

**EVU - RegAut 2009**

# **Regularitet og Automater**

**Sigurd Meldgaard**  
**(Slides efter Søren Besenbacher)**

# Plan

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

# Regularitet og Automater

Formål med kurset:

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

# Eksempel

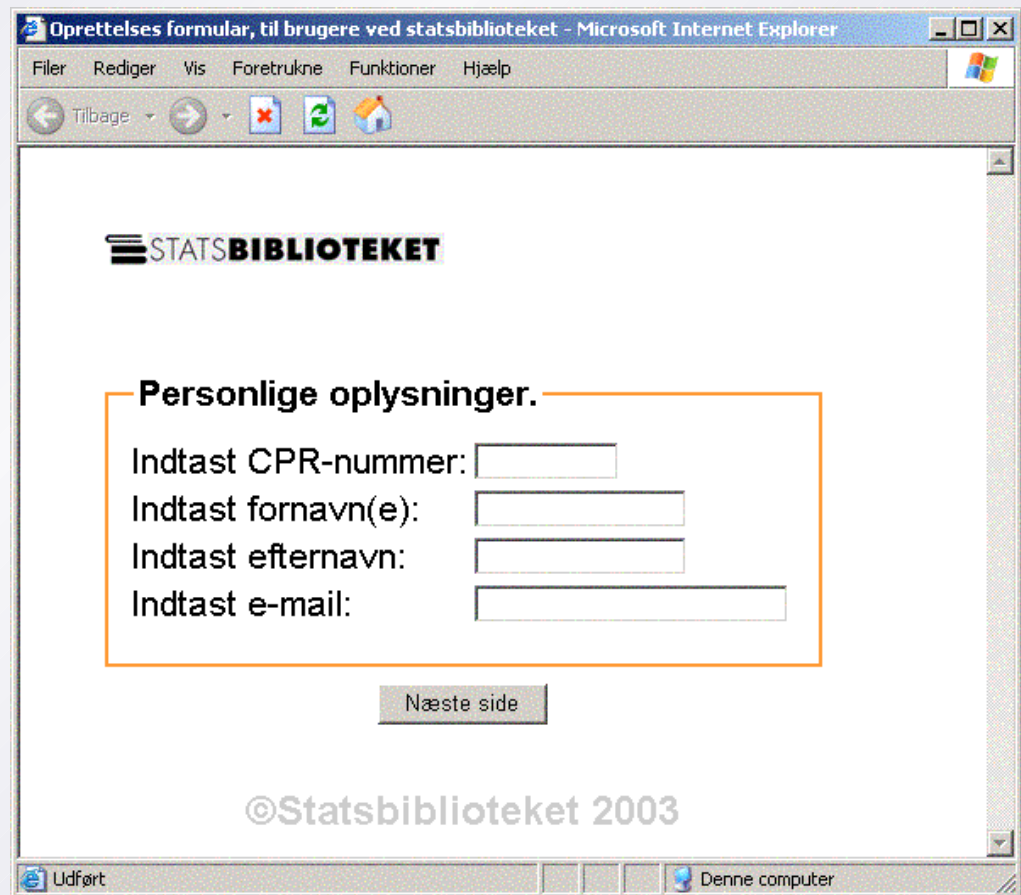


# HTML formularer

HTML formularer indeholder *input-felter*, hvor brugeren kan indtaste tekststreng

Eks.:

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



The screenshot shows a Microsoft Internet Explorer window with the title 'Oprettelses formular, til brugere ved statsbiblioteket'. The browser's address bar and menu bar are visible. The main content area displays the 'STATSBIBLIOTEKET' logo and a section titled 'Personlige oplysninger.' enclosed in an orange border. This section contains four input fields: 'Indtast CPR-nummer:', 'Indtast fornavn(e):', 'Indtast efternavn:', and 'Indtast e-mail:'. Below the form is a button labeled 'Næste side'. At the bottom of the page, the copyright notice '©Statsbiblioteket 2003' is visible. The browser's status bar at the bottom shows 'Udført' and 'Denne computer'.

# 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

1. Analysér problemområdet
  2. Design et domæne-specifikt højniveau sprog
  3. 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**



# Regulære udtryk

- Et **alfabet** er en endelig mængde af tegn
- En **streng** er en endelig sekvens af tegn fra alfabetet
- Et **sprog** er en mængde af strenge
- Et **regulært udtryk** beskriver et sprog:
  - $\emptyset$  – den tomme mængde af strenge
  - $\square$  – mængden bestående af den tomme streng
  - $a \in \square$  – mængden bestående af en enkelt streng, som er det ene tegn  $a$  fra alfabetet  $\square$
  - $r_1 + r_2$  – 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

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

# Et mere realistisk eksempel

## Floating-point tal (i Pascal):

- alfabet:  $\Sigma = \{\underline{0}, \underline{1}, \underline{2}, \underline{3}, \underline{4}, \underline{5}, \underline{6}, \underline{7}, \underline{8}, \underline{9}, \underline{+}, \underline{-}, \underline{.}, \underline{E}\}$
- eksempler på gyldige strenge:

3.14

5.6E13

-42.0

- forkortelser:

- $d \sqsubseteq \underline{0} + \underline{1} + \underline{2} + \underline{3} + \underline{4} + \underline{5} + \underline{6} + \underline{7} + \underline{8} + \underline{9}$

- $d^+ \sqsubseteq d^*d$

- $s \sqsubseteq \sqsubseteq + \underline{+} + \underline{-}$

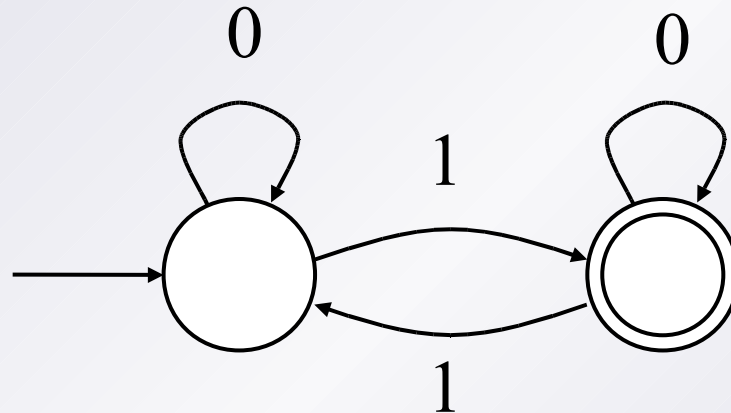
- samlet udtryk:  $sd^+(\underline{.}d^+ + \underline{.}d^+\underline{E}sd^+ + \underline{E}sd^+)$

# Genkendelse af strenge

- Givet et regulært udtryk  $r$  og en streng  $x$ , hvordan undersøger vi om  $x$  matcher  $r$  ?
- Den naive løsning:  
Vi prøver os frem, afhængigt af strukturen af  $r$ 
  - $\emptyset$  – matcher intet
  - $\square$  – matcher kun den tomme streng
  - $a \in \square$  – matcher kun strengen  $a$
  - $r_1 + r_2$  – matcher  $x$  hvis  $r_1$  eller  $r_2$  matcher  $x$
  - $r_1 r_2$  – opdel  $x$  så  $x = x_1 x_2$  på alle mulige måder, check om der eksisterer en opdeling så  $r_1$  matcher  $x_1$  og  $r_2$  matcher  $x_2$
  - $r^*$  – opdel  $x$  så  $x = x_1 x_2 \dots x_n$  på alle mulige måder, check om der eksisterer en opdeling så  $r$  matcher  $x_i$  for alle  $i = 1 \dots n$
- Det virker! – men er håbløst ineffektivt...

# Endelige automater

- 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
- Hver tilstand repræsenterer en viden om den læste delstreng
- Hvis automaten ender i en *accept*-tilstand, så accepteres(=genkendes) strengen

# Kleenes sætning

- “*Regulære udtryk og endelige automater har samme udtrykskraft*”
- **Konstruktive** beviser:
  - ethvert *regulært udtryk* kan oversættes til en ækvivalent *endelig automat*
  - enhver *endelig automat* kan oversættes til 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

# PowerForms eksempel

29/02 | 

29/02@ | 

29/02-1996 | 



# 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

# Verifikasjon

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

1421 del-automater

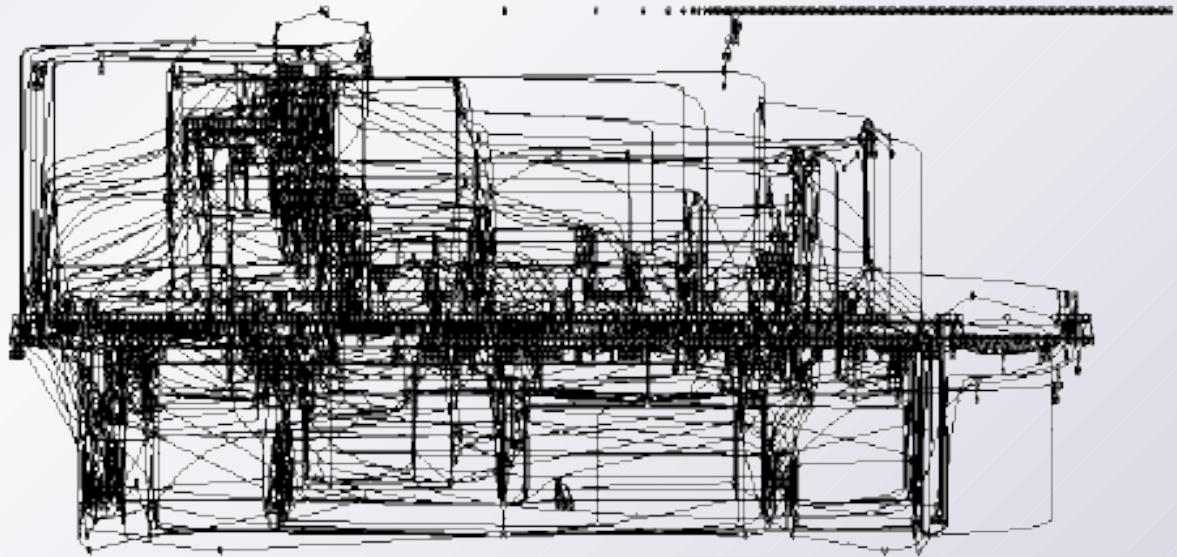
11102 transitioner

2981 inputs

2667 outputs

3204 lokale tilstande

ialt antal tilstande: 10<sup>476</sup>



Virker toget?

# Beregnelighed



- Input og Output er strenge
- PROGRAM er en algoritme, der kører på en maskine
  - Eks.: Givet et naturligt tal  $N$  som input, maskinen beregner  $N^2$  som output

# Beslutningsproblemer



- Hvis vi ignorerer effektivitet, så kan ethvert beregningsproblem omformuleres som et **beslutningsproblem**
  - Eks.: Givet to naturlige tal  $N$  og  $M$  som input, maskinen svarer "ja" hvis og kun hvis  $M=N^2$

# Beslutningsproblemer og sprog

- Ethvert beslutningsproblem  $P$  er et sprog:  
 $L = \{ \text{de strenge } x, \text{ hvor svaret på } P \text{ er "ja"} \}$
- Ethvert sprog  $L$  er et beslutningsproblem:  
 $P$ : “er input  $x$  i  $L$ ?”

# Eksempler på beslutningsproblemer

Givet en streng,

- 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

- Turing-maskiner:
  - 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*)
- 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*



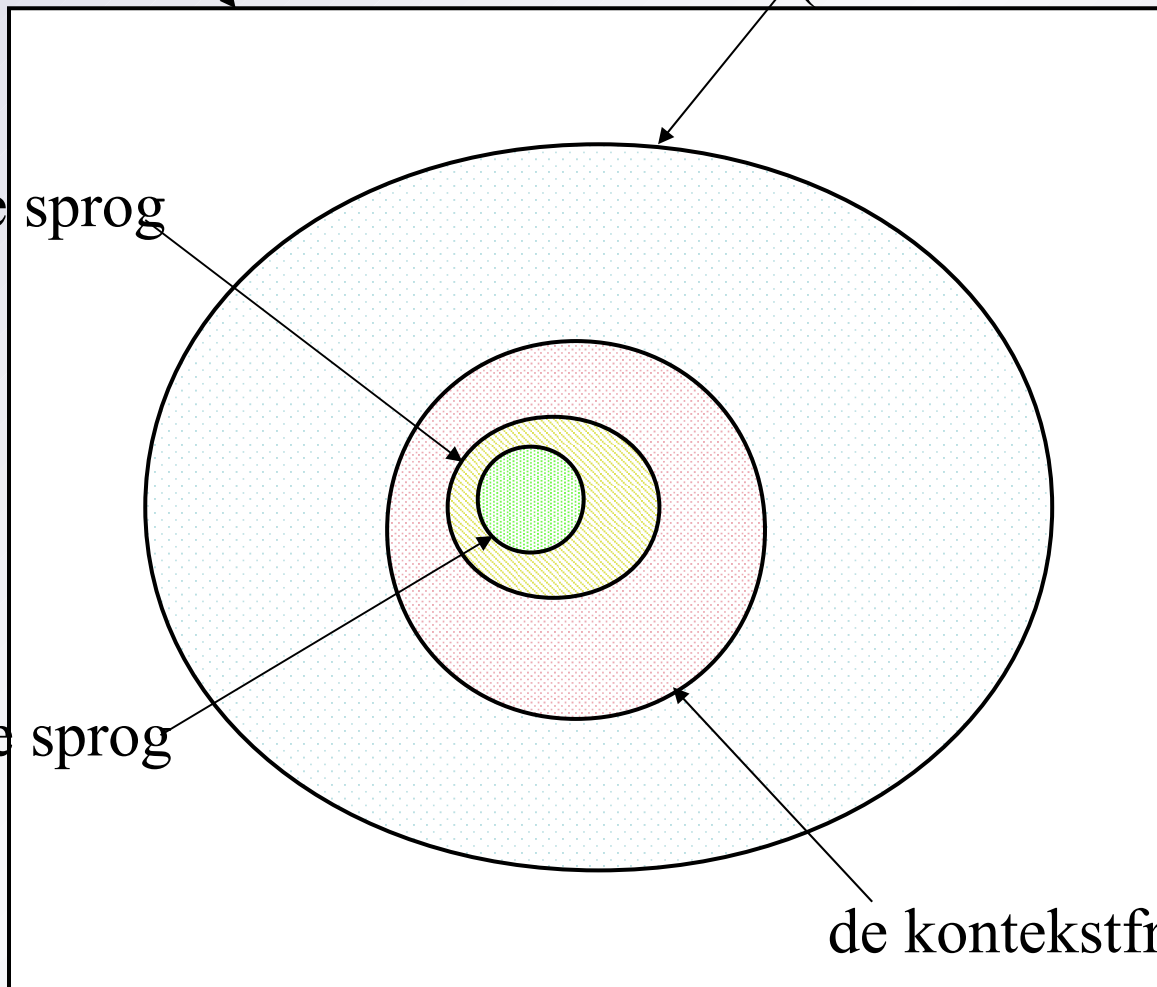
# Klasser af formelle sprog

klassen af alle sprog (over et givet alfabet) de *rekursivt numerable* sprog  
(svarer til Turing-maskiner)

de regulære sprog

de endelige sprog

de kontekstfri sprog



# Hvorfor nøjes med endelige 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 udtrykskraft og afgørlighedsegenskaber

# Uafgørlighed

Her er et lille program:

```
while (x≠1) {  
    if (even(x))  
        x = x/2;  
    else  
        x = 3*x+1;  
}
```

Terminerer programmet for alle input  $x > 0$ ? 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?*”

# Plan

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

# Praktiske oplysninger om kurset

- Kursushjemmeside

<http://www.daimi.au.dk/~besen/RegAut/>

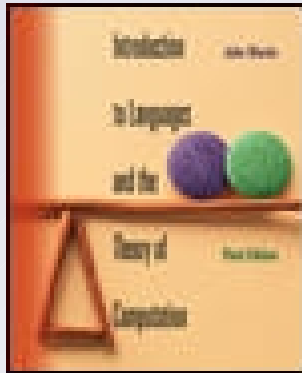
- AULA side (webboard, afleverings-interface):

<http://www.aula.au.dk/courses/EVUREGAUT07/>

- Seminarer

- Lørdag d. 25. August, 9-16
- Lørdag d. 8. September, 9-16
- Lørdag d. 29 September, 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 pr. Uge: ~15 timer
  - $(6 \text{ uger} * 15 \text{ timer/uge} = 90 \text{ timer})$
- Seminarer: 21 timer
- Mellem seminarer: 69
- Forventet hjemmearbejde: ~11 timer/uge
  - $(69 \text{ timer} / 6 \text{ uger} = 11.5 \text{ timer/uge})$
- Der forventes, at man
  - **læser de angivne 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

# Eksamen

Formålet med kurset at den studerende skal opnå følgende kompetencer:

- **referere den basale terminologi** (streng, sprog, klasser af sprog, samt basale operationer på disse)
- **beskrive basale abstrakte sprogformalismer** (regulære udtryk, endelige automater, regulære grammatikker, kontekstfri grammatikker) - fra intuitivt niveau og konkrete eksempler til formel notation og generelle definitioner
- **beskrive egenskaber** ved formalismerne, bl.a. ækvivalens, begrænsninger og beslutningsprocedurer
- **forklare og udføre algoritmer**, der oversætter mellem formalismerne eller afgør beslutningsproblemer - fra konkrete eksempler til generelle og formelle beskrivelser
- **bevise og analysere** egenskaber ved formalismerne (ved hjælp af konstruktive beviser og induktionsbeviser) - fra intuitivt niveau til formelle detaljer.

*Eksamen vil vurdere i hvor høj grad den studerende besidder disse kompetencer*

# Plan

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

# Alfabeter, strenge og sprog

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

# 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:  $x \cdot \epsilon = \epsilon \cdot x = x$  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

- $L_1 = \{\epsilon, 10, 212\}$

- $L_2 = \{cab, abba\}$

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

- Bemærk:

- $L \cdot \{\epsilon\} = \{\epsilon\} \cdot L = L$  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

- $L^k = \underbrace{LL \cdots L}_k$   
konkatenering af  $k$  forekomster af  $L$
- $L^0 = \{\square\}$
- $L^* = \bigcup_{i=0 \dots \square} L^i$  (Kleene stjerne af  $L$ )
- $L^+ = L^*L$

# Rekursive definitioner

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

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

- ~~Enhver selv-referens~~ skal referere til noget "mindre" og føre til endeligt mange selv-referencer



# En rekursiv definition af strenge

- $x$  er en **streng** over alfabetet  $\Sigma$ , dvs.  $x \in \Sigma^*$ , hvis
  - $x = \epsilon$ , eller
  - $x = y \cdot a$  hvor  $y \in \Sigma^*$  og  $a \in \Sigma$

(underforstået:  $\Sigma^*$  er den *mindste* mængde, der opfylder dette)

- Eksempel:

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

# Syntaks af regulære udtryk

Mængden  $R$  af **regulære udtryk** over  $\Sigma$  er den mindste mængde, der indeholder følgende:

- $\emptyset$
- $\Sigma$
- $a$  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  $L(r)$  af et regulært udtryk  $r$  defineres i strukturen af  $r$ :

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

# Regulære sprog

Definition:

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

# Parenteser 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  $\{0,1\}$ :

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

- På grund af parentesreglerne er dette det samme som

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

- Så sproget for  $r$  er

$$\begin{aligned} L(r) &= (((\{1\}\square\{\square\})\{0\})\{0\})\{1\}) \\ &= \{1001,001\} \end{aligned}$$

# Quiz!

1. Hvad betyder  $\{a,bc\}^*$  ?
1. 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

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

# Reverse-operatoren

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

# Reverse på sprog

- Givet et sprog  $L \subseteq \Sigma^*$ , definer
$$\text{Reverse}(L) = \{ \text{reverse}(x) \mid x \in L \}$$
- Eksempel:  
Hvis  $L = \{ \epsilon, 123, abc \}$  så er
$$\text{Reverse}(L) = \{ \epsilon, 321, cba \}$$

# Rekursive definitioner 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  $X$* ”

# Et induktionsbevis

- 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

# Basis

- $r = \emptyset$ :  
vælg  $r' = \emptyset$   
 $L(r') = \emptyset = \text{Reverse}(\emptyset) = \text{Reverse}(L(r))$
- $r = \square$ :  
vælg  $r' = \square \dots$
- $r = a$  hvor  $a \in \Sigma$   
vælg  $r' = a \dots$

# Induktionsskridt

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  $r_1'$  og  $r_2'$  er givet af i.h. ...
- $r = r_1 r_2$  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 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*



# Algoritme

Input: et regulært udtryk  $r$

Definer en rekursiv funktion  $REV$  ved:

- $REV(\emptyset) = \emptyset$
- $REV(\square) = \square$
- $REV(a) = a$ , hvor  $a \in \square$
- $REV(r_1 + r_2) = REV(r_1) + REV(r_2)$
- $REV(r_1 r_2) = REV(r_2) REV(r_1)$
- $REV(r_1^*) = (REV(r_1))^*$

Output: det regulære udtryk  $REV(r)$

# Øvelse

- Lad  $r$  være det regulære udtryk  $((a+\square)cbc)^*$  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.

## dRegAut.RegExp

- Java-repræsentation af regulære udtryk
- Speciel syntax:
  - # betyder  $\emptyset$
  - % betyder  $\square$
- Alfabetet angives som en mængde af Unicode tegn

# Resume

- Alfabeter, strenge, sprog
- Regulære udtryk og regulære sprog
- Rekursive definitioner, induktionsbeviser, konstruktive beviser
- **Java:** `dRegAut.RegExp` **klassen**

# Plan

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

# Løsninger

- [Martin] 3.2:

a) 00

b) 01

c) 0

d) 010

# Løsninger

- [Martin] 3.9 (a-e):

a)  $1*01*01*$

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

c)  $+ 1 + (0+1)^*0 + (0+1)^*11$

d)  $(00 + 11)(0+1)^* + (0+1)^*(00 + 11)$

e)  $(1 + 01)^*(0 + )$