

1. Seminar EVU RegAut

Sigurd Meldgaard

Datalogisk Institut
Århus Universitet
`stm@cs.au.dk`

?? . ?? 2010

Plan

- Hvad er Regularitet og Automater
- Praktiske oplysninger om kurset
- Regulære udtryk
- Induktionsbevis
- Frokost
- Endelige automater
- Skelnelighed, Produktkonstruktion
- Præsentation af Java projekt

Plan

Introduktion

Regulære udtryk

Induktionsbevis

Regulære automater

Skelnelighed og produktkonstruktion

dRegAut pakken

Automater til modellering og verifikation

Hvad er formålet med Regularitet og Automater?

- At præsentere matematiske teknikker og centrale begreber, der anvendes i datalogi
 - Rekursive definitioner, induktionsbeviser
 - Formelle sprog
 - Modeller for beregnelighed
 - Regularitet (“egenskaber som generelt kendetegner beregningsprocesser i it-systemer med begrænset mange tilstande”)
- Fundament for andre kurser
 - Logik og Beregnelighed
 - Oversættelse,
 - Sprog og Semantik
 - Søgning og Optimering, ...

Tekstgenkendelse

- Specificere og genkende tekststreng
- søgning i tekster (Unix grep)
- leksikalsk analyse i oversættere (flex)
- HTML input validering (PowerForms)
- ...
-
- Konkret anvendelse af regulære udtryk og endelige automater

Eksempel...



Eks. HTML formularer

HTML formularer indeholder input-felter, hvor brugeren kan indtaste tekststrengene.

Oprettelses formular, til brugere ved statsbiblioteket - Microsoft Internet Explorer

Filer Redger Vis Forebruke Funktioner Hjælp

Tilbage

STATSBIBLIOTEKET

Personlige oplysninger.

Indtast CPR-nummer:

Indtast fornavn(e):

Indtast efternavn:

Indtast e-mail:

Næste side

©Statsbiblioteket 2003

Udført Denne computer

For eksempel

- datoer
- telefonnumre
- CPR-numre
- emailadresser
- URL'er
- ...

HTML input validering

- Brugeren må ikke indtaste ugyldige strenge
- Den traditionelle løsning: Programmer input validering i JavaScript (til browseren – så input valideres løbende mens formularen udfyldes), og Java (til serveren – for det tilfælde at browseren ikke udfører JavaScript-koden)
- Problemer:
 - Det er svært at programmere JavaScript, der virker på alle (nyere) browsere
 - Vi skal skrive den samme kode i to forskellige sprog
 - Store dele af koden skal skrives igen og igen...

Den datalogiske løsning

- Analysér problemområdet
- Design et domæne-specifikt højniveau sprog
- Lav en oversætter, der genererer JavaScript- og Java-koden fra højniveau specifikationer

Sproget *PowerForms* er udviklet efter denne metode
Input-felter beskrives med *regulære udtryk*, der oversættes til *endelige automater*

Plan

Introduktion

Regulære udtryk

Induktionsbevis

Regulære automater

Skelnelighed og produktkonstruktion

dRegAut pakken

Automater til modellering og verifikation

Grundliggende begreber

Vi starter med nogle *matematiske definitioner*

- Et *alfabet* er en endelig mængde af tegn
 - Ex. $\{a, b, c, \dots z\}$
 - Ex. ASCII, Unicode
 - Ex. $\{0, 1\}$
- En *streng* er en *endelig* sekvens af tegn fra alfabetet
 - Ex. "onkel sune drejer den usle kno"
 - Ex. "10110"
 - Ex. " " (Den tomme streng. Skrives også Λ (andre steder ε)).
- Et *sprog* er en mængde af strenge
 - Ex. {"hans", "ole"}
 - Ex. $\{\Lambda, a, aa, aaa, aaaa, \dots\}$
 - Ex. $\{\}$ (Det tomme sprog)
 - Ex. Alle korrekte danske sætninger

Regulære udtryk

Et *regulært udtryk* beskriver et *sprog*

- Regulære udtryk findes på 6 former
- **3 basis-tilfælde:**
 - \emptyset – den tomme mængde af strenge
 - Λ – mængden bestående af den tomme streng
 - $a \in \Sigma$ – mængden bestående af en enkelt streng, som er det ene tegn a fra alfabetet Σ
- **Og 3 Sammensatte tilfælde (rekursive tilfælde):**
 - $r_1 + r_2$ – de strenge der beskrives af r_1 eller r_2
 - $r_1 r_2$ – de strenge der kan opdeles i to dele, så venstre del beskrives af r_1 og højre del af r_2
 - r^* – de strenge der kan opdeles i et antal dele, der hver beskrives af r

Eksempler på regulære udtryk

- Streng over alfabetet $\{0, 1\}$ med et *lige* antal tegn:
 $(00 + 11 + 01 + 10)^*$ eller $((0 + 1)(0 + 1))^*$
- Streng over alfabetet $\{0, 1\}$ med et *ulige* antal 1'er:
 $0^*1(0^*10^*1)^*0^*$ eller $0^*10^*(10^*10^*)^*$
- Gyldige datoer, telefonnumre, CPR-numre, emailadresser, URL'er,
...

Et mere realistisk eksempel

Floating point tal i Pascal

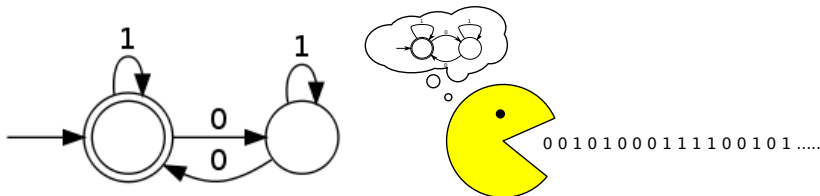
- Eksempler på gyldige strenge: `"3.14"` `"5.6E13"` `"-42.0"`
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ., +, -, E\}$
- Forkortelser:
 - $d = 0 + 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9$
 - $r^+ = rr^*$ (mindst én af r)
 - $s = \Lambda + + + -$
- Samlet udtryk: $sd^+(\cdot d^+ + \cdot d^+ E s d^+ + E s d^+ + \Lambda)$

Genkendelse af strenge

Givet en streng x , og et regulært udtryk r , hvordan ved vi om r matcher x

- Den naive metode: vi prøver os frem:
 - \emptyset matcher intet
 - Λ matcher kun den tomme streng.
 - $a \in \Sigma$ matcher kun strengen bestående af den ene karakter **a**
 - $r_1 + r_2$ matcher hvis r_1 matcher, eller r_2 matcher
 - $r_1 \cdot r_2$ Opdel x så $x = x_1 \cdot x_2$ på alle mulige måder Og prøv om r_1 matcher x_1 og r_2 matcher x_2
 - r_1^* Opdel x så $x = x_1 \cdot x_2 \cdot \dots \cdot x_n$ på alle mulige måder. Og se om r_1 matcher alle x_i for alle $i = 1, \dots, n$
- Det virker! Men det er håbløst ineffektivt.

Endelige Automater (forsmag)



- En endelig automat der genkender strenge over alfabetet $\Sigma = \{0, 1\}$ med et *lige* antal 0'er
- Automaten starter i den tilstand der er markeret med pilen
- Den "spiser" et tegn af gangen af strengen fra venstre mod højre
- Hvis den ender i tilstanden med dobbelt-cirkel, så accepterer den

Kleenes sætning

“Regulære udtryk og endelige automater har samme udtrykskraft”

- *Konstruktivt* bevis:
- For ethvert regulært udtryk findes en ækvivalent endelig automat
- For enhver endelig automat findes et ækvivalent regulært udtryk

Powerforms eksempel

- Lad R være et regulært udtryk, der svarer til gyldige datoer på form `dd/mm-åååå`
- Oversæt R til en ækvivalent endelig automat F
- Repræsenter F som et JavaScript-program, der kan svare på om en streng x er:
 - **accepteret**
 - **ikke accepteret**, men der er en sti til accept
 - **ikke accepteret** og ingen sti til accept

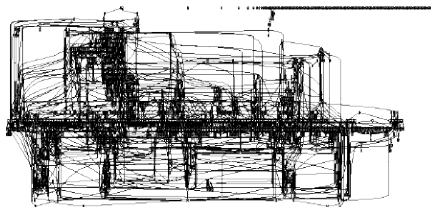
<http://www.brics.dk/bigwig/powerforms/examples/date.html>

Endelige automater til modellering af systemer

- Endelige automater er også nyttige uden regulære udtryk
- Endelige automater kan modellere systemer og egenskaber
- De teoretiske resultater om endelige automater kan bruges til at kombinere modeller og verificere om et givet system har en given egenskab

En endelig automat, der modellerer en togsimulator (fra VisualSTATE)

- 1421 del-automater
- 11102 transitioner
- 2981 inputs
- 2667 outputs
- 3204 lokale tilstande
- Antal tilstande ialt:
 10^{476}



Virker toget?

Beregnelighed



- Input og Output er strenge
- Program er en algoritme som kører på en maskine
- Eksempel: Givet et naturligt tal N i binær repræsentation som Input, beregn N^2 som output.

Beslutningsproblemer



- Hvis vi ignorerer effektivitet, så kan alle beregningsproblemer omformuleres til beslutningsproblemer.
- Eksempel: Givet to naturlige tal N, M i binær repræsentation som Input, svar ja hvis og kun hvis $N^2 = M$

Beslutningsproblemer som sprog

- Ethvert beslutningsproblem er et sprog (en mængde af strenge)
- $L = \{x | P(X) = \text{ja}\}$
- Ethvert sprog L er også et beslutningsproblem:

$$P(x) = \begin{cases} \text{ja} & \text{hvis } x \in L \\ \text{nej} & \text{ellers} \end{cases}$$

Eksempler på beslutningsproblemer

Givet en streng x ,

- er den en gyldig dato på form dd/mm-åååå?
- er den et syntaktisk korrekt Java program?
- er den et primtal?
- er den en konfiguration i skak hvor det er muligt for hvid at vinde?
- er den et semantisk korrekt Java-program?
- er den en syntaktisk korrekt sætning i dansk?
- er den en litterær klassiker?
- — vi vil kun se på formelle sprog og veldefinerede problemer

Endelige automater som model for beregnelighed

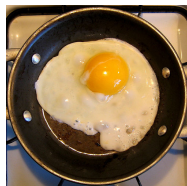
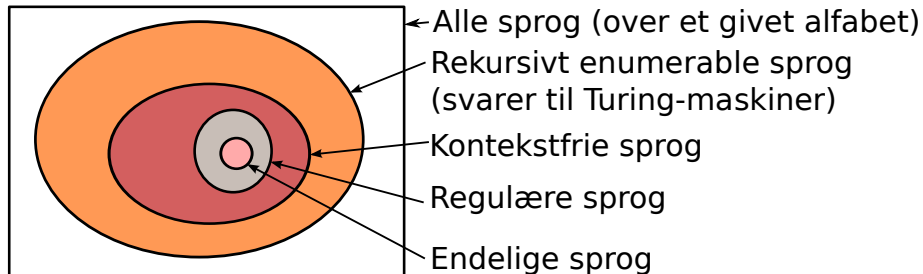
Vi vil studere følgende emne:

- Hvilke problemer kan afgøres af en maskine med endeligt meget hukommelse?
- Med andre ord: Hvilke sprog kan genkendes af endelige automater?

Mere generelle modeller for beregnelighed

- Pushdown-automater:
 - endelige automater med adgang til en vilkårligt stor stak (last-in-first-out)
 - anvendes ofte i parsere i oversættere
 - svarer til kontekstfri grammatikker
- Turing-maskiner:
 - ligesom endelige automater med adgang til en uendeligt stor notesblok
 - kan udføre vilkårlige algoritmer (Church-Turing-tesen)
 - svarer til uindskrænkede grammatikker (hvor endelige automater svarer til regulære grammatikker)

Klasser af formelle sprog



Picture from <http://www.flickr.com/photos/33602849@N00/5894257>

Hvorfor så nøjes med regulære automater

- Klassen af regulære sprog har mange “pæne” egenskaber:
 - afgørlighed (f.eks, “givet en FA M, accepterer den nogen strenge overhovedet?”)
 - lukkethed (snit, forening, ...)
- Til sammenligning:
 - Ved Turing-maskiner er næsten alt uafgørligt (Rices sætning: “alt interessant vedrørende sproget for en Turing-maskine er uafgørligt”)
 - Pushdown-automater / kontekstfri grammatikker: en mellemting, både med hensyn til udtrykskraft og afgørlighedsegenskaber

Uafgørlighed

```
while (x $\neq$ 1) {  
    if (even(x))  
        x = x/2;  
    else  
        x = 3·x+1;  
}
```

- Terminerer dette program på alle input x ? Ja eller nej?
- Tilsyneladende ja, men ingen har endnu bevist det!
- Men vi kan bevise, at der ikke findes et program (=en Turing-maskine), der kan afgøre det generelle problem “givet et program P , terminerer P på alle input?”

Praktiske oplysninger om kurset

- Hjemmeside: <http://cs.au.dk/~stm/RegAut>
- Seminarer:
 - 27/8 2010 Fredag 9-16
 - 10/9 2010 Fredag 9-16
 - 1/10 2010 Fredag 9-16

Materiale



- John Martin Introduction to Languages and the Theory of Computation 3. udgave, McGraw-Hill, 2002 ISBN: 0071198547 eller 0072322004
- Opgaver på ugesedlerne

Aktivitetsniveau

- Forventet aktivitet per uge ~ 15 timer
- $6 \text{ uger} \cdot 15 \text{ timer/uge} = 90 \text{ timer}$
- Seminarer: 21 timer
- Mellem seminarer: 69
- Forventet hjemmearbejde ca. 11 timer per uge
- Det forventes at man:
 - Læser de relevante kapitler i bogen
 - Løser opgaverne
 - Laver programmeringsprojektet

Opgaver

- Teoretiske opgaver
 - udfordrer forståelsen af det gennemgåede materiale øvelse i typisk “datalogisk matematik”
- Programmeringsprojekt
 - (dRegAut Java-pakken) implementation af de gennemgåede algoritmer, der udledes af konstruktive beviser Øvelse i at implementere formelle specifikationer i Java.

Eksamen

- Mundtlig, ekstern censur, 13-skalaen 20 min. per person, uden forberedelsestid
- For at kunne indstilles til eksamen skal man have godkendt besvarelser af de obligatoriske opgaver

Alfabeter, strenge og sprog

- Et alfabet Σ er en endelig mængde (af tegn/symboler) eks.:
 $\Sigma = a, b, c$
- En streng x er en endelig sekvens af tegn fra alfabetet eks.:
 $x = abba \wedge$ repræsenterer den tomme streng (strengen af længde 0), $\wedge \notin \Sigma$
- Et sprog L er en (vilkårlig) mængde af strenge eks.:
 $L = \wedge, cab, abba$
- Σ^* er mængden af alle strenge over Σ dvs. $L \subseteq \Sigma^*$ hvis L er et sprog over Σ eks.: hvis $\Sigma = a, b, c$ så er
 $\Sigma^* = \wedge, a, b, c, aa, ab, ac, aaa, aab, \dots$

Konkatenering af strenge

- Hvis $x, y \in \Sigma^*$, så er $x \cdot y$ (konkateneringen af x og y) den streng, der fremkommer ved at sætte tegnene i x før tegnene i y
 - Eks.: hvis $x = abb$ og $y = a$, så er

$$x \cdot y = abba$$

$$y \cdot x = aabb$$

- Bemærk: $x \cdot \Lambda = \Lambda \cdot x = x$ for alle x
- $x \cdot y$ skrives ofte xy (uden “.”)

Konkatenering af sprog

- Hvis $L_1, L_2 \subseteq \Sigma^*$, så er $L_1 \cdot L_2$ (konkateneringen af L_1 og L_2) defineret ved

$$L_1 \cdot L_2 = \{x \cdot y \mid x \in L_1 \wedge y \in L_2\}$$

- Eks.: Hvis $\Sigma = \{0, 1, 2, a, b, c\}$ og $L_1 = \{\wedge, 10, 212\}$,
 $L_2 = \{cab, abba\}$ så er:

$$L_1 \cdot L_2 = \{cab, 10cab, 212cab, abba, 10abba, 212abba\}$$

- Bemærk:
 - $L \cdot \{\wedge\} = \{\wedge\} \cdot L = L$ for alle L
 - $L \cdot \emptyset = \emptyset \cdot L = \emptyset$ for alle L
 - $L_1 \cdot L_2$ skrives ofte $L_1 L_2$ (uden “.”)

Kleene stjerne

Kleene stjerne er en måde at udtrykke “0 eller flere forekomster”

- $L^k = \underbrace{LL \dots L}_{k \text{ gange}}$ (konkatenering af k forekomster af L)
- $L^0 = \{\Lambda\}$ (0 forekomster af L)
- $L^* = \bigcup_{i=0}^{\infty} L^i$ (**Kleene stjerne af L**)
- $L^+ = L^*L$ (1 eller flere forekomster)
- Eks.: Hvis $L = \{aa, b\}$ så er

$$L^* = \{\Lambda, aa, b, aaaa, aab, baa, bb, aaaaaa, \dots\}$$

Rekursive definitioner

- En definition er rekursiv, hvis den refererer til sig selv
- Eks.: Fibonacci $f : N \rightarrow N$

$$f(n) = \begin{cases} 1, & \text{hvis } n = 1 \vee n = 0 \\ f(n-1) + f(n-2), & \text{ellers} \end{cases}$$

- Enhver selv-reference skal referere til noget "mindre" og dermed føre til endeligt mange selv-referencer

Rekursiv definition af strenge

- x er en streng over alfabetet Σ , dvs. $x \in \Sigma^*$ hvis:
- $x = \Lambda$, eller
- $x = y \cdot a$ hvor $y \in \Sigma^*$ og $a \in \Sigma$
- (underforstået Σ^* er den mindste mængde der opfylder dette)
- Eksempel:

$$abc = (((\Lambda \cdot a) \cdot b) \cdot c) \in \Sigma^*, (\text{hvor } \Sigma = \{a, b, c, d\})$$

Syntax af regulære udtryk

Mængden R af regulære udtryk over Σ er den mindste mængde, der indeholder følgende:

- \emptyset
- \wedge
- a for hver $a \in \Sigma$
- $(r_1 + r_2)$ hvor $r_1, r_2 \in R$
- $(r_1 r_2)$ hvor $r_1, r_2 \in R$
- (r^*) hvor $r \in R$

Semantik af regulære udtryk

Sproget $L(r)$ for $r \in R$ er defineret rekursivt i strukturen af R

- $L(\emptyset) = \emptyset$
- $L(\wedge) = \{\wedge\}$
- $L(a) = \{a\}$
- $L((r_1 + r_2)) = L(r_1) \cup L(r_2)$
- $L((r_1 r_2)) = L(r_1)L(r_2)$
- $L((r^*)) = (L(r))^*$

Regulære sprog

- Definition: Et sprog S er regulært hvis og kun hvis der eksisterer et regulært udtryk r hvor $L(r) = S$

Paranteser i regulære udtryk

- Forening og konkatenering er **associative**, så vi vælger at tillade f.eks.
 - at $(a + (b + c))$ kan skrives $a + b + c$
 - at $(a(bc))$ kan skrives abc
- Vi definerer **præcedens** for operatorerne:
 - $*$ binder stærkest
 - konkatenering binder middel
 - $+$ binder svagest
 - Eks.: $(a + ((b^*)c))$ kan skrives $a + b^*c$

Eksempel

- Betragt følgende regulære udtryk r over alfabetet $\{0, 1\}$:

$$r = (1 + \wedge)001$$

- På grund af parentesreglerne er dette det samme som

$$r = (((((1 + \wedge)0)0)1)$$

- Så sproget for r er

$$L(r) = (((\{1\} \cup \{\wedge\})\{0\})\{0\})\{1\} = \{1001, 001\}$$

Quiz

- 1 Hvad betyder $\{a, bc\}^*$?
- 2 Hvad er betingelsen for at et sprog S er regulært?

Øvelser

- [Martin] Opg. 3.2
- [Martin] Opg. 3.9 (a-e)
- [Martin] Opg. 3.10 (a-b)

Plan

Introduktion

Regulære udtryk

Induktionsbevis

Regulære automater

Skelnelighed og produktkonstruktion

dRegAut pakken

Automater til modellering og verifikation

Reverse-operatoren

- Givet en streng $x \in \Sigma^*$, definer $reverse(x)$ rekursivt i strukturen af x :
- $reverse(\Lambda) = \Lambda$
- $reverse(ya) = a(reverse(y))$, hvor $y \in \Sigma^*$, $a \in \Sigma$
- Eksempel:
 $reverse(123) = 3 \cdot reverse(12) = \dots = 321 \cdot reverse(\Lambda) = 321$

Reverse på et sprog

- Givet et sprog $L \subseteq \Sigma^*$, definer

$$\text{reverse}(L) = \{\text{reverse}(x) \mid x \in L\}$$

- Eksempel: Hvis $L = \{\wedge, 123, abc\}$ så er

$$\text{reverse}(L) = \{\wedge, 321, cba\}$$

Rekursion og induktionsbeviser

- Rekursive definitioner giver ofte anledning til induktionsbeviser
- Hvis vi skal bevise noget på form “for alle X gælder $P(X)$ ”, hvor mængden af X 'er er defineret rekursivt, så kan vi prøve bevisteknikken “induktion i strukturen af X ”

Eksempel på et induktionsbevis (1/3)

- Påstand: Hvis S er et regulært sprog, så er $reverse(S)$ også regulært
(dvs. de regulære sprog er lukkede under Reverse)
- Bevis:
S er regulært, så der eksisterer et regulært udtryk r så $L(r) = S$
Vi vil vise ved **induktion** i strukturen af r , at der eksisterer et regulært udtryk r' hvor $L(r') = reverse(L(r))$, hvilket medfører, at $reverse(S)$ er regulært

Eksempel på et induktionsbevis (2/3)

Basis

- $r = \emptyset: r' = \emptyset$

$$L(\emptyset) = \emptyset = \textit{reverse}(\emptyset) = \textit{reverse}(L(\emptyset))$$

- $r = \Lambda: r' = \Lambda$

...

- $r = a: r' = a$

...

Eksempel på et induktionsbevis (3/3)

Induktionsskridtet

For alle deludtryk s af r kan vi udnytte **induktionshypotesen**:

Der eksisterer et regulært udtryk s' hvor $L(s') = \text{Reverse}(L(s))$

- $r = r_1 + r_2$ hvor $r_1, r_2 \in R$: vælg $r' = r'_1 + r'_2$ hvor r'_1 of r'_2 er givet i induktionshypotesen.
- $r = r_1 r_2$ hvor $r_1, r_2 \in R$: vælg $r' = r'_2 r'_1$
- $r = r_1^*$ hvor $r_1 \in R$: vælg $r' = (r'_1)^*$
- Lemma 1: $\forall x, y \in \Sigma^*: \text{reverse}(xy) = \text{reverse}(y)\text{reverse}(x)$
Bevis: induktion i strukturen (eller længden) af y
- Lemma 2: $\forall i \geq 0, E \subseteq \Sigma^*: \text{reverse}(E^i) = (\text{reverse}(E))^i$
Bevis: induktion i i

Konstruktive beviser

- Bemærk at dette induktionsbevis implicit indeholder en algoritme til – givet et regulært udtryk for S – at konstruere et regulært udtryk for $reverse(S)$
- Sådanne beviser kaldes konstruktive
- Husk altid både *konstruktionen* **og** *beviset for dens korrekthed*

Algoritmen

- Input: et regulært udtryk r
- Definer en rekursiv funktion REV ved:
 - $REV(\emptyset) = \emptyset$
 - $REV(\wedge) = \wedge$
 - $REV(a) = a$, hvor $a \in \Sigma$
 - $REV(r_1 + r_2) = REV(r_1) + REV(r_2)$
 - $REV(r_1 r_2) = REV(r_2) \cdot REV(r_1)$
 - $REV(r_1^*) = (REV(r_1))^*$
- Output: det regulære udtryk $REV(r)$

Øvelse

- Lad r være det regulære udtryk $((a + \wedge)cbc)^*$ over alfabetet $\{a, b, c\}$.
 - Bevis at enhver streng i sproget $L(r)$ har et lige antal c 'er.
 - Argumentér kort og præcist for hvert trin i beviset.
- Hint: Brug definitionen af sprog for regulære udtryk (Definition 3.1 i [Martin]), definitionen af $*$ på sprog (s. 31 øverst i [Martin]), og lav induktion.

Løsninger

[Martin] 3.2

- a) 00
- b) 01
- c) 0
- d) 010

Løsninger

[Martin] 3.9

- a) $1^*01^*01^*$
- b) $(0 + 1)^*0(0 + 1)^*0(0 + 1)^*$
- c) $\Lambda + 0 + 1 + (0 + 1)^*(00 + 10 + 11)$
- d) $(00 + 11)(0 + 1)^* + (0 + 1)^*(00 + 11)$
- e) $(1 + 01)^*(0 + \Lambda)$

Løsninger

[Martin] 3.10

- a) The language of all strings containing an odd number of 1's
- b) The language of all strings containing $3n$ or $3n + 1$ characters for any natural number n

Plan

Introduktion

Regulære udtryk

Induktionsbevis

Regulære automater

Skelnelighed og produktkonstruktion

dRegAut pakken

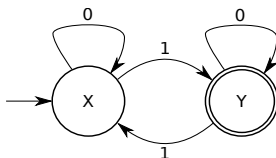
Automater til modellering og verifikation

Regulære udtryk vs endelige automater

- Regulære udtryk: deklarative
 - dvs. ofte velegnede til at specificere regulære sprog
- Endelige automater: operationelle
 - dvs. bedre egnet til at afgøre om en given streng er i sproget
- Ethvert regulært udtryk kan oversættes til en endelig automat – og omvendt
 - (bevises næste seminar...)

En endelig automat

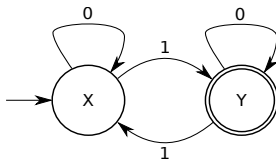
- En endelig automat, der genkender strenge over alfabetet $\Sigma = \{0, 1\}$ med ulige antal 1'er:



- Automaten læser strengen ét tegn ad gangen, fra venstre mod højre
- Hvis automaten ender i en accept-tilstand, så accepteres(=genkendes) strengen

At køre en streng på en automat

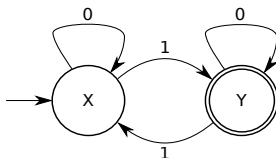
- Eksempel: vi vil vide om strengen 1010 accepteres



- Vi starter i starttilstanden og læser strengen ét tegn ad gangen
- Vi ender i en ikke-accept tilstand, så strengen accepteres ikke

Hvad repræsenterer tilstandende

- Hver tilstand repræsenterer en viden om den hidtil læste delstreng
- Eksempel:



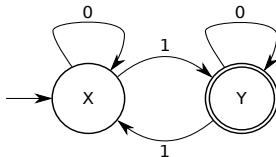
- X: “der er læst et lige antal 1’er”
- Y: “der er læst et ulige antal 1’er”

Formel definition af endelige automater

- En endelig automat (finite automaton/FA) er et 5-tupel $(Q, \Sigma, q_0, A, \delta)$ hvor
- Q er en endelig mængde af tilstande
- Σ er et alfabet
- $q_0 \in Q$ er en starttilstand
- $A \subseteq Q$ er accepttilstandene
- $\delta : Q \times \Sigma \rightarrow Q$ er en transitionsfunktion

Eksempel

- Denne grafiske repræsentation af en automat:



- svarer til 5-tuplet $(Q, \Sigma, q_0, A, \delta)$ hvor
- $Q = \{X, Y\}$
- $\Sigma = \{0, 1\}$
- $q_0 = X$
- $A = \{Y\}$
- $\delta : Q \times \Sigma \rightarrow Q$ er denne funktion:

	0	1
X	X	Y
Y	Y	X

Hvorfor en formel definition

- Den formelle definition viser kort og præcist hvad en FA er
- For eksempel,
 - en FA har endeligt mange tilstande
 - den har præcis én starttilstand
 - en vilkårlig delmængde af tilstandene kan være accepttilstande
 - for enhver tilstand q og alfabetsymbol a er der én udgående transition (til tilstanden $\delta(q, a)$)
 - der er ikke noget krav om, at alle tilstande kan nås fra starttilstanden

Sproget af en automat

- 5-tupel-definitionen fortæller hvad en FA er
- Vi vil nu definere hvad en FA kan:
- En FA accepterer en streng, hvis dens kørsel fra starttilstanden ender i en accepttilstand
- Sproget $L(M)$ af en FA M er mængden af strenge, den accepterer
- M genkender sproget $L(M)$

Formel definition af $L(M)$

- Givet en FA $M = (Q, \Sigma, q_0, A, \delta)$, definer den udvidede transitionsfunktion $\delta^* : Q \times \Sigma^* \rightarrow Q$ ved

$$\delta^*(q, x) = \begin{cases} q & \text{hvis } x = \Lambda \\ \delta(\delta^*(q, y), a) & \text{hvis } x = ya \text{ hvor } y \in \Sigma^* \text{ og } a \in \Sigma \end{cases}$$

- $x \in \Sigma^*$ accepteres af M hvis og kun hvis $\delta^*(q_0, x) \in A$
- Definer $L(M) = \{x \in \Sigma^* | x \text{ accepteres af } M\}$

Quiz

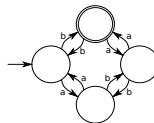
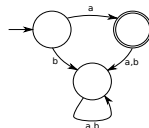
Konstruer en FA så

$$L(M) = \Sigma^*$$

$$L(M) = \emptyset$$

$$L(M) = \{a\}$$

$$L(M) = \{x \in \Sigma^* \mid n_a(x) \text{ lige og } n_b(x) \text{ ulige}\}$$



Øvelser

- [Martin] Opg. 3.17 (e)
- [Martin] Opg. 3.18
- [Martin] Opg. 3.19 (a-c)

Plan

Introduktion

Regulære udtryk

Induktionsbevis

Regulære automater

Skelnelighed og produktkonstruktion

dRegAut pakken

Automater til modellering og verifikation

Skelnelighed

- Givet et sprog L , hvor mange tilstande er nødvendige i en FA M hvis $L(M) = L$?
- To strenge, x og y , skal ende i forskellige tilstande, hvis der er behov for at kunne **skelne** dem:
- dvs., $\delta^*(q_0, x) \neq \delta^*(q_0, y)$ hvis
$$\exists z \in \Sigma^* : (xz \in L \wedge yz \notin L) \vee (xz \notin L \wedge yz \in L)$$

Definition af skelnelighed

- Lad $L \subseteq \Sigma^*$ og $x, y \in \Sigma^*$
- **Kvotientsproget** L/x defineres som

$$L/x = \{z \in \Sigma^* | xz \in L\}$$

- x og y **er skelnelige** mht. L hvis

$$L/x \neq L/y$$

- z **skelner** x og y mht. L hvis

$$z \in L/x - L/y \text{ eller } z \in L/y - L/x$$

Eksempel

- Hvis
 - $L = \{s \in \{0, 1\}^* \mid s \text{ ender med } 10\}$
 - $x = 00$
 - $y = 01$
- så er x og y skelnelige mht. L
- Bevis: $z = 0$ skelner x og y
- Heraf kan vi se, at hvis $M = (Q, \Sigma, q_0, A, \delta)$ genkender L så er

$$\delta^*(q_0, x) \neq \delta^*(q_0, y)$$

Uanset hvordan M ellers er opbygget.

Nødvendigt antal tilstande i en FA

- Antag $x_1, x_2, \dots, x_n \in \Sigma^*$
og for ethvert par $x_i, x_j, i \neq j$ er x_i og x_j skelnelige mht. L
- Enhver FA der genkender L har mindst n tilstande
- Bevis (skitse):
 - Antag FA'en har færre tilstande
 - Det medfører at $\exists i \neq j : \delta^*(q_0, x_i) = \delta^*(q_0, x_j)$ (skuffeprincippet)
 - Det er i modstrid med at x_i og x_j var skelnelige mht. L



<http://www.flickr.com/photos/curiousexpeditions/2194711073/>

Eksempel 1: En stor automat

- Dette eksempel kan give intuition for begrebet skelnelighed
- Lad $L_{42} = \{x \in \{0, 1\}^* \mid |x| \geq 42 \text{ og det 42. symbol fra højre i } x \text{ er et } 1\}$
- Lad $x_1, x_2, \dots, x_{2^{42}}$ være alle strenge af længde 42 over alfabetet $\{0, 1\}$
- Disse strenge er alle parvist skelnelige mht. L_{42}
- En automat der genkender L_{42} har derfor mindst 2^{42} tilstande
- (...hvis den overhovedet findes)
- Bevis: $x \neq y$ må være forskellige i i 'te tegn fra venstre. Strengen z som skelner kan være 0^{i-1}

Eksempel 2: Palindromer

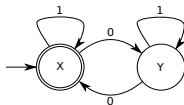
- Lad $pal = \{x \in \{0, 1\}^* \mid x = reverse(x)\}$
- Lad x og y være vilkårlige forskellige strenge over $\{0, 1\}$
- x og y er skelnelige mht. pal (bevis: se bogen Theorem 3.3 side 109)
- Vi kan altså finde en vilkårligt stor mængde parvist skelnelige strenge, så pal er ikke regulært.

Forening af regulære sprog

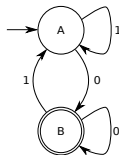
- Givet to regulære sprog, L_1 og L_2 er $L_1 \cup L_2$ også regulært?
- Ja! (dvs. klassen af regulære sprog er lukket under forening)

Eksempel

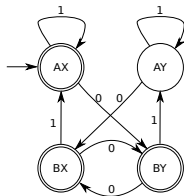
M1: (strengene med lige antal 0'er)



M2: (strengene der ender med 0)



$$L(M) = L(M1) \cup L(M2)$$



Produktkonstruktionen

- Antag vi har to FA'er:
- $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$
- $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$
- Definér en ny FA: $M = (Q, \Sigma, q_0, A, \delta)$ hvor
 - $Q = Q_1 \times Q_2$ (produktmængden af tilstande)
 - $q_0 = (q_1, q_2)$
 - $A = \{(p, q) | p \in A_1 \vee q \in A_2\}$
 - $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$
- Der gælder nu:

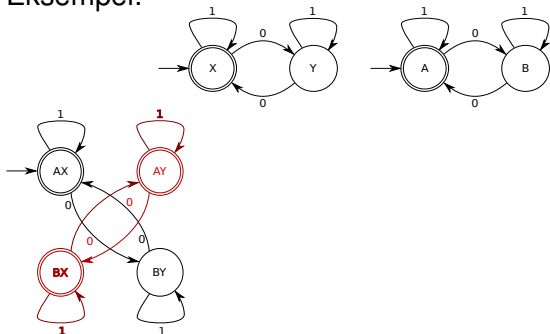
$$L(M) = L(M_1) \cup L(M_2)$$

Konstruktivt bevis for korrekthed

- Lemma: $\forall x \in \Sigma^* : \delta^*((p, q), x) = (\delta_1^*(p, x), \delta_2^*(q, x))$
- (Bevis: opgave 3.32, induktion i x)
 - Brug lemmaet samt definitionerne af M og $L(\bullet)$

Nøjes med opnåelige tilstande

- Nøjes med opnåelige tilstande
- Produktkonstruktionen bruger $Q = Q_1 \times Q_2$
- I praksis er hele tilstandsrummet sjældent nødvendigt
- Eksempel:



- Kun tilstande, der er opnåelige fra starttilstanden er relevante for sproget!

Snitmængde og differens

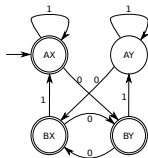
- Givet to regulære sprog, L_1 og L_2
 - er $L_1 \cap L_2$ også regulært?
 - er $L_1 - L_2$ også regulært?
- Ja! (dvs. klassen af regulære sprog er lukket under snit og differens)
- Bevis: produktkonstruktion som ved \cup men
 - for \cap , vælg $A = (p, q) | p \in A_1 \wedge q \in A_2$
 - for $-$, vælg $A = (p, q) | p \in A_1 \wedge q \notin A_2$

Komplement

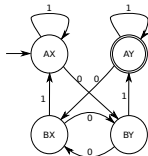
- Givet et regulære sprog R er R' (R s komplement) også regulært?
- Ja! (dvs. klassen af regulære sprog er lukket under komplement)
- Bevis 1:
 - Vælg $L_1 = \Sigma^*$ og $L_2 = R$, hvorved $R' = L_1 - L_2$
- Bevis 2:
 - Givet en FA $M = (Q, \Sigma, q_0, A, \delta)$ hvor $L(M) = R$
 - Definer $M' = (Q, \Sigma, q_0, Q - A, \delta)$
 - Derved gælder at $L(M') = R'$

Eksempel

- M : (Strengene der enten har et lige antal 0'er, **eller** slutter med 0)



- M' : (Strengene der både har et **ulige** antal 0'er, **og ikke** slutter med 0)



Øvelse

- [Martin] 3.33 (a-c)

Plan

Introduktion

Regulære udtryk

Induktionsbevis

Regulære automater

Skelnelighed og produktkonstruktion

dRegAut pakken

Automater til modellering og verifikation

dRegExp

- Java-repræsentation af regulære udtryk
- Speciel syntax:
 - # betyder \emptyset
 - % betyder \wedge
- Alfabetet angives som en mængde af Unicode tegn
- Eks.: $((a + \wedge)cbc)^*$ skrives som `"((a+%))cbc)*"`

dRegAut pakken

Udleverede programdele:

- FA.java:
 - repræsentation af FA'er
- Alphabet.java, State.java, StateSymbolPair.java, AutomatonNotWellDefinedException.java:
hjelpeklasser til FA.java

FA.java

```
public class FA {  
    public Set<State> states;           //  $Q$   
    public Alphabet alphabet;          //  $\Sigma$   
    public State initial;               //  $q_0 \in Q$   
    public Set<State> accept;           //  $A \subseteq Q$   
    public Map<StateSymbolPair, State>  
        transitions;                    //  $\delta: Q \times \Sigma \rightarrow Q$   
    ...  
}
```

- et `Alphabet` objekt indeholder mængde af `Character` objekter
- et `StateSymbolPair` objekt består af et `State` objekt og et `Character` objekt

Nyttige metoder i FA.java

- `FA()` — konstruerer uinitialiseret FA objekt
- `FA(Alphabet a)` — konstruerer FA for det tomme sprog
- `clone()` — kloner et FA objekt
- `checkWellDefined()` — undersøger om FA objektet repræsenterer en veldefineret FA
- `getNumberOfStates()` — returnerer størrelsen af states
- `setTransition(State q, char c, State p)` — tilføjer en c transition fra q til p
- `toDot()` — konverterer FA objekt til 'Graphviz dot' input (til grafisk repræsentation)

Java projektet (1. del)

- Studér udleverede programdele:
 - repræsentation af FA'er
 - ekstra udleverede metoder: delta, deltaStar, complement
- Implementér FA metoder:
 - accepts, intersection, union, minus
 - Opbyg en FA og vis den grafisk

Plan

Introduktion

Regulære udtryk

Induktionsbevis

Regulære automater

Skelnelighed og produktkonstruktion

dRegAut pakken

Automater til modellering og verifikation

Eksempel

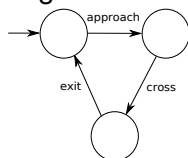
En jernbaneoverskæring

- Tre komponenter:
 - et **tog**
 - krydser vejen
 - kommunikerer med kontrolsystemet
 - et **kontrolsystem**
 - styrer bommen
 - en **bom**
- Sikkerhedsegenskab: bommen er altid nede, når toget krydser vejen

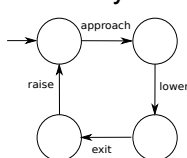


Modellering af systemet

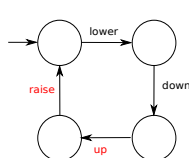
Tog



Kontrolsystem



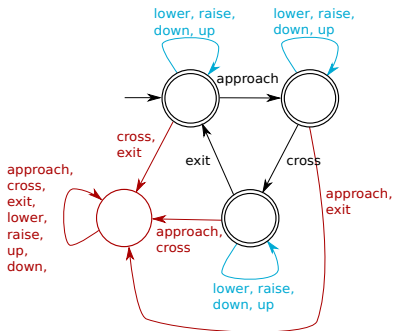
Bom



- Begivenheder (alfabet):
 - **approach**: toget nærmer sig
 - **cross**: toget krydser vejen
 - **exit**: toget forlader området
 - **lower**: besked til bommen om at gå ned
 - **raise**: besked til bommen om at gå op
 - **down**: bommen går ned
 - **up**: bommen går op

Modellering som FA

Eksempel:



- definer accepttilstande
- tilføj loop-transitioner så komponenterne får samme alfabet
- tilføj crash-tilstand og ekstra transitioner så transitionsfunktionen bliver total

Kombination af elementerne

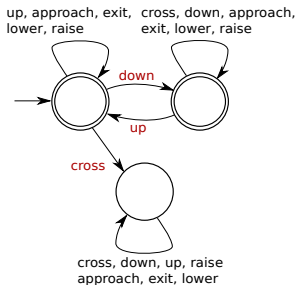
- Vi er interesseret i de sekvenser af begivenheder, der opfylder alle komponenterne
- Produktkonstruktion:

$$L(M) = L(M_{TOG}) \cap L(M_{KONTROL}) \cap L(M_{BOM})$$

Modellering af sikkerhedsegenskaben

“Bommen er altid nede når toget krydser vejen”

- S:



Verifikation

- Korrekthed: $L(M) \cap (L(S))' = \emptyset$
- dvs. vi skal bruge
 - produktkonstruktion (igen)
 - komplement
 - algoritme til at afgøre om sproget for en given FA er tomt (3. seminar)
- hvis $L(M) \cap (L(S))' \neq \emptyset$: enhver streng i $L(M) \cap (L(S))'$ svarer til et **modeksempel** (algoritme: 3. seminar)

Verifikation med dRegAut-pakken

- Opbyg FA-objekter svarende til M_{TOG} , $M_{KONTROL}$, M_{BOM} , og S
- Kombiner med `FA.intersection()` og `FA.complement()`
- Brug `FA.isEmpty()` og `FA.getAShortestExample()`
- Resultat:
modeksempel:
approach · lower · down · up · cross

Resume

- Definition af **endelige automater** og deres sprog
- **Skelnelighed**, hvad repræsenterer tilstandene, nødvendigt antal tilstande
- **Produktkonstruktionen**, komplement (*konstruktive beviser*)
- `dRegAut.FA` klassen, Java-repræsentation af FA'er
- Eksempel: automater til modellering og verifikation