 0)(xy^i z \in A)$
2. $|y| > 0$
3. $|xy| \leq p$

Vi tager strengen $a^p b a^p b a^p b$, hvor $w = a^p b$. Lad $\mathsf{aabaabaab} \in A_2$. Pumpelængden er 2. $x = a, z = a, y = baabaab$. Dermed har vi $(a)(a)(baabaab)$. Hvis vi pumper midterdelen med $i = 2$ får vi $(a)(aa)(baabaab)$ hvilket ikke er en del af sproget. Dermed er sproget ikke regulært. □

1.7.6 Sipser 1.30

Describe the error in the following "proof" that 0^*1^* is not a regular language. (An error must exist because 0^*1^* *is* regular.) The proof is by contradiction. Assume that 0^*1^* is regular. Let p be the pumping length for 0^*1^* given by the pumping lemma. Choose s to be the string 0^p1^p. You know that s is a member of 0^*1^*, but Example 1.73 shows that s cannot be pumped. Thus you have a contradiction. So 0^*1^* is not regular.

Eksempel 1.73 gælder kun fordi antallet af `0` og 1'ere skal være lige. Dette er ikke eksemplet her. For eksempel er strengen `000000000` en del af sproget beskrevet her, men det er den ikke i eksemplet. Dog er resten af beviset i eksempel 1.73 god nok med undtagelse af betingelse 3. Grunden til at betingelse 3 ikke fungerer, i eksempel 1.73 er fordi det ville forstyrre ligheden mellem 0 og 1. Dette er jo ikke nødvendigt her, og derfor gælder eksemplet ikke.

1.7.7 Sipser 1.36

Let $B_n = \{a^k | k$ is a multiple of $n\}$. Show that for each $n \geq 1$, the language B_n is regular.

Regulært udtryk:

$$\mathsf{a}^{\mathsf{n}*4}$$

Siden det beskrives af at regulært udtryk er det regulært.

1.7.8 Sipser 1.43

Let A be any language. Define *DROP-OUT(A)* to be the language containing all strings that can be obtained by removing one symbol from a string in A. Thus, *DROP-OUT(A)* = $\{xz | xyz \in A$ where $x, z \in \Sigma^*, y \in \Sigma\}$.

[4] Bemærk at det er $(a^n)^*$

Show that the class of regular languages is closed under *DROP-OUT* operation. Give both a proof by picture and a more formal proof by construction as in Theorem 1.47.

1.7.9 Sipser 1.49

a. Let $B = \{1^k y | y \in \{0,1\}^* \text{ and } y \text{ contains at least } k \text{ 1s, for } k \geq 1\}$. Show that B is a regular language.

Bevis af regulært udtryk:

$$1(0 \cup 1)^* 1(0 \cup 1)^*$$

b. Let $C = \{1^k y | y \in \{0,1\}^* \text{ and } y \text{ contains at most } k \text{ 1s, for } k \geq 1\}$. Show that C isn't a regular language.

Forskellen her er at det er **højest** k 1'ere, og ikke mindst. Vi beviser med modstrid v.h.a. pumpelemmaet.

1.7.10 Sipser 1.41

For languages A and B, let the *perfect shuffle* of A and B be the language

$$\{w \mid w = a_1 b_1 \cdots a_k b_k, \text{ where } a_1 \cdots a_k \in A \text{ and } b_1 \cdots b_k \in B, \text{ each } a_i, b_i \in \Sigma\}$$

Show that the class of regular languages is closed under shuffle.

Til at gøre dette laver vi simpelt en NFA hvis start state er start staten af a, og accept er accept staten i b. Derudover er transitionsfunktionen en funktion der først går fra a_1 til b_1, så til a_2 til b_2 etc. Siden vi kan lave en sådan automat er perfect shuffle regulært.

1.7.11 Sipser 1.38

An all-NFA M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ that accepts $x \in \Sigma^*$ if *every* possible state that M could be in after reading input x is a state from F. Note, in contrast, that an ordinary NFA accepts a string if *some* state among these possible states is an accept state. Prove that all-NFAs recognize the class of regular languages.

En DFA er en NFA, men med nogle ekstra begrænsninger, herunder at den kun kan være i én accept state af gangen. En NFA er en generaliseret DFA. Dermed bevist.

1.7.12 Sipser 1.54

onsider the language $F = \{a^i b^j c^k \mid i, j, k \geq 0$ and if $i = 1$ then $j = k\}$.

a. Show that F is not regular.

I've no clue

b. Show that F acts like a regular language in the pumping lemma. In other words, give a pumping length p and demonstrate that F satisfies the three conditions of the pumping lemma for this value of p.

c. Explain why parts (a) and (b) do not contradict the pumping lemma.

2

Kontekstfrie Sprog

I det tidligere kapitel om endelige automater, så vi den klasse af sprog som de genkender, regulære sprog, men vi så også eksempler på sprog som ikke er regulære, og hvordan dette kan bevises. I dette kapitel vil vi introducere *kontekstfrie grammatikker* som er stærkere end endelige automater i deres deskriptionskraft. Sprogene der genkendes af kontekstfrie grammatikker kaldes *kontekstfrie sprog*. Ydermere introducerer vi *stakautomater* (pushdown automata på engelsk) som er en klasse af maskiner der genkender kontekstfrie sprog.

Et sprog kaldes *kontekstfri*, fordi dets udledning[1] *ikke* afhænger af dens kontekst, e.g., hvis du erstatter A i uAv så er det ikke afhængigt af u eller v.

[1] Definition for udledning er i starten af Sektion 2.1

2.1 Kontekstfrie Grammatikker

En grammatik består af en samling af **substitueringsregler**, også kaldet *produktioner*. Hver regel forekommer som en linje i grammatikken, og består af et symbol efterfulgt af en pil efterfulgt af en streng. Følgende grammatik er et eksempel på en kontekstfri grammatik G_1:

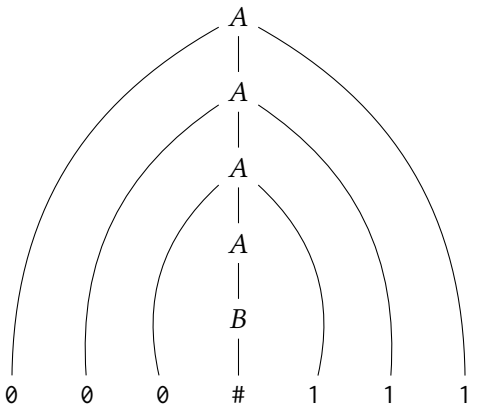
$$\begin{aligned} A &\to \texttt{0}A\texttt{1} \\ A &\to B \\ B &\to \# \end{aligned} \tag{G_1}$$

Symbolet på venstresiden kaldes en *variabel*[2]. Strengen (højresiden) består af variabler og andre symboler, kaldet *terminale*. Terminale minder om alfabetet fra en endelig automat, da disse er de endelige symboler, der udgør en resulterende streng[3]. Variabler er oftest repræsenteret som store bogstaver, hvor terminale kan være små bogstaver, tal, eller specielle symboler (såsom #.) En af variablerne (venstrehåndssiden) bliver udpeget til at være **startvariablen**. Denne variabel skal alle udledninger af grammatikken starte på (analogt til en start state i en endelig automat.)

For at beskrive et sprog med en grammatik, *udleder* vi strenge fra grammatikken. Dette bliver gjort systematisk ved at *substituere* variabler med andre variabler eller terminale. Eksempeltvis kan gramatikken G_1 udlede strengen `000#111`. En udledning af denne streng i G_1 er følgende: $A \Rightarrow \texttt{0}A\texttt{1} \Rightarrow \texttt{00}A\texttt{11} \Rightarrow \texttt{000}A\texttt{111} \Rightarrow \texttt{000}B\texttt{111} \Rightarrow \texttt{000}\#\texttt{111}$. En måde hvorpå du kan vise dette grafisk er *parse-træer*. Et eksempel kan ses i Figur 2.1, som viser et parsetræ for strengen `000#111` i grammatikken G_1.

[2] Kært barn har mange navne, jeg har også hørt non-terminal og, simpelt, LHS.

[3] Dette vil vi komme ind på i mere detalje senere.



Figur 2.1: Parse-træ for `000#111` i G_1

Alle strenge du kan udlede på denne måde er en del af sproget, og udgør tilsammen det kontekstfri sprog som G_1 genkender. Hvis du ikke har lagt mærke til det nu, er denne meget simple kontekstfri grammatik faktisk et eksempel på et sprog der **ikke** kan genkendes af en endelig automat. Ofte, fremfor at skrive $A \to \texttt{0}A\texttt{1}$ og så på en ny linje skrive $A \to B$, så skriver man blot $A \to \texttt{0}A\texttt{1}|B$, hvor du kan se | som "eller".

Vi vil nu kigge på den formelle definition på en kontekstfri grammatik.

Definition 2.1 (Formel Definition på en Kontekstfri Grammatik)
En **kontekstfri grammatik** er en 4-tuple (V, Σ, R, S), hvor

1. V er et endeligt sæt kaldet **variabler**
2. Σ er et endeligt sæt, disjunkt fra V, kaldet **terminaler**
3. R er et endeligt sæt af **regler**, hvor hver regel er en streng af

variabler og terminaler og en variabel

4. $S \in V$ er startvariablen.

Givet u, v og w er strenge af variabler, og terminale, og $A \rightarrow w$ er en regel af grammatikken, så siger vi at uAv giver (*yields*) uwv, skrevet $uAv \Rightarrow uwv$. Vi siger at u udleder v, skrevet, $u \overset{*}{\Rightarrow} v$, hvis $u = v$, eller hvis en sekvens $u_1, u_2, \ldots, u_k$ eksisterer når $k \geq 0$ og $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$. Sproget af grammatikken er $\{w \in \Sigma * | S \overset{*}{\Rightarrow} w\}$.

Altså betyder det, at hvis *u giver v*, så tager det kun ét skridt. Hvis *u udleder v* tager det mere end ét skridt.

2.1.1 Regularitet

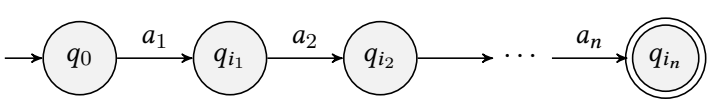
Alle regulære sprog er også kontekstfri sprog. Det vil altså sige, at kontekstfri sprog *udvider* på regulære sprog, og kan mere, end et regulært sprog kan.

Teorem 2.2
Alle regulære sprog er også kontekstfri.

Bevis:
Lad $M = (Q, \Sigma, \delta, q_0, F)$ være en DFA med $L(M) = L$, $Q = \{q_0, q_1, \ldots, q_k\}$.
Vi konsturerer en kontekstfri grammatik $G = (V, \Sigma, R, S)$ hvor,

1. $V = \{X_0, X_1, \ldots, X_k\}$
2. R konstrueres således:
   - Hvis $\delta(q_i, a) = q_j$ så $X_i \to aX_j \in R$.
   - Hvis $q_l \in F$ så $X_l \to \varepsilon \in R$.
3. X_0 er startsymbolet i G.

Antag at $w \in L, w = a_1 a_2 \cdots a_n$:



Så

$$X_0 \Rightarrow a_1 X_{i_1} \Rightarrow a_1 a_2 X_{i_2} \Rightarrow \cdots \Rightarrow a_1 a_2 \cdots a_n X_{i_n} \Rightarrow a_1 a_2 \cdots a_n = w$$

Dermed bevist at regulære sprog er et subset af kontekstfri sprog.
□

2.1.2 Tvetydighed

En grammatik der kan generere den sammen streng på to eller flere forskellige måder (og dermed have mere end ét parsetræ for én streng), kaldes *tvetydig*. For eksempel er den følgende grammatik tvetydig:

$$\langle EXPR \rangle \rightarrow \langle EXPR \rangle + \langle EXPR \rangle | \langle EXPR \rangle + \langle EXPR \rangle | (\langle EXPR \rangle) | a$$

Strengen a + a × a kan genereres tvetydigt, enten:

$$\langle EXPR \rangle \Rightarrow \langle EXPR \rangle + \langle EXPR \rangle \Rightarrow \langle EXPR \rangle + \langle EXPR \rangle \times \langle EXPR \rangle \Rightarrow \cdots \Rightarrow a + a \times a$$

eller

$$\langle EXPR \rangle \Rightarrow \langle EXPR \rangle \times \langle EXPR \rangle \Rightarrow \langle EXPR \rangle + \langle EXPR \rangle \times \langle EXPR \rangle \Rightarrow \cdots \Rightarrow a + a \times a$$

.

Definition 2.3 (Tvetydighed)
En streng w er udledt ***tvetydigt*** i en kontekstfri grammatik G hvis den har to eller flere venstremest udledninger. Grammatik G er *tvetydig* hvis den genererer en streng tvetydigt.

Ofte, hvis man har en tvetydig grammatik, er det muligt at konvertere den til en grammatik der ikke er tvetydig. Dog er dette ikke muligt for alle sprog. Sprog hvor dette **ikke** gælder kalder vi **i sig selv tvetydige** (inherently ambiguous.)

2.1.3 Chomsky Normal Form

Chomsky Normal Form er en form af en kontekstfri grammatik. Bemærk at den også kaldes "Chomsky Grammar".

Definition 2.4 (Chomsky Normal Form)
En kontekstfri grammatik er i ***Chomsky Normal Form*** hvis hver regel har formen

$$A \to BC$$
$$A \to a$$

hvor a er enhver terminal, og A, B og C er enhver variabel, dog må B og C ikke være startvariablen. Derudover tillader vi reglen $S \to \varepsilon$, hvor S er startvariablen.

Teorem 2.5
Ethvert kontekstfri sprog er genereret af en kontekstfri grammatik i Chomsky Normal Form.

Vi vil vise at vi kan konvertere enhver grammatik til chomsky normal form. Vi vil gøre dette ved at fjerne alle regler der går imod betingelserne for CNF, og erstatte med regler der overholder. Først tilføjer vi en ny startvariabel, og eliminerer alle regler af formen $A \to \varepsilon$ (kaldet ε-regler.) Vi eliminerer også alle **unit rules**, som har formen $A \to B$. Til sidst konverterer vi resten til at være i ordentlig form.

Bevis:
Skridt 1: Først tilføjer vi en ny variabel S_0 og reglen $S_0 \to S$, hvor S er det originale startvariabel.
Skridt 2: Efter dette fjerner vi alle regler af formen $A \to \varepsilon$, hvor $A \neq S$. Når dette er gjort fjerne vi alle regler hvor A er på højresiden af en

regel, og tilføjer en ny regel med A fjernet. Eksempelvis bliver $R \to uAv$ til $R \to uv$ og $R \to uAvAw$ bliver til tre regler $R \to uvAw, R \to uAvw, R \to uvw$. Hvis vi har reglen $R \to \varepsilon$ tilføjer vi $R \to \varepsilon$, **undtagen hvis vi allerede har fjernet denne regel**. Vi forstætter dette indtil alle ε-regler er væk (undtagen startvariablen.)
Skridt 3: Efter vi har gjort dette tager vi os af *unit rules*. Vi fjerner en unit rule $A \to B$. Så, når $B \to u$ forekommer, tilføjer vi $A \to u$ (bemærk at vi ikke fjerner $B \to u$, da den kan komme et andet sted fra.) Vi bliver ved indtil vi har fjernet alle unit rules.
Skridt 4: Til sidst konverterer vi resten af reglerne til den rigtige form. Vi erstatter hver regel $A \to u_1 u_2 \cdots u_k$ hvor $k \geq 3$ og hvert u_i er en variabel eller terminalt symbol, med reglerne $A \to u_1 A_1, A_1 \to u_2 A_2, A_2 \to u_3 A_3, \ldots,$ og $A_{k-2} \to u_{k-1} u_k$. A_i'erne er nye variabler. Vi erstatter en hver terminal u_i i de tidligere regler men den nye variabel U_i og tilføjer reglen $U_i \to u_i$. □

Dette kan godt være svært at forstå, og derfor kigger vi på et eksempel.

Eksempel 2.6
Vi konverterer følgende grammatik til CNF:

$$S \to ASA \mid aB$$
$$A \to B \mid S$$
$$B \to b \mid \varepsilon$$

Skridt 1:
Første skridt er at tilføje en ny startvairabel. Vi gør dette ved at introducere reglen $S_0 \to S$

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid aB \\
A &\to B \mid S \\
B &\to b \mid \varepsilon
\end{aligned}
$$

Skridt 2:
Vi fjerner ε–reglerne.

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid aB \mid a \qquad \text{da B kan blive } \varepsilon \\
A &\to B \mid S \mid \varepsilon \\
B &\to b
\end{aligned}
$$

Dog introducerer dette en ny epsilon-regel som vi må fjerne:

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid aB \mid a \mid SA \mid AS \mid S \\
A &\to B \mid S \\
B &\to b
\end{aligned}
$$

Skridt 3:
Vi fjerner nu unit-reglen $S \to S$:

$$
\begin{aligned}
S_0 &\to S \\
S &\to ASA \mid aB \mid a \mid SA \mid AS \\
A &\to B \mid S \\
B &\to b
\end{aligned}
$$

Og nu $S_0 \to S$:

$$
\begin{aligned}
S_0 &\to ASA \mid aB \mid a \mid SA \mid AS \\
A &\to B \mid S \\
B &\to b
\end{aligned}
$$

Nu fjerner vi reglen $A \to B$:

$$
\begin{aligned}
S_0 &\to ASA \mid aB \mid a \mid SA \mid AS \\
A &\to S \mid b \\
B &\to b
\end{aligned}
$$

Til sidst fjerner vi reglen $A \to S$:

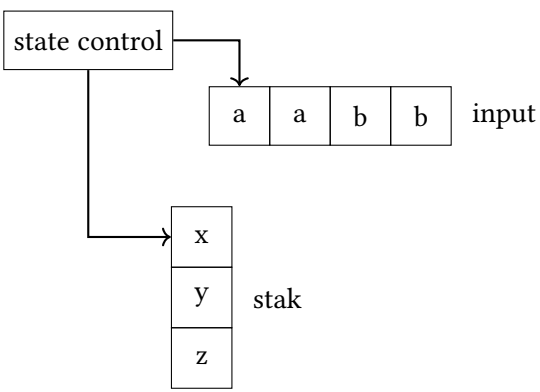
$$
\begin{aligned}
S_0 &\to ASA \mid aB \mid a \mid SA \mid AS \\
A &\to b \mid ASA \mid aB \mid a \mid SA \mid AS \\
B &\to b
\end{aligned}
$$

Skridt 4:
Nu skal vi konvertere resten.

$$
\begin{aligned}
S_0 &\to AA_1 \mid UB \mid a \mid SA \mid AS \\
S &\to AA_1 \mid UB \mid a \mid SA \mid AS \\
A &\to b \mid AA_1 \mid a \mid SA \mid AS \\
A_1 &\to SA \\
U &\to a \\
B &\to b
\end{aligned}
$$

2.2 Pushdown Automater

Pushdown Automata (PDA) er ligesom NFA'er, men med en **stak** tilføjet. Denne stak tillader at PDA'en kan genkende kontekstfrie sprog. Det vil altså sige at en PDA er ækvivalent i dens kraft med kontekstfrie grammatikker.



Figur 2.2: Skematik af en pushdown automat

I Figur 2.2 ses en skematik af en PDA med en stak, hvor "state control" er delen med alle statesne, pile, etc. "input" er inputtet til automaten, og "stak" er stakken. En PDA kan skrive symboler til stakken og så læse dem tilbage senere. Den fungerer i en last-in-first-out princip. Vi referer i noterne til "pushing" som værende at give data til stakken, og "popping" som værende at læse data fra stakken.

Eksempel 2.7
Sproget $\{0^n1^n | n \geq 0\}$ kan **ikke** genkendes af en endelig automat, men det kan den af en PDA. Følgende er en beskrivelse af hvordan automa-

ten fungerer:
Læs symbolerne fra input. Hver gang et 0 bliver læst, pushes den til stakken. Hver gang et 1 er læst, pop den fra stakken. Hvis inputet er færdiglæst så snart stakken er tom, accepteres inputtet. Ellers accepteres det ikke (f.eks. hvis der er flere 1'ere når stakken er tom, hvis der er flere 0'ere efter der er fundet en 1'er, hvis stakken stadig har 0'ere.)

Det er vigtigt at bemærke at de pushdown automater vi arbejder med er nondeterministiske, da deterministiske og nondeterministiske pushdown automater **ikke** er ævkvivalente i deres kraft.

2.2.1 Formel Definition af Pushdown Automat

Den formelle definition for en PDA minder meget om den for en FA, dog med et ekstra alfabet: Γ som er alfabetet for stakken.

Definition 2.8 (Formel Definition af en Pushdown Automat)
En *pushdown automat* er en 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, hvor Q, Σ, Γ, og F alle er endelige sæt, og

1. Q er sættet af states
2. Σ er input alfabetet
3. Γ er stak alfabetet
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow P(Q \times \Gamma_\varepsilon)$ er transitionsfunktionen
5. $q_0 \in Q$ er start staten
6. $F \subseteq Q$ er sættet af accept states.

2.2.2 Pushdown Automat Komputering

En PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ komputerer som følger. Den accepterer input w hvis w kan skrives $w = w_1 \cdots w_m$, hvor hvert $w_i \in \Sigma_\varepsilon$ og sekvensen af states $r_0, r_1 \ldots, r_m \in Q$ og strenge $s_0, s_1, \ldots, s_m \in \Gamma^*$ eksisterer som tilfredsstiller følgene betinglser, hvor s_i repreæsenterer sekvensen af stakindhold som M har på den accepterende afdeling af komputering.

1. $r_0 = q_0$ og $s_0 = \varepsilon$. Altså starter vi ved start staten og har en tom stak.

2. For $i = 0, \ldots, m-1$ har vi $(r_{i+1}, b) \in \delta(r_i, w_{i+1}, a)$, hvor $s_i = at$ og $s_{i+1} = bt$ for alle $a, b \in \Gamma_\varepsilon$ og $t \in \Gamma^*$. Altså M bevæger sig ifølge staten, stakken og inputsymbolet.

3. $r_m \in F$. Den sidste state er en accept state.

2.2.3 Eksempler

Eksempel 2.9 ($\{0^n 1^n | n \geq 0\}$)
Vi kigger på et eksempel af en formel definition af en PDA der genkender sproget $\{0^n 1^n | n \geq 0\}$. Lad M_1 være $(Q, \Sigma, \Gamma, \delta, q_1, F)$, hvor

1. $Q = \{q_1, q_2, q_3, q_4\}$.

2. $\Sigma = \{0, 1\}$

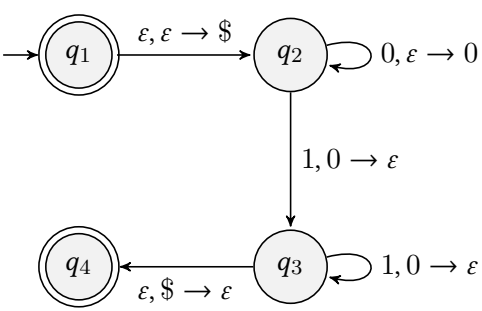
3. $\Gamma = \{0, \$\}$

4. $F = \{q_1, q_4\}$

5. δ gives af følgende tabel, hvor tomme felter er $\emptyset$:

Input	0			1			ε		
Stak:	0	$	ε	0	$	ε	0	$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

Altså betyder det at når den er i state 1, og den ser en tom stak, så pusher den $ som symboliserer enden på strengen skal være nået. Når den er ved state 2, stakken er tom (den tomme streng), og den ser et 0, så skal den forblive ved state 2, og pushe et 0. Dette bliver den ved med for alle 0'er muligt. Hvis den måder et 1 tal når der er et 0 på stakken imens den er i state 2, skifter den til state 3, og popper fra stakken. Sidst, når den er i state 3 og strengen er færdiglæst, og der kun er et $ tilbage på stakken, så er strengen accepteret.



Figur 2.3: State diagram af PDA'en $M_1, L(M_1) = \{0^n 1^n | n \geq 0\}$

Vi kan også illustrere denne PDA i form af et state diagram, ligesom

ved endelige automater. Se Figur 2.3. Her betyder notationen $a, b \to c$ at når den ser et a erstatter den b på toppen af stakken med et c.

2.2.4 Konvertering af CFG til PDA

Teorem 2.10
Hvis A er et kontekstfrit sprog, findes der en pushdown automat der genkender A.

Vi beviser ved at konvertere en kontekstfri grammatik til en pushdown automat. En grammatik genererer strenge, hvor en PDA genkender sprog.

Vi vil gerne have en PDA der begynder med startvariablen og gætter på substitutions. Den gemmer de mellemliggende genereret sterenge på stakken, og når den er færdig sammenligner den med inputtet.

Bevis:
Vi beviser ved konstruktion. Konverter CFG'en for A til den følgende PDA:

1. Push startsymbolet på stakken
2. Hvis toppen af stakken er en:
   (a) **Variabel**: erstat med højrehåndssiden af reglen (nondeterminisme tillader flere)
   (b) **Terminal**: pop den og match med næste input symbol.
3. Hvis stakken er tom, så accepter.

□

2.2.5 Ækvivalens af CFG og PDA

Teorem 2.11
A er et kontekst frit sprog hvis og kun hvis[4] en PDA genkender *A*

Lemma 2.12
Hvis et sprog er kontekstfrit, så er der en PDA som genkender det.

Lad A være et kontekstfrit sprog. Fra definitionen af et kontekstfrit sprog, at der findes en kotekstfri grammatik som genkender det. Vi vil så gerne lave en pushdown automat P ud fra en kontekstfri grammatik, G. Følgende er en uformel beskrivelse af P's processering.

1. Placer EOF symbolet \$på toppen af stakken, sammen med startvariablen.
2. Gentag de følgende skridt forevigt
    a. Hvis toppen af stakken er et variabelsymbol A, så vælg nondeterministisk alle mulige regler for A, og substituer A med strengen på højrehåndssiden af reglen.
    b. Hvis toppen af stakken er en terminal a, så læs det næste symbol og sammenlign med a. Hvis de matcher, så gentag processen, og ellers reject denne gren.
    c. Hvis toppen af stakken er \$, så gå ind i accept staten. Dette accepterer så inputtet hvis det hele er læst.

Vi vil nu komme med det formelle bevis på dette.

Bevis:
Lad $P = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, F)$.

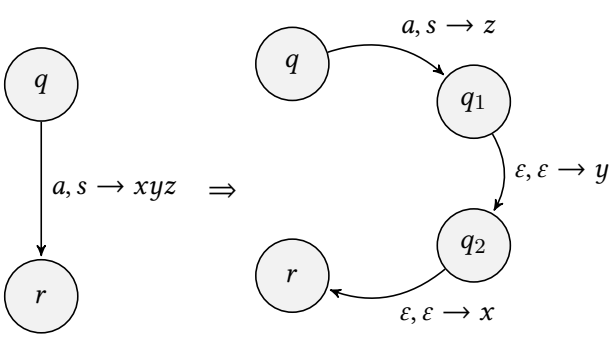
[4] Ligheden går begge veje, altså $\leftrightarrow$

Lad $q, r \in Q$, $a \in \Sigma_\varepsilon, s \in \Gamma_\varepsilon$. Vi vil gerne have at, når P går fra state q til state r, så pusher den a og popper s. Her tillader vi pushing og popping af hele strenge, trods det ikke er en mulighede i en præcis definition, vi vil nu se hvordan det kan gøres.

Vi vil gerne pushe en hel streng $u = u_1 u_2 \cdots u_l$ til stakken. Dette kan vi implementere ved at introducere de nye states $q_1 q_2 \cdots q_{l-1}$, og sætte transitionsfunktionen som følger:

$$\begin{aligned}
\delta(q, a, s) &\text{ indeholder } (q_1, u_l), \\
\delta(q_1, \varepsilon, \varepsilon) &= \{(q_2, u_{l-1})\}, \\
\delta(q_2, \varepsilon, \varepsilon) &= \{(q_3, u_{l-2})\}, \\
&\vdots \\
\delta(q_{l-1}, \varepsilon, \varepsilon) &= \{(r, u_1)\}.
\end{aligned}$$

Følgende billede klargører hvordan dette fungerer:



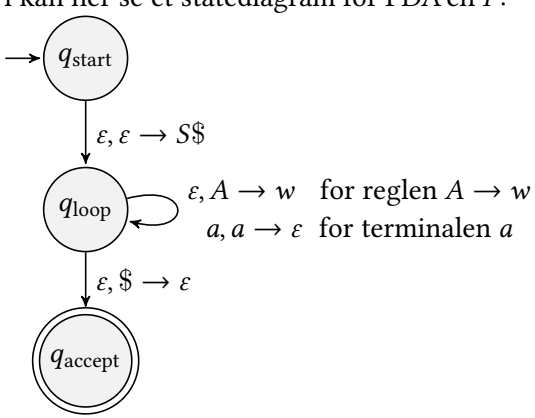
Statesne i P, $Q = \{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}\} \cup E$, hvor E er sættet af states nødvendige for at implementere forenklingen vist før. Start staten er q_{start}, og accept staten er q_{accept}.

Transitionsfunktionen defineres som følger:

Først vil vi pushe symbolet \$og S. Dermed $\delta(q_{\text{start}}, \varepsilon, \varepsilon) = \{(q_{\text{loop}}, S\$)\}$. Så putter vi transitionerne for hovedskridtet af skridt 2: Vi kigger på de forskellige tilfælde beskrevet før starten af beviset.

- Tilfældet hvor toppen af stakken er en variabel. Lad $\delta(q_{\text{loop}}, \varepsilon, A) = \{(q_{\text{loop}}, w \mid \text{ hvor } A \to w \text{ er en regel i } R)\}$
- Tilfældet hvor toppen af stakken er en terminal. Lad $\delta(q_{\text{loop}}, a, a) = \{(q_{\text{loop}}, \varepsilon)\}$.
- Tilfældet hvor toppen af stakken er \$. Lad $\delta(q_{\text{loop}}, \varepsilon, \$) = \{(q_{\text{accept}}, \varepsilon)\}$.

Vi kan her se et statediagram for PDA'en P:



QED. □

Vi vil nu gerne bevise den omvendte del af Teorem 2.11.

Lemma 2.13
Hvis en PDA genkender et sprog er det kontekstfrit.

Beviset for dette er mere kompliceret end omvendt. Vi beviser trinvist.

Bevis:
Trin 1:
Vi vil gerne modificere PDA'en til hvad der kaldes en *begrænset* PDA. En begrænset PDA opfylder følgende krav:

- Den har én accept state, q_{accept} (brug epsilon-pile fra de gamle til den nye.)
- Stakken skal altid være tømt før en streng accepteres.
- Hver transition popper eller pusher et symbol, men ikke begge.
  - For eksempel, hvis du både popper og så pusher, i stedet lav én transition der popper, og så én der pusher for at få samme effekt.
  - Det er heller ikke tilladt at lade være med at gøre noget med stakken; derfor kan man i stedet have to transitions: én der pusher et nyt symbol, og så en epsilon transition der popper samme symbol.

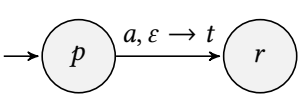
Trin 2:
Vi vil gerne forstå hvordan en PDA fungerer i mere detalje. Hvis vi har en PDA med en stak indeholdende kun slutsymbolet, \$, og der så bliver processeret en hel streng, så regner vi med at slutsymbolet er det eneste tilbage på stakken. Vi kan videreføre dette, til hvis stakken allerede har nogle elementer i sig. Hvis dette er tilfældet, og den derefter læser en streng w, så vil den have de samme elementer i sig efter, altså **der bliver ikke rørt ved den del af stakken der ikke er pushet under gennemgang af en streng**.

Trin 3:

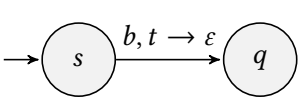
Grammatikken G. For alle mulige valg af states, $p, q \in Q(m)$, vil G indeholde variablen A_{pq}, som vil generere alle strenge $w \in \Sigma^*$ hvor det gælder at den starter på en tom stak, gennemgår strengen, og ender på en tom stak (se trin 2.) Ved disse strenge er der to muligheder. Givet at en stak starter på level l, det vil sige, at stakken har l elementer i sig, så kan stakken enten:

- konstant være over l indtil strengen w er processeret, **eller**
- stakken kan gå tilbage til niveau l før den går op igen, og så ned igen.

Husk at det ikke er muligt at gå under niveau l, og altså dermed ikke muligt at poppe stakken hvis man er på niveau l. Hvis vi i andet tilfælde siger at den state hvor stakken er tilbage på niveau l hedder r, start staten hedder p, og slut staten hedder q, så vil vi gerne lave to regler: $A_{pr} \overset{*}{\Rightarrow} u$ og $A_{pr} \overset{*}{\Rightarrow}$ hvor det gælder at $w = uv$. I første tilfælde er det lidt nemmere; siden niveau l ikke bliver ramt før state q, laver vi bare en regel $A_{rq} \overset{*}{\Rightarrow}$ *delen efter første push, og før sidste pop.* Vi vil nu gerne definere gramatikken baseret på mulighederne. Lad gramatikken $G = (V, \Sigma, R, S)$ hvor $V = \{A_{pq} \mid p, q \in Q(m)\}$, $S = A_{q_0 q_{accept}}$. $R =$ $\forall p, q, r, s \in Q(m)$ $\forall t \in \Gamma$ $\forall a, b \in \Sigma_\varepsilon$: hvis



og



er transitions af m, så tilføj reglen $A_{pq} \rightarrow aA_{rs}b$ til R. Ydermere, $\forall p, q, r \in Q(m)$ tilføj $A_{pq} \rightarrow A_{pr}A_{rq}$ til R og $\forall p \in Q$ tilføj $A_{pp} \rightarrow \varepsilon$ til R. □

Påstand 2.14

Denne grammatik beskrevet i beviset kun genkender sproget som PDA'en genkender, og ikke mere. Altså, vil stakken forblive samme størrelse efter processering af strengen x.

Bevis:

Vi beviser ved induktion over antallet af skridt i udledelsen.

Basis:

1 Skridt: Hvis der kun er ét skridt, så er $p = q$ og $A_{pp} \rightarrow \varepsilon$ så $x \in \Sigma^*$. Overvej en udledning med $k+1$ skridt og kig på den første regel brugt i udledningerne. Hvis $A_{pq} \rightarrow aA_{rs}b$, så $x = ayb$ når $A_{rs} \overset{*}{\Rightarrow} y$ sidste udledning har k skridt, så ved induktion gælder det at stakken forbliver samme størrelsen efter processering af x. Fra a til y kan der pushes, og fra y til b kan der poppes. □

Påstand 2.15

Hvis x kan bringe P fra p med en tom stak til q med en tom stak, så genererer A_{pq} x.

2.2.6 Non-kontekstfrie Sprog

Ligesom der findes non-regulære sprog findes der også non-kontekstfrie sprog. Et eksempel på et sprog der ikke er regulært er $\{a^n b^n \mid n \geq 0\}\}$. Vi så dog også at dette sprog var kontekstfrit, men hvilke sprog er så ikke kontekst-frie? Sjovt nok kan vi udvide sproget en lille smule til $\{a^n b^n c^n \mid n \geq 0\}$ til at gøre det ikke-kontekst frit. Vi introducerer her *pumpelemmaet for kontekstfrie sprog* til at hjælpe med at bevise at et sprog **ikke** er kontekst-frit.

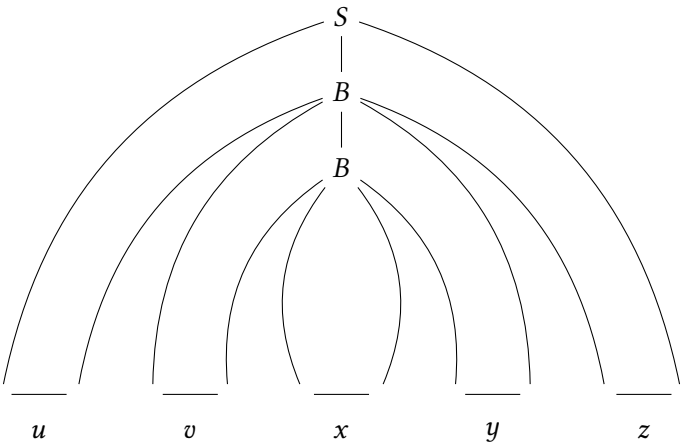
Teorem 2.16 (Pumpelemmaet for kontekstfri sprog)
For hvert kontekstfrit sprog L eksisterer der et $p \in \mathbb{N}$ således at $\forall w \in L$ hvor $|w| \geq p$ $\exists u, v, x, y, z \in \Sigma^*$ således at $w = uvxyz$ og

1. $uv^i xy^i z \in L, \forall i \geq 0$
2. $|vy| > 0$
3. $|vxy| \leq p$

Bevis:
Da L er et kontekstfrit sprog, så eksisterer der en kontekstfri grammatik $G = (V, \Sigma, R, S)$ i Chomsky Normal Form, således at $L = L(G)$. Lad pumpelængden $p = 2^{|V|+1}$.

Antag at $|w| \geq p$ og kig på et parsetræ, T, fra en udledning af w i G.[5] Vi ved at hvert blad i parsetræet korresponderer til en terminal, og dermed må der være $\geq 2^{|V|+1}$ blade i parsetræet. Da du kun kan fordoble antallet af elementerne, må højden af træet ($h(T)$) være **mindst** $|V|+2$. Dette betyder at der må være mindst 2 variabler der er gentagne.

[5] Jeg undlader at lave parsetræet her, da det sagtens kan forstås uden grafik.



Figur 2.4: Gentagelse i parsetræ for CFG

I Figur 2.4 kan man se hvordan gentagelsen i parsetræet fungerer. Dette betyder naturligvis derfor også at **både** v og y kan gentages 0 eller flere gange. □

2.2.7 Hovedpointer

Disse pointer er taget fra Jørgens Weekly Notes.

- En pushdown automat (PDA) er en nondeterministisk endelig automat forbedret med en stack, som gør PDA'en mere kraftfuld end en DFA eller NFA.
  - Dette er grundet stakkens uendelige længde. Dog er der stadig begrænsninger, da stakken kører på en LIFO måde.

- If we require that a PDA is deterministic, then we loose power in terms of which languages can be recognized. Note that this did not happen for Finite automata!
    - Deterministiske PDA'er mister noget kraft, og ud fra Siper's egen forklaring (fra hans kursus på YouTube) er de meget tekniske, men ikke specielt nødvendige for forståelse af automater eller lignende.
- I mentioned the following important fact without proof (see the paper listed under notes on bottom of the homepage) that **every context-free language over a one-symbol alphabet is also regular**. You are not supposed to be able to prove this, but you must be able to use this fact.
- There exist languages that are not context-free, and the pumping lemma can be used to prove that this is the case for a given language. As for regular languages, the proof goes by contradiction. A typical example of a non-context free language is $\{a^n b^n c^n \mid n \geq 0\}$.
    - Ligesom beskrevet før, er dette fordi stakken ikke er uendeligt kraftfuld, trods dens uendelige længde. Dette er primært grundet dens LIFO måde at agere på.
- Every regular language is context-free
    - Alle sprog som er regulære er også kontekst-frie. Det vil sige at både endelige automater og pushdown automater kan genkende regulære sprog.
- The class of context free languages is not closed under intersection. Note that this just means that there exists context free

languages L_1, L_2 such that $L_1 \cap L_2$ is not context-free. If we take $L_1 = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j\}$ and $L_2 = \{a^i b^j c^k \mid i, j, k \geq 0 \text{ and } j = k\}$, then both of these are context-free but $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ which we know is not context free.

  - Det går begge veje. Der eksisterer også **nogle** snit af kontekstfrie sprog hvis resultat er kontekstfrie. (Specielt givet at alle regulære sprog også er kontekstfri, og snit er lukket under regulære sprog.)

- PDA's are equivalent to context-free grammars, since, for any PDA A, one can construct a grammar G such that $L(A) = L(G)$, and vice versa.

  - Dette kan ses i sektion 2.2.5.

- The class of context-free languages is closed under union, concatenation, and star, but **not** under intersection and complement. Note that this is NOT the same as saying that the complement of a context-free language is never context free. For example both Σ^* and its complement, the empty set are context-free (they are also regular). Also $L = \{a^n b^n \mid n \geq 0\}$ and its complement are context-free (but not regular).

- The intersection of a context free language L_1 with a regular language L_2 is again a context-free language. This can be seen by observing that if we are given a PDA M_1 for the context-free language L_1 over Σ and a DFA M_2 for the regular language L_2, then we can make a PDA M which simulates M_1 and M_2 in parallel: think of having pairs of states (q, p) where q is the current state of M_1 and p is the current state of M_2. Then on a symbol

$a \in \Sigma$ the PDA M will behave as M_1 in the first coordinate (including updating the stack as M_1 would) and as M_2 in the second coordinate. Now we can make M accept a string w precisely when M_1 would accept w (be in an accepting state with empty stack AND M_2 would be in an accepting state after reading w). Thus M accepts w if and only if $w \in L_1 \cap L_2$, so $L_1 \cap L_2$ is context-free (as it is accepted by a PDA).

2.2.8 Opgaver

2.2

a. Use the languages $A = \{a^m b^n c^n \mid m, n \geq 0\}$ and $B = \{a^n b^n c^m \mid m, n \geq 0\}$ together with example 2.36 to show that the class of context-free languages is not closed under intersection.

Vi har to sprog, hvis snit vi skal finde, og derefter vise at dette sprog ikke er kontekst-frit. Snittet af disse to sprog er $\{a^n b^n c^n \mid n \geq 0\}$, som er vist i eksempel 2.36 til ikke at være regulært. (Hvorfor er det snittet?)

b. Use part (a) and DeMorgan's Law (Theorem 0.20) to show that the class of context-free languages is not closed under complementation.

Teorem 2.17 (DeMorgan's Law)
For alle to sæt A og B, $\overline{A \cup B} = \overline{A} \cap \overline{B}$

Hvis vi vender teoremet om, til at være $\overline{A} \cap \overline{B} = \overline{A \cup B}$ får vi at snittet af komplementet af to sprog er lig komplementet af fællesmængden af de samme to sprog. Vi ved dog at snittet ikke er kontekst-frit, så da

der kan sættes lighed mellem disse to kan de ikke være kontekst-fri. Vi kan også fjerne komplementet, så teoremet bliver transformeret til at blive $A \cap B = \overline{\overline{A} \cup \overline{B}}$

2.9

Give a context-free grammar that generates the language

$$A = \{a^i b^j c^k \mid i = j \text{ or } j = k \text{ where } i, j, k \geq 0\}$$

$$\begin{aligned} S &\to I \mid K \\ I &\to aIbC \mid \varepsilon \\ C &\to cC \mid \varepsilon \\ K &\to AbKc \mid \varepsilon \\ A &\to aA \mid \varepsilon \end{aligned}$$

2.20

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

Hvis A er kontekst-frit og B er regulært, og vi fjerner B har vi noget der kun er kontekst-frit tilbage, og dermed må A/B være kontekstfrit.

2.30

Use the pumping lemma to show that the following languages are not context-free.

a. $\{0^n1^n0^n1^n \mid n \geq 0\}$

Pumping lemma er Teorem 2.16.

Vi tager strengen $0^p1^p0^p1^p$, da $|w| = 4p$ satisfier vi kravet at længden er $\geq p$.

Vi inddeller således: $u = 0^p, v = 1^p, x = 0^p, y = 1^p, z = \varepsilon$.

Dette er umuligt, da 3. betingelse kræver at $|vxy| \leq p$

b. $\{0^n\#0^{2n}\#0^{3n} \mid n \geq 0\}$

Vi tager strengen $w = 0^p\#0^{2p}\#0^{3p}$. Det er nærmest umuligt at vise at det **er** kontekstfrit.

c. $\{w\#t \mid w \text{ is a substring of } t \text{ where } w, t \in \{a, b\}^*\}$

Vi tager strengen $a^pb^p\#a^pb^p$. Stortset uanset hvordan den opdeles, kan den ikke dække alle strenge, som er en del af sproget, og ender med at gå ud over sproget. F.eks. hvis $uvx = w_1, y = \#, z = w_2$.

d. $\{t_1\#t_2\#\cdots\#t_k \mid k \geq 2, \text{ each } t_i \in \{a, b\}^*, \text{ and } t_i = t_j \text{ for some} i \neq j\}$

Hvis vi tager samme streng som sidste opgave, får vi samme resultat.

2.32

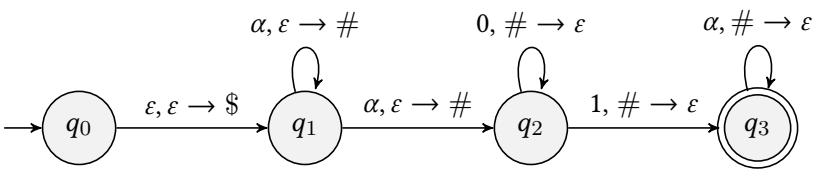
$\Sigma = \{1, 2, 3, 4\}, C = \{w \in \Sigma^* \mid \text{ in } w \text{ the number of 1s equals the number of 2s, and the number of 3s equals the number of 4s}\}$ Show that t is not context-free.

Givet streng $1^p3^p2^p4^p$, er strengen ikke mulig at pumpe.

2.47

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.



PDA'en gætter nondeterministisk på hvornår vi er halvvejs.

b. Give a CFG that generates B.

Det gider vi faktisk ikke, så i stedet lader vi bare som om vi har kørt algoritmen.

Tekst opgave

Show that the class of context-free languages is closed under union, that is, if L, L' are context-free languages over the same alphabet Σ, then $L \cup L'$ is also context-free.

Vi beviser ved at have en CFG, G:

$$S' \to S_1 \mid S_2$$
$$\vdots$$

Hvor S_1 er startsymbolet for grammatikken der genkender L, og S_2 er startsymbolet for grammatikken der genkender L'.

Problem 4 January 2002

a) Consider the context-free grammar $G = (V, \Sigma, S, R)$, given by $V := \{S, A, B, a, b\}$, $\Sigma := \{a, b\}$ and

$$R := \begin{cases} S \to AB, \\ S \to Ba, \\ A \to a, \\ A \to abAS, \\ A \to bAa, \\ B \to b, \\ B \to bSS, \\ B \to aSBB \end{cases}$$

Prove that all words in $L(G)$ contain the same number of occurrences of the letters a and b.

Vi beviser dette ved at lave det om til en Chomsky Grammar (Chomsky Normal Form).

Step One: Add new start symbol

$$\begin{aligned} C &\to S \\ S &\to AB \mid Ba \\ A &\to a \mid abAS \mid bAa \\ B &\to b \mid bSS \mid aSBB \end{aligned}$$

Step Two: Remove ε rules

$$
\begin{aligned}
C &\rightarrow S \\
S &\rightarrow AB \mid Ba \\
A &\rightarrow a \mid abAS \mid bAa \\
B &\rightarrow b \mid bSS \mid aSBB
\end{aligned}
$$

Step Three: Remove unit rules

$$
\begin{aligned}
C &\rightarrow AB \mid Ba \\
S &\rightarrow AB \mid Ba \\
A &\rightarrow a \mid abAS \mid bAa \\
B &\rightarrow b \mid bSS \mid aSBB
\end{aligned}
$$

Step Four: Split rules into parts

$$
\begin{aligned}
C &\rightarrow AB \mid Ba \\
S &\rightarrow AB \mid Ba \\
A &\rightarrow a \mid aE \mid bF \\
B &\rightarrow b \mid bG \mid aI \\
D &\rightarrow AS \\
E &\rightarrow bD \\
F &\rightarrow Aa \\
G &\rightarrow SS \\
H &\rightarrow BB \\
I &\rightarrow SH
\end{aligned}
$$

Step Five: Replace terminals with variables

$$
\begin{aligned}
C &\to AB \mid BJ \\
S &\to AB \mid BJ \\
A &\to J \mid JE \mid KF \\
B &\to K \mid KG \mid JI \\
D &\to AS \\
E &\to KD \\
F &\to AJ \\
G &\to SS \\
H &\to BB \\
I &\to SH \\
J &\to a \\
K &\to b
\end{aligned}
$$

Vi kan her se at der er lige mange occurences af J og K og dermed må den altid være sådan bum.

b) Consider the context-free language

$$L := \{a^n b^n | n \geq 1\} \cup \{a^n b^{2n} | n \geq 1\}$$

over $\Sigma = \{a, b\}$. Write down a Pushdown automaton M which precisely accepts L. This includes the definition of a transition table. Explain the way your PDA works.

Hint for b): To simplify the task your PDA is allowed to write in a single step several symbols onto the stack (but it can read only one at a time).

Jeg skriver i stedet for at lave den, fordi jeg ikke gider. Relativt simpelt, vi starter med at pushe 2 a'er, for hvert a. Så branhcer vi nondeterministisk ud, og popper hhv. et a og 2 a'er, indtil vi ikke kan mere, når vi ser et b. Hvis ingen af disse bliver genkendt er de ikke en del af sproget.

Problem 4 January 2007

a. Konstruer en kontekstfri grammatik G for følgende sprog

$$L := \{a^i b^j a^k \mid i + k \geq j; i, j, k \in \mathbb{N} \cup \{0\}\}$$

$$\begin{aligned} S &\longrightarrow AB \\ A &\longrightarrow aA \mid aAb \mid a \\ B &\longrightarrow Ba \mid bBa|a \end{aligned}$$

b. Betragt Sproget

$$L := \{a^i b^j \mid i = j^2, i, j \in \mathbb{N}\}$$

Er L kontekstfrit? Bevis dit svar.

Vi bruger pumpelemmaet (Teorem 2.16) til at bevise nonkontekstfrihed. Vi bruger strengen $a^{p^2} b^p$ hvis længde er p^3 hvilket tydeligt er $\geq p$. Uanset hvordan vi opdeler i substrenge $uvxyz$ vil en af reglerne altid blive brudt, mest sandsynligt regel. For eksempel $u = a^{p^2}, vxy = \varepsilon, z = b^p$, så bliver regel 2, $|vy|$ ikke overholdt.

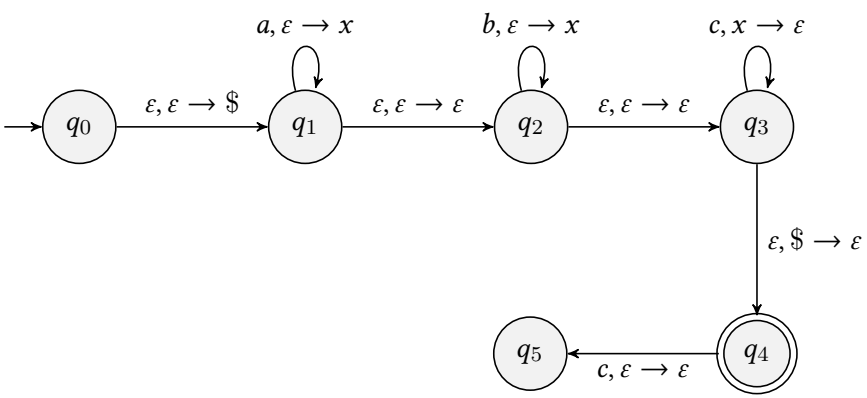
Problem 2 January 2008

Lad $L = \{a^n b^k c^m \mid n, k, m \geq 0 \text{ og } n + k = m\}$

a. Gør rede for at L ikke er regulært.

Vi beviser nonregulæritet vha. pumpelemmaet, i teorem 1.12. Vi tager strengen $a^p b^p c^{2p}$, hvilket er klart over p og er en del af sproget. Hvis vi så deler sproget op således at $x = a^p, y = b^p, z = c^{2p}$, ser vi at vi ikke kan pumpe på nogen som helst måde uden at det forlader sproget. Dermed er sproget ikke regulært.

b. Er L kontekstfrit?



Ja!

Problem 2, October 2010.

Betragt følgende sprog over alfabetet $\Sigma = \{a, b\}$.

$$L_2 = \{a^m a^n b^{n-1} a^{n-2} b^{n-3} \cdots a^1 : m = 1, 2, 3 \ldots \text{ og } n = 1, 3, 5\}$$

3

Church-Turing Tese

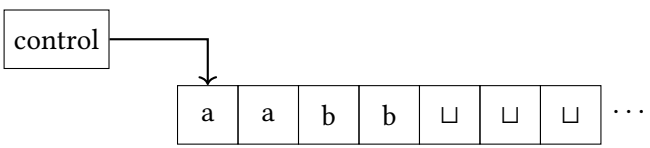
De maskiner vi har kigget på indtil videre har enten været med endelig hukommelse (Endelige Automater) eller med en uendelig hukommelse der kun fungerer i en LIFO stak-hukommelse. Dermed er de for begrænset til at kunne agere som modeller af generelt-formåls computere.

3.1 Turingmaskiner

I 1936 introducerede Alan Turing *turingmaskinen*, en deterministisk model som minder om en endelig automat, men med uendeligt og ubegrænset hukommelse. Denne model af en maskine kan alt det som en normal general purpose computer kan gøre. Dermed kan vi bruge denne model til at finde ud af begrænsninger m.m. om computere.

En turingmaskine bruger et uendeligt bånd som dets hukommelse. Den har en *båndhoved* som kan læse *og skrive* symboler og flytte sig selv rundt på båndet. Til at starte med indeholder båndet kan input

strengen, dog kan den skrive på dette bånd. Til at læse informationen går båndhovedet over de symboler der skal læses. Maskinen bliver ved med at komputere indtil den vælger at producere et output, enten *accept* eller *afvis*. Hvis den ikke kommer til et accept eller afvis state, så kører den for evigt, uden nogensinde at stoppe.



Figur 3.1: Skematik af en turingmaskine.

I Figur 3.1 kan der ses en skematik på en turingmaskine. Læg her mærke til at båndet er uendeligt langt, og efter input strengen er der bare blanke symboler, her betegnet som ⊔.

Følgende punkter opsummerer forskellen mellem en endelig automat og en turingmaskine:

1. En Turingmaskine kan både læse og skrive fra båndet.
2. Læs-skriv (read-write) hovedet kan bevæge sig både til athøjre og til venstre.
3. Båndet er uendeligt.
4. De speceielle states til at acceptere og afvise et input træder i kraft med det samme.

3.1.1 Formel Definition af en Turingmaskine

Trods vi næsten aldrig bruger den formelle beskrivelse af en turingmaskine, da de oftest ender med at blive alt for store, giver vi den generelle

formelle beskrivelse af en turingmaskine her.

Transitionsfunktionen i en turing maskine har formen $Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$. For eksempel betyder $\delta(q, a) = (r, b, L)$ at når båndet er over symbol a og maskinen er i state q, så skal den skrive b til symbolet hvor a tidligere stod, og gå til state r. L betyder her at den skal bevæge sig til venstre.

Definition 3.1 (Formel Definition af en Turingmaskine)
En **Turingmaskine** er en 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, hvor Q, Σ, Γ alle er endelige sæt, og

1. Q er sættet af states.
2. Σ er inputalfabetet ikke indeholdende det blanke symbol, $\sqcup$.
3. Γ er båndalfabetet, hvor $\sqcup \in \Gamma$ og $\Sigma \subseteq \Gamma$.
4. $\delta, Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ er transitionsfunktionen.
5. $q_0 \in Q$ er startstaten.
6. $q_{\text{accept}} \in Q$ er accept staten.
7. $q_{\text{reject}} \in Q$ er afvis staten, hvor $q_{\text{accept}} \neq q_{\text{reject}}$.

Bemærk at, på trods af at Sipser ikke skriver det som en mulighed, kan man nemt implementere "stay" (her skrevet S) som en mulighed, fromfor kun at vælge left eller right. Du kan gøre dette ved at først skrive, så gå til højre, så gå til venstre uden at gøre noget med symbolet.

3.1.2 Turingmaskine Komputering

En turingmaskine komputere ved at starte sit båndhoved på den venstremest symbol på båndet. Inputtet $w = w_1 w_2 \dots w_n \in \Sigma^*$ er på de

første (venstremest) n pladser på båndet, og det resterende er blankt. Siden Σ ikke indeholder de blanke symbol, kan man være sikker på at så snart det blanke symbol kommer på båndet, har man læst inputtet færdigt. Hvis hovedet nogensinde forsøger at bevæge sig mere til venstre end muligt, så forbliver hovedet på den venstremest symbolplacering, trods den "skal" gå til venstre. Når M er begyndt bevæger den sig efter reglerne beskrevet i transitionsfunktionen, indtil den enten accepterer eller afviser inputtet og dermed stopper. Hvis den hverken accepterer eller afviser inputtet kører den forevigt.

En *konfiguration* er en kombination af de følgende tre ting:

- Den nuværende state.
- Det nuværende indhold af båndet.
- Hovedet nuværende lokation.

De her konfigurationer bliver oftest repræsenteret på følgende måde: Givet en state q og to strenge u og v over båndalfabetet Γ, skriver vi uqv for konfigurationen hvor den nuværende state er q, båndindholdet er uv og hovedets lokation er ved det andet symbol, altså v.

Vi siger at en kofngiruation C_1 *giver* konfigurationen C_2 hvis turingmaskinen lovligt kan gå fra C_1 til C_2 på et enkelt skridt.

Startkonfigurationen af M på input w er konfigurationen q_0w. Altså hvor staten er helt i start, og resterende af strengen, w efterfølger. I en *accepterende konfiguration* er staten i en konfiguration q_{accept}. I en *afvisende konfiguration* er staten q_{reject}. Accept og afvisekonfigurationer er *standsende konfigurationer*, altså konfigurationer som ikke giver flere konfigurationer.

Vi kalder samlingen af strenge som M accepterer *sproget af* M eller *sproget genkendt af* M, betegnet $L(M)$.

Definition 3.2
Kald et sprog *Turing-genkendeligt* hvis en Turingmaskine genkender det.

Dette kan også beskrives $L(m) = \{w \mid q_0 w \overset{*}{\Rightarrow} u q_{\text{acc}} v \text{ for nogen } u, v \in \Gamma^*\}$

Når en turingmaskine startes er tre resultater muligt, enten vil maskinen *acceptere*, *afvise* eller *løkke*[1], hvor *løkke* betyder at maskinen vil køre forevigt.

Vi kalder maskiner der aldrig kommer i en løkke for beslutningstagere[2], fordi de altid beslutter hvorvidt en streng skal accepteres eller afvises. En beslutningstager som genkender et sprog, siges også at *beslutte* det sprog.

Definition 3.3
Kald et sprog Turing-afgjort eller simpelt afgjort, hvis en turingmaskine afgører (beslutter) det.

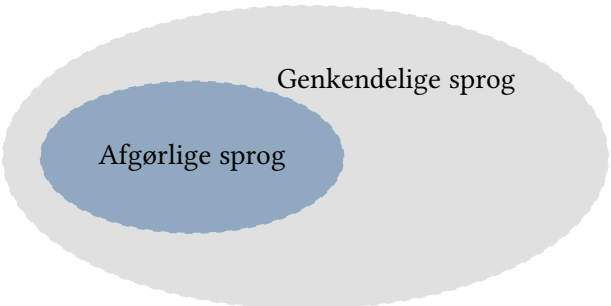
Dette kaldes også *recursively enumerable* i litteraturen.

I figur 3.2 ses der hvordan de afgørlige sprog kun er en delmængde af alle de genkendelige sprog. Under de afgørlige sprog ligger også kontekstfriie sprog og regulære sprog.

3.2 Varianter af Turingmaskiner

Vi kalde andre versioner af turingmaskinen, herunder maskiner med flere bånd eller nondeterministiske maskiner for *varianter*. Den originale model og alle dens varianter har samme deskriptive kraft. Vi kal-

[1] *loop* på engelsk
[2] Måske? Deciders på engelsk.



Figur 3.2: Univers af sprog

der dette *robusthed*, altså, at maskinerne alle beskriver samme klasse af sprog.

3.2.1 Multibånds Turingmaskiner

En **multibånds turingmaskine** er som en normal turingmaskine, men med mere end ét bånd (hvor antallet ikke ændrer sig under kørsel.) Transitionsfunktionen ændres så den kan køre på mere end ét bånd:

$$\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k$$

hvor k er antallet af states. Dermed fungerer transitionsfunktionen ved at tage den nuværende state i alle tapes og giver resultatet i alle states, eksempelvis: $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, L, R, \ldots, L)$.

Teorem 3.4
Hver multibånds turingmaskine har en ækvivalent enkeltbånds turingmaskine.

Bevis:
For at bevise dette, skal vi vise at enhver multibånds Turingmaskine M kan konverteres til en enkeltbånds Turingmaskine S. Vi siger at M har k bånd. Så bruger S symbolet # til at indikere at et nyt bånd starter. Derudover skal S også holde styr på lokationen af hovederne fra diverse bånd. Den gør dette ved at skrive en bolle over symbolet: $\mathring{b}$.

Så en maskine med tre bånd som f.eks.:

aaaaababababb
bbbbb**b**bbbbba
a**b**aaababbaba

Hvor **tyk skrift** indikerer hovedets placering, bliver lavet om til i S:

$$\mathring{a}\text{aaaababababb\#bbbbb}\mathring{b}\text{bbbbba\#a}\mathring{b}\text{aaababbaba\#}.$$

Vi definerer nu S.
S = "På input $w = w_1 \cdots w_n$":

1. Først konverterer S til enkeltbånd.

2. Til at simulere en enkelt bevægelse, scanner S fra den første # til $k + 1$'e #, som er slutningen i højresiden, så den kan finde ud af hvad symbolerne under de virtuelle hoveder er. Så laver S en til passthrough til at opdatere ifølge reglerne. Hvis på noget tidpsunkt, at S vil flytte til højre til et #, så rightshifter S alt derfra, da det betyder at båndet normaltvis ville shifte ud i blanke symboler.

□

Corollary 3.5
Et sprog er Turing-genkendeligt hvis og kun hvis en multibånds Turingmaskine genkender det.

Bevis:
En Turingmaskine er genkendt af en multibåndsmaskine med ét bånd. Den anden vej bevises i Teorem 3.4. □

3.2.2 Nondeterministisk Turingmaskine

En nondeterministisk turingmaskine fungerer nondeterministisk ligesom PDA og NFA. Den brancher ud for hver mulighed. Transitionsfunktionen beskrives således:

$$\delta : Q \times \Gamma \longrightarrow P(Q \times \Gamma \times \{L, R\})$$

Hvis en af de her branches ender i en accept state, så accepteres inputtet.

Teorem 3.6
Hver nondeterministisk turingmaskine har en ækvivalent deterministisk Turingmaskine.

Vi vil gerne bevise ved at have en deterministisk turingmaskine D til at simulere alle branches der er mulige v.h.a. nondeterminisme. Vi designer D til at bruge breadth-first-search til at kigge alle branches igennem.

Bevis:
Den simulerende turingmaskine D har tre bånd. Bånd 1 indeholder altid

strengen uden at ændre den. Bånd 2 har en kopi af den nondeterministke turingmaskine N's bånd på en gren af dens nondeterministiske komputering. Bånd 3 holder styr på D's lokation i N's nondeterministiske komputationertræ. □

Corollary 3.7
Et sprog er turing-genkendeligt hvis og kun hvis en nondeterministisk Turingmaskine genkender det.

Bevis:
Enhver deterministisk Turingmaskine er automatisk også en nondeterministisk turingmaskine. Den anden retning følger fra 3.6. □

Vi kalder en nondeterministisk turingmaskine en **beslutningstager** hvis den stopper på alle branches.

Indeks