Indhold

1

Endelige Automater

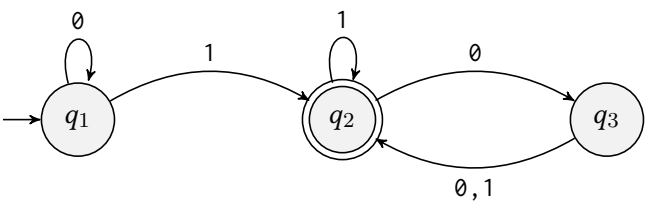
Endelige automater er en type abstrakt maskine med endelig hukommelse, som kan bruges til for eksempel at matche mønstre med.

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. $r_n \in F$

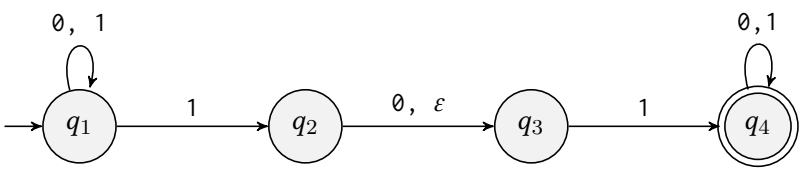
Det vil altså sige at **betingelse 1** siger at maskinen starter i den initielle state. **Betingelse 2** siger at ud fra reglerne af transitionsfunktionen går maskinen fra state til state. **Betingelse 3** siger at maskinen accepterer dets input hvis den ender i en accept state.

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 vær 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.

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. $F' = \{R \in Q |\ R$ 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:

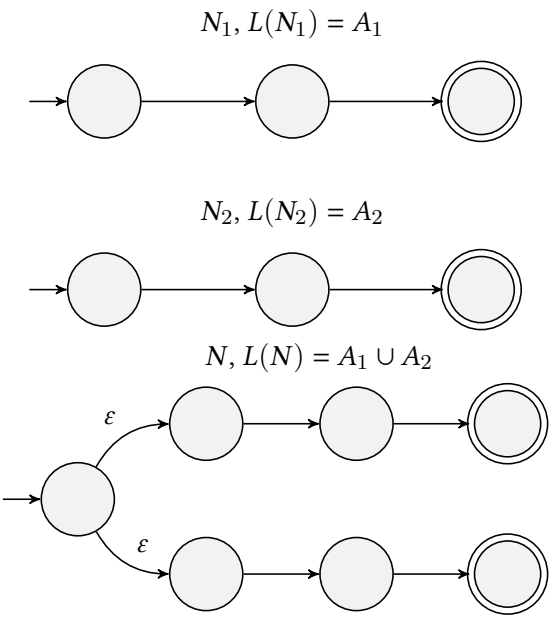
$$F = \{(r_1, r_2) | r_1 \in F_1 \text{ eller } r_2 \in F_2\}$$

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.

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 Balanced Parentheses Problemet, altså $L = \{w | w \text{ has balanced parentheses}\}$. Dette gælder strenge såsom `((((()))))(())` og `((((((((((...))))))))))`. Et hovedteorem i at finde ud af om et sprog, som for eksempel sproget beskrevet før, er regulært, er *pumpe lemmaet*. Ifølge pumpe lemmaet 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 1.11 (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. $|xy| \leq p$

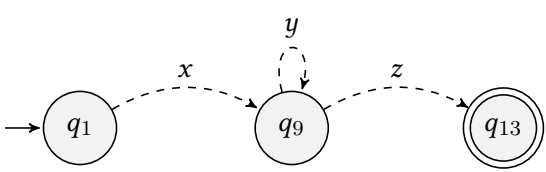
Betingelse 1 siger at en del af strengen, y skal kunne "pumpes" et uendeligt antal gange (hvor $y^0 = \varepsilon$.) Det vil sige at hvis strengen "abcdef" er en del af et regulært sprog, og $y = cde$, så vil "abf" også være en del af sproget, ligesom "abcdecdecdecdef". **Betingelse 2** siger at $y \neq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

[2] Af dueslagsprincippet

1.6 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.