

# DM553 - Komplexitet og Beregnlighed

Grundige noter til kursets indhold og opgaver

Kevin Vinther  
2024

# Indhold

<b>1</b>	<b>Endelige Automater</b>	<b>5</b>
1.1	Deterministiske Endelige Automater . . . . .	5
1.2	Nondeterministiske Endelige Automater . . . . .	9
1.3	Ækvivalens af NFA'er og DFA'er . . . . .	13
1.4	Regulære Operationer . . . . .	15
1.5	Pumpelemmatet og ikke-regulære sprog . . . . .	22
1.6	Hovedpointer . . . . .	25
1.7	Opgaver . . . . .	30
1.7.1	Sipser 1.9 . . . . .	31
1.7.2	Sipser 1.11 . . . . .	35
1.7.3	Sipser 1.12 . . . . .	35
1.7.4	Sipser 1.31 . . . . .	36
1.7.5	Sipser 1.29(b) . . . . .	37
1.7.6	Sipser 1.30 . . . . .	38
1.7.7	Sipser 1.36 . . . . .	38
1.7.8	Sipser 1.43 . . . . .	38
1.7.9	Sipser 1.49 . . . . .	39
<b>2</b>	<b>Kontekstfrie Sprog</b>	<b>40</b>
2.1	Kontekstfrie Grammatikker . . . . .	40

2.1.1	Tvetydighed . . . . .	44
2.1.2	Chomsky Normal Form . . . . .	45
2.2	Pushdown Automater . . . . .	49
2.2.1	Formel Definition af Pushdown Automat . . .	50
2.2.2	Pushdown Automat Komputering . . . . .	51

# 1

## Endelige Automater

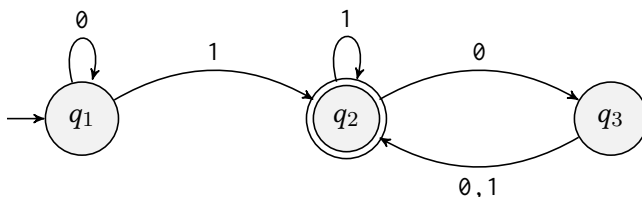
Endelige automater er en type abstrakt maskine med endelig hukommelse, som kan bruges til for eksempel at matche mønstre med.

### 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.  $\Sigma$  er et endeligt sæt af symboler kaldet alfabetet
3.  $\delta$  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.  $\Sigma = \{\emptyset, 1\}$

3.  $\delta$  beskrives som en tabel:

	$\emptyset$	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  $\Sigma$ .  $M$  accepterer her  $w$  hvis en sekvens af states  $r_0, r_1, \dots, r_n$  i  $Q$  eksisterer med tre betingelser:

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

Det vil altså sige at **betingelse 1** siger at maskinen starter i den initiale 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.

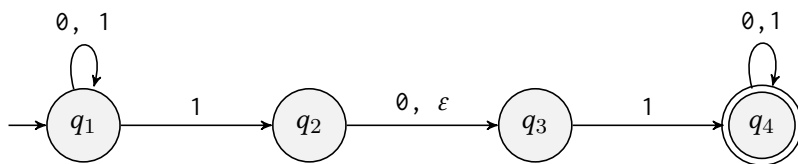
### **Definition 1.3**

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  $\varepsilon$  (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 forvirre 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  $\emptyset$  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  $\epsilon$ , 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.  $\Sigma$  er et endeligt alfabet
3.  $\delta : Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$  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  $\Sigma_\epsilon$  blot er  $\Sigma \cup \{\epsilon\}$ , 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.  $\delta$  vises som en tabel:

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

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

$\Sigma$ . 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  $\Sigma_\epsilon$  og en sekvens af states  $r_0, r_1, \dots, r_m$  eksisterer i  $Q$  med tre betingelser:

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

#### 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  $\varepsilon$  pile er tilstæde, for at simplificere processen. Efter dette udvider vi så  $\varepsilon$  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  $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for en } r \in R\}$ .

Dette kan også skrives således:

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

3.  $q'_0 = \{q_0\}$

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

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

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

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:  $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a)) \text{ for en } r \in R\}$   $\square$

## 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, klene star), som følger:

1. **Fællesmængde:**  $A \cup B = \{x \mid x \in A \text{ eller } x \in B\}$ .
2. **Sammenkædning:**  $A \circ B = \{xy \mid x \in A \text{ og } y \in B\}$ .
3. **Stjerne:**  $A^* = \{x_1x_2 \dots x_k \mid k \geq 0 \text{ og hvert } x_i \in A\}$ .

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

### Eksempel 1.8

Lad alfabetet  $\Sigma$  være det danske alfabet  $\{a, b, \dots, \text{\AA}\}$ . Hvis  $A = \{\text{god, dårlig}\}$  og  $B = \{\text{bog, video}\}$ , så:

$$A \cup B = \{\text{god, dårlig, bog, video}\}$$

$$A \circ B = \{\text{godvideo, godbog, dårligvideo, dårligbog}\}$$

$$A^* = \{\epsilon, \text{god, dårlig, godgod, dårligdårlig, goddårlig,} \\ \text{dårliggod, godgodgod, godgoddårlig, goddårliggod, \dots}\}$$

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<sup>1</sup>.

## Teorem 1.9

Klassen af regulære sprog er lukket under fællesmængde operationen.

Vi beviser dette ved at konstruere 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 konstruerer en ny DFA,  $M$  som genkender  $A_1 \cup A_2$ , hvor  $M = (Q, \Sigma, \delta, q_0, F)$ , hvor:

1.  $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ og } r_2 \in Q_2\}$ .

Dette sæt er det **Kartesiske Produkt** af sætterne  $Q_1$  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.  $\Sigma$  alfabetet, er det samme som i både  $M_1$  og  $M_2$ . Hvis det ikke er ville  $\Sigma = \Sigma_1 \cup \Sigma_2$ .
3.  $\delta$  er transitionsfunktionen som defineres som følgende. For hvert  $(r_1, r_2) \in Q$  og hvert  $a \in \Sigma$ , lad

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

---

<sup>1</sup>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.

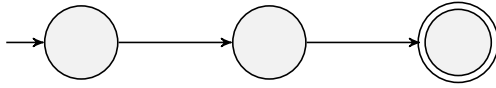
□

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

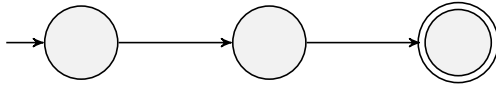
### Bevis:

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.

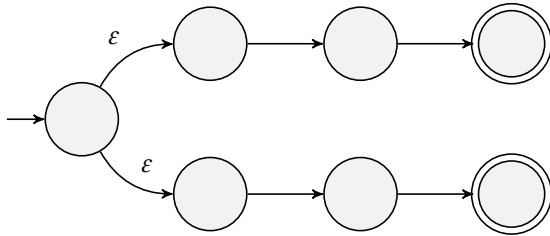
$$N_1, L(N_1) = A_1$$



$$N_2, L(N_2) = A_2$$



$$N, L(N) = A_1 \cup A_2$$



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  $a \in \Sigma_\epsilon$ :

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

□

### **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 konstruerer 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.  $\delta$  for alle  $q \in Q$  og alle  $a \in \Sigma_\epsilon$ ,

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

□

### **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.  $\delta$  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 parenteser er balanceret**, altså  $L = \{w \mid w \text{ hver ( er (på et tidspunkt) efterfulgt af et )}\}$ . Dette gælder strenge såsom  $(((((()))))())$  og  $(((((((((((((\dots)))))))))))))$ . 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.  $(\forall i \geq 0)(xy^iz \in A)$
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 “abc-def” 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  $z = \varepsilon$ .)

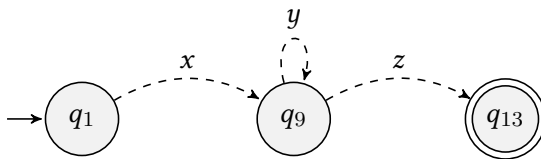
Læg mærke til i betingelserne at det er tilladt for enten  $x$  eller  $z$  at være lig  $\varepsilon$ , men ikke  $y$ , da denne skal kunne “pumpes”.

### 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  $r_1, \dots, r_{n+1}$  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.<sup>2</sup> Vi kalder den første  $r_j$  og den anden  $r_l$ . Fordi  $r_l$ . Da  $n + 1 \geq p + 1$ , gennemgås  $r_l$  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.  $\square$



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-

---

<sup>2</sup>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 starten 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  $D = \{1^n \mid n \geq 0\}$  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  $xy^i z$  også en del af sproget. For at bevise det, lad os kigge på to strenge:  $xyz$  og  $xy^2 z$ . 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  $xy^2 z \leq p^2 + p$  men vi vil gerne have  $(p+1)^2 = p^2 + 2p + 1$ . Fordi betingelse 2 siger at  $p \neq \varepsilon$  gælder det at  $|xy^2 z| > p^2$ , og derfor er  $xy^2 z$  imellem to andengradspotenser, og dermed er  $xy^2 z \notin D$  og  $D$  er dermed ikke regulært.

### Eksempel 1.14 (Sipser Eksempel 1.77)

Vi bruger pumpelemmaet til at vise at sproget  $E = \{0^i 1^j \mid i > j\}$  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  $xy^0 z$  også gælde. Men da  $xy^0 z = xz$  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 begrænsninger 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*<sup>3</sup> 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.

---

<sup>3</sup>Graf-teoretisk for en måde at gå fra A til B

- Hvis strengen  $w$ ,  $|w| = n$  går igennem en sekvens af states  $q_0 \cdot \dots \cdot q_n$  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 polynomiell 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 polynomiell, 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 konverteret.
- 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  $\bar{L}$  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  $w_1, w_2, \dots, w_k$  for et endeligt heltal  $k$ , så lav NFA'erne  $M(w_i)$ ,  $i = 1, 2, \dots, k$  og byg (i  $k - 1$  fællesmængde skridt) en NFA hvis sprog er præcis  $L$ . Så hvert non-regulært sprog indeholder arbitrært 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 modstanderen vælge strenge  $x, y, z$  over  $\Sigma$  således at

1.  $s = xyz$
2.  $xy^iz \in L$  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  $xy^iz \notin L$ , 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 he must cross to continue his travel. Fortunately, there is a small boat at the shore which he can use. However, the boat is so small that the man cannot bring more than himself and exactly one more item along (from  $\{w, g, c\}$ ). The man knows that if left alone with the goat, the wolf will surely eat it and the goat if left alone with the cabbage will also surely eat that. The man's task is hence to devise a transportation scheme in which, at any time, at most one item from  $\{w, g, c\}$  is 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  $\Sigma$  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,  $gm\bar{c}g$  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

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

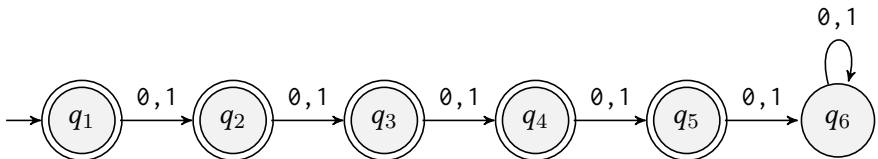
1.6i

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

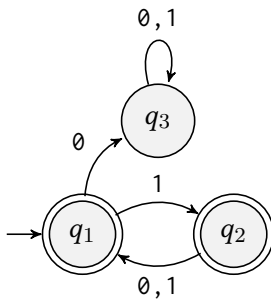
Alfabetet i begge sprog er  $\{0, 1\}$

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.  $q_1$



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

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

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

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

$$3. \delta =$$

	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:

$$1. Q = \{q_{11}, q_{12}, q_{13}, q_{14}, q_{15}, q_{16}\} \cup \{q_{21}, q_{22}, q_{23}\}$$

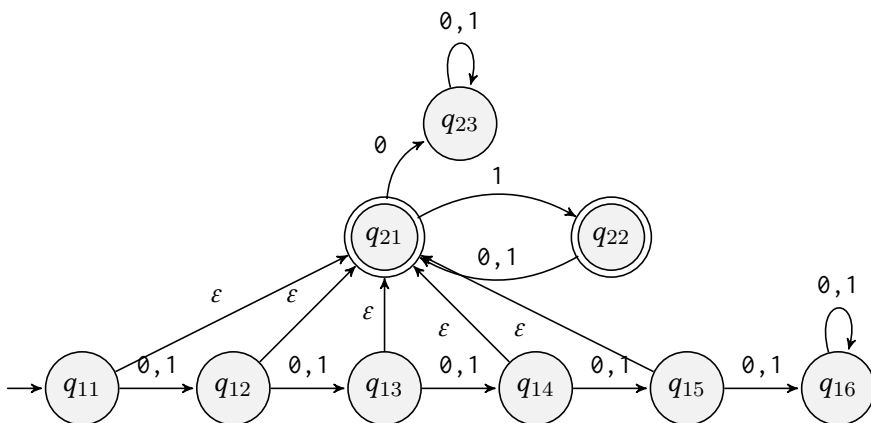
$$2. \Sigma = \{0, 1\}$$

$$3. \delta \text{ undlades}$$

$$4. q_{11}$$

$$5. F = \{q_{21}, q_{22}\}$$

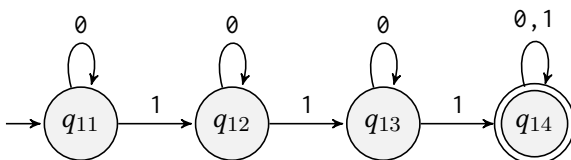
Vi laver det om til et state diagram:



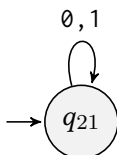
b. Exercises 1.6b and 1.6m

Nu når vi kender rutinen bliver denne langt mindre formel:

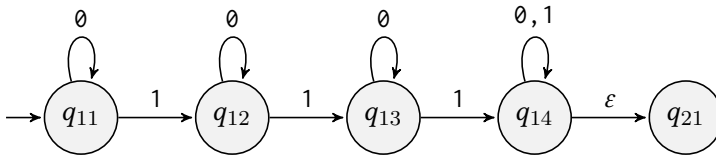
$$M_1 = \{w \mid w \text{ contains at least three } 1s\}$$



$M_2 =$  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  $\epsilon$  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

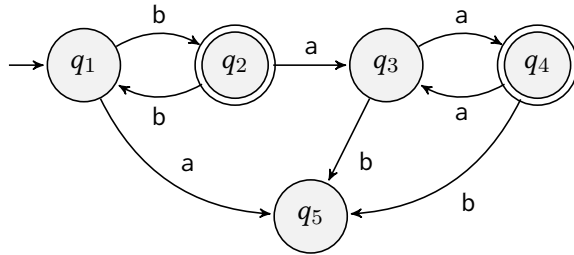
$$D = \{w \mid w \text{ contains an even number of } a\text{'s and an odd number of } b\text{'s and does not contain substring } ab\}.$$

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

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  $M_R = (Q, \Sigma, \delta, q_0, F)$  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 omvendte pilen fra hvad den var tidligere, i.e., hvis  $\delta_1(q_{100}, a) = q_{101}$ , så  $\delta(q_{101}, a) = q_{100}$ , hvor  $\delta_1$  er fra den originale automat, og  $\delta$  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.

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

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

Vi tager strengen  $a^pba^pba^pb$ , hvor  $w = a^pb$ . Lad  $aabaabaab \in A_2$ . 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.  $\square$

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

Regulært udtryk:

$$a^{n*4}$$

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,  $DROP-OUT(A) = \{xz|xyz \in A \text{ where } x, z \in \Sigma^*, y \in \Sigma\}$ .

---

<sup>4</sup>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  $B = \{1^k y \mid y \in \{0, 1\}^* \text{ and } y \text{ contains at least } k \text{ 1s, for } k \geq 1\}$ . Show that  $B$  is a regular language.

Bevis af regulært udtryk:

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

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

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

# 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.

### 2.1 Kontekstfrie Grammatikker

En grammatik består af en samling af **substitueringsregler**, også kaldet *produktioenr*. 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  $G_1$ :

$$\begin{aligned}A &\rightarrow \emptyset A1 \\A &\rightarrow B \\B &\rightarrow \#\end{aligned}\tag{G_1}$$

Symbolet på venstresiden kaldes en *variabel*<sup>1</sup>. 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<sup>2</sup>. 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<sup>3</sup> 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 grammatikken  $G_1$  udlede strengen  $\emptyset\emptyset\emptyset\#111$ . En udledning af denne streng i  $G_1$  er følgende:  $A \Rightarrow \emptyset A1 \Rightarrow \emptyset\emptyset A11 \Rightarrow \emptyset\emptyset\emptyset A111 \Rightarrow \emptyset\emptyset\emptyset B111 \Rightarrow \emptyset\emptyset\emptyset\#111$ . 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  $\emptyset\emptyset\emptyset\#111$  i grammatikken  $G_1$ .

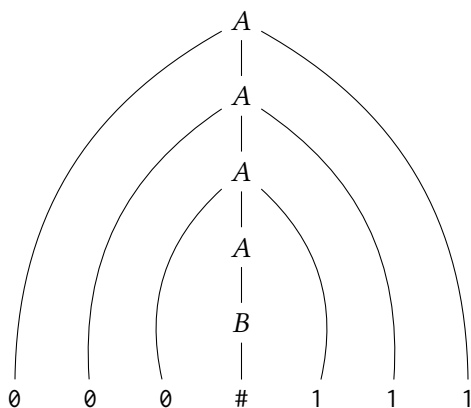
Alle strenge du kan udlede på denne måde er en del af sproget, og udgør tilsammen det kontekstfri sprog som  $G_1$  genkender. Hvis du ikke har lagt mærke til det nu, er denne meget simple kontekstfri grammatik faktisk et eksempel på et sprog der *ikke* kan genkendes af en endelig

---

<sup>1</sup>Kært barn har mange navne, jeg har også hørt non-terminal og, simpelt, LHS.

<sup>2</sup>Dette vil vi komme ind på i mere detalje senere.

<sup>3</sup>En udledning bliver beskrevet i mere detalje i næste afsnit.



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

automat. Ofte, fremfor at skrive  $A \rightarrow 0A1$  og så på en ny linje skrive  $A \rightarrow B$ , så skriver man blot  $A \rightarrow 0A1|B$ , 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  $(V, \Sigma, R, S)$ , hvor

1.  $V$  er et endeligt sæt kaldet **variabler**
2.  $\Sigma$  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 variable, 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,  $u \Rightarrow^* v$ , hvis  $u = v$ , eller hvis en sekvens  $u_1, u_2, \dots, u_k$  eksisterer når  $k \geq 0$  og  $u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_k \Rightarrow v$ . Sproget af grammatikken er  $\{w \in \Sigma^* \mid S \Rightarrow^* w\}$ .

### 2.1.1 Tvetydighed

En grammatik der kan generere den samme 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:

$$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle | \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle | (\langle \text{EXPR} \rangle) | a$$

Strengen  $a + a \times a$  kan generes tvetydigt: enten

$$\langle \text{EXPR} \rangle \Rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle \Rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \Rightarrow \dots \Rightarrow a + a \times a$$

eller

$$\langle \text{EXPR} \rangle \Rightarrow \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \Rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \Rightarrow \dots \Rightarrow a + a \times a$$

.

#### Definition 2.2 (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.)

### 2.1.2 Chomsky Normal Form

Chomsky Normal Form er en simple form af en kontekstfri grammatik.

#### **Definition 2.3** (Chomsky Normal Form)

En kontekstfri grammatik er i **Chomsky Normal Form** hvis hver regel har formen

$$A \rightarrow BC$$

$$A \rightarrow a$$

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  $S \rightarrow \varepsilon$ , hvor  $S$  er startvariablen.

#### **Teorem 2.4**

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  $A \rightarrow \varepsilon$  (kaldet  $\varepsilon$ -regler.) Vi eliminerer også alle **unit rules**, som har formen  $A \rightarrow B$ . Til sidst konverterer vi resten til at være i ordentlig form.

#### **Bevis:**

**Skridt 1:** Først tilføjer vi en ny variabel  $S_0$  og reglen  $S_0 \rightarrow S$ , hvor  $S$  er det originale startvariabel.

**Skridt 2:** Efter dette fjerner vi alle regler af formen  $A \rightarrow \varepsilon$ , hvor  $A \neq S$ . Når dette er gjort fjerner vi alle regler hvor  $A$  er på højresiden af en regel, og tilføjer en ny regel med  $A$  fjernet. Eksempelvis bliver  $R \rightarrow$

$uAv$  til  $R \rightarrow uv$  og  $R \rightarrow uAvAw$  bliver til tre regler  $R \rightarrow uvAw$ ,  $R \rightarrow uAvw$ ,  $R \rightarrow uvw$ . Hvis vi har reglen  $R \rightarrow \varepsilon$  tilføjer vi  $R \rightarrow \varepsilon$ , undtagen hvis vi allerede har fjernet denne regel. Vi forstætter dette indtil alle  $\varepsilon$ -regler er væk (undtagen startvariablen.)

**Skridt 3:** Efter vi har gjort dette tager vi os af *unit rules*. Vi fjerner en unit rule  $A \rightarrow B$ . Så, når  $B \rightarrow u$  forekommer, tilføjer vi  $A \rightarrow u$  (bemærk at vi ikke fjerner  $B \rightarrow u$ , 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  $A \rightarrow u_1u_2 \cdots u_k$  hvor  $k \geq 3$  og hvert  $u_i$  er en variabel eller terminalt symbol, med reglerne  $A \rightarrow u_1A_1$ ,  $A_1 \rightarrow u_2A_2$ ,  $A_2 \rightarrow u_3A_3, \dots$ , og  $A_{k-2} \rightarrow u_{k-1}u_k$ .  $A_i$ 'erne er nye variabler. Vi erstatter en hver terminal  $u_i$  i de tidligere regler men den nye variabel  $U_i$  og tilføjer reglen  $U_i \rightarrow u_i$ .  $\square$

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

## Eksempel 2.5

Vi konverterer følgende grammatik til CNF:

$$S \rightarrow ASA \mid aB$$

$$A \rightarrow B \mid S$$

$$B \rightarrow b \mid \varepsilon$$

### Skridt 1:

Første skridt er at tilføje en ny startvairabel. Vi gør dette ved at introducere reglen  $S_0 \rightarrow S$

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

### **Skridt 2:**

Vi fjerner  $\varepsilon$ -reglerne.

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

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

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

### **Skridt 3:**

Vi fjerner nu unit-reglen  $S \rightarrow S$ :

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

Og nu  $S_0 \rightarrow S$ :

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

Nu fjerner vi reglen  $A \rightarrow B$ :

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

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

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

#### **Skridt 4:**

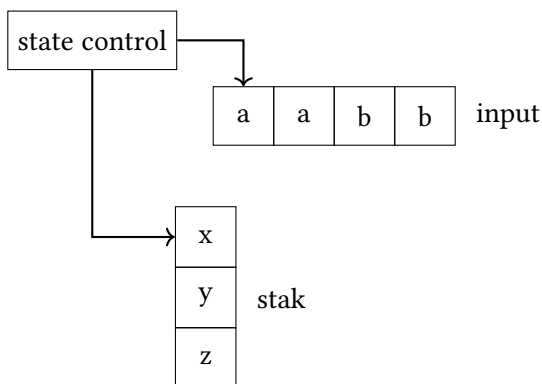
Nu skal vi konvertere resten.

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



## 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 2.6

Sproget  $\{0^n 1^n | n \geq 0\}$  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 ækvivalente 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:  $\Gamma$  som er alfabetet for stakken.

**Definition 2.7** (Formel Definition af en Pushdown Automat)

En *pushdown automat* er en 6-tuple  $(Q, \Sigma, \Gamma, \delta, q_0, F)$ , hvor  $Q, \Sigma, \Gamma$ , og  $F$  alle er endelige sæt, og

1.  $Q$  er sættet af states
2.  $\Sigma$  er input alfabetet
3.  $\Gamma$  er stak alfabetet
4.  $\delta : Q \times \Sigma_{\epsilon} \times \Gamma_{\epsilon} \longrightarrow P(Q \times \Gamma_{\epsilon})$  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  $r_0, r_1, \dots, r_m \in Q$  og strenge  $s_0, s_1, \dots, s_m \in \Gamma^*$  eksisterer som tilfredsstiller følgende betingelser, hvor  $s_i$  repræsenterer 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  $i = 0, \dots, m-1$  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.