

# Plan

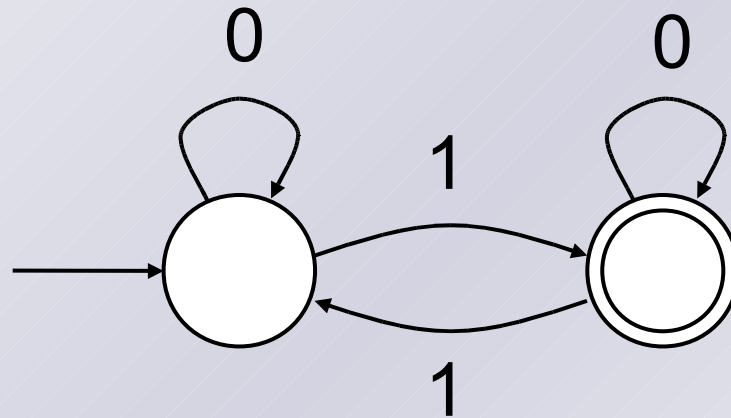
- Hvad er Regularitet og Automater
- Praktiske oplysninger om kurset
- Regulære udtryk
- Induktionsbevis
- Frokost
- **Endelige automater**
- Skelnelighed, Produktkonstruktion
- Præsentation af Java projekt

# Regulære udtryk og 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...)

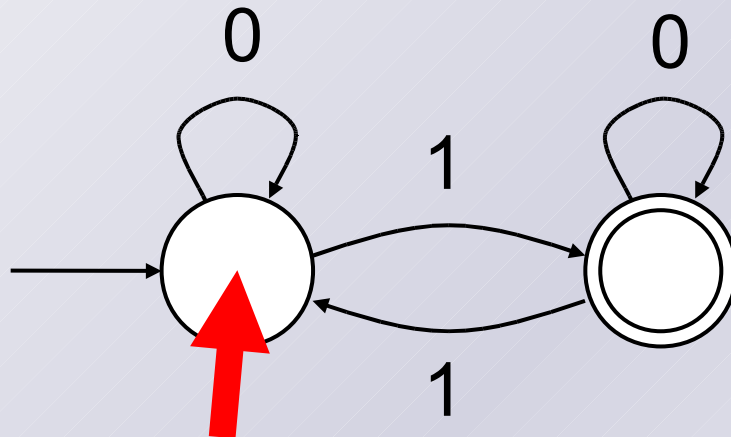
# En endelig automat

- 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



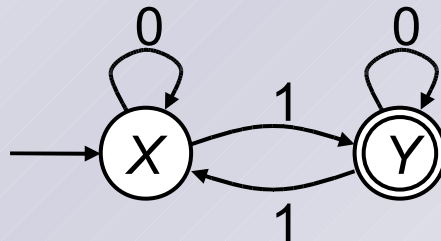
- Eksempel: vi vil vide om strengen 1010 accepteres
- Vi starter i starttilstanden og læser strengen

1010

- Vi ender i en ikke-accept tilstand, så strengen accepteres ikke

# Hvad repræsenterer tilstandene?

- 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”*

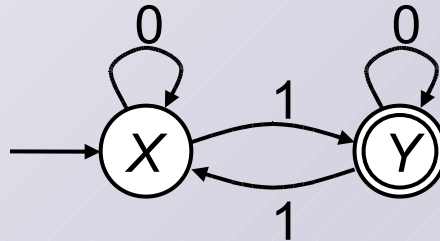
# Formel definition af endelige automater

En *endelig automat* (finite automaton/FA) er et 5-tupel  $(Q, \Sigma, q_0, A, \delta)$  hvor

- $Q$  er en endelig mængde af tilstande
- $\Sigma$  er et alfabet
- $q_0 \in Q$  er en starttilstand
- $A \subseteq Q$  er accepttilstande
- $\delta: Q \times \Sigma \rightarrow Q$  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:

	input	
	0	1
X	X	Y
Y	Y	X

tilstand

# 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 $L(M)$

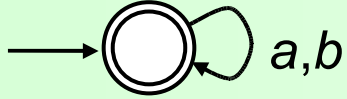
- Givet en FA  $M=(Q, \Sigma, q_0, A, \delta)$ , definer den udvidede transitionsfunktion  $\delta^*: Q \times \Sigma^* \rightarrow Q$  ved

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

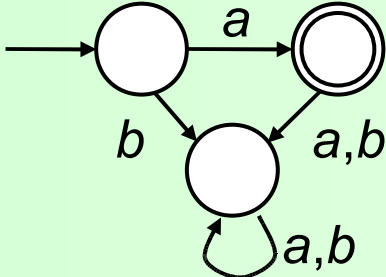
- $x \in \Sigma^*$  accepteres af  $M$  hvis og kun hvis  $\delta^*(q_0, x) \in A$
- Definer  $L(M) = \{ x \in \Sigma^* \mid x \text{ accepteres af } M \}$

# Quiz!

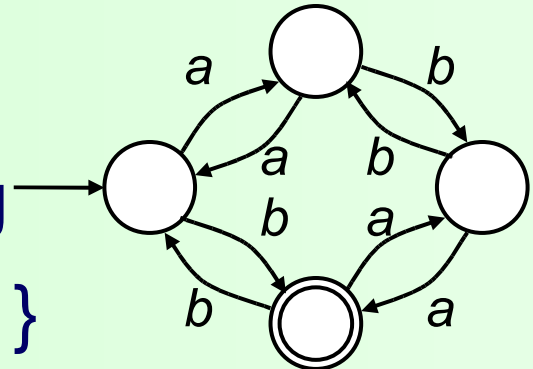
Lad  $\Sigma = \{a, b\}$ . Konstruér en FA  $M$  så...

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

1.  $L(M) = \emptyset$  

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

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



## Øvelser:

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

# Plan

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

# 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  $\exists z \in \Sigma^*: (xz \in L \neq yz \in L) \vee$   
 $(xz \in L \neq yz \in L)$

# Definition af skelnelighed

Lad  $L \subseteq \Sigma^*$  og  $x, y \in \Sigma^*$

- **Kvotientsproget**  $L/x$  defineres som

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

- $x$  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  $y$

- dvs. hvis  $(Q, \delta, q_0, A, \epsilon)$  genkender  $L$   
så er  $\delta^*(q_0, x) \notin \delta^*(q_0, y)$



# 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  $\exists i \neq j: \delta^*(q_0, x_i) = \delta^*(q_0, x_j)$
- modstrid med at  $x_i$  og  $x_j$  er skelnelige mht.  $L$

# Eksempel 1: en stor automat

Lad  $L_{42} = \{ x \in \{0,1\}^* \mid |x| \leq 42 \text{ og det 42. symbol fra højre i } x \text{ er et } 1 \}$

- 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...)
- Vi kan altså finde en vilkårligt stor mængde parvist skelnelige strenge, så  $pal$  er **ikke regulært**

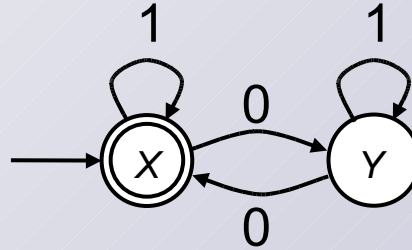
# Forening af regulære sprog

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

# Eksempel

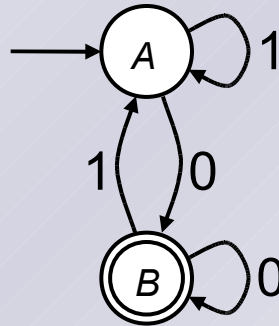
$M_1$ :

(strengene med lige antal 0'er)



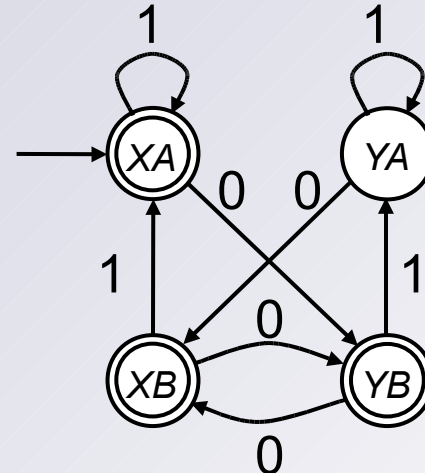
$M_2$ :

(strengene der ender med 0)



$M$ :

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




# Produktkonstruktionen

Antag vi har to FA'er:

- $M_1 = (Q_1, \square, q_1, A_1, \square_1)$
- $M_2 = (Q_2, \square, q_2, A_2, \square_2)$

Definer en ny FA:

$M = (Q, \square, q_0, A, \square)$  hvor

- $Q = Q_1 \square Q_2$   produktmængden af tilstande
- $q_0 = (q_1, q_2)$
- $A = \{ (p, q) \mid p \square A_1 \square q \square A_2 \}$
- $\square((p, q), a) = (\square_1(p, a), \square_2(q, a))$

Der gælder nu:

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

# Konstruktivt bevis for korrekthed

- Lemma:

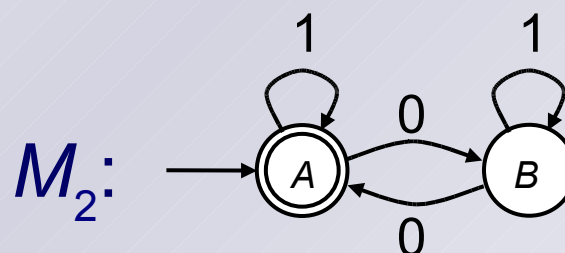
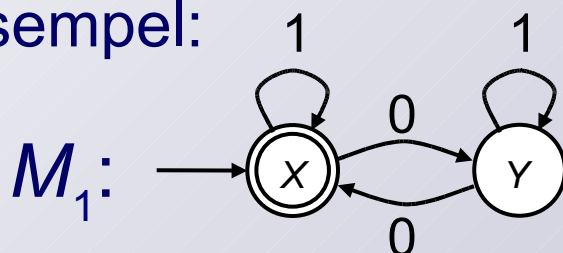
$$\forall x \in \Sigma^*: \Sigma^*((p, q), x) = (\Sigma_1^*(p, x), \Sigma_2^*(q, x))$$

(Bevis: opgave 3.32, induktion i  $x$ )

- Brug lemmaet samt definitionerne af  $M$  og  $L(\bullet)$

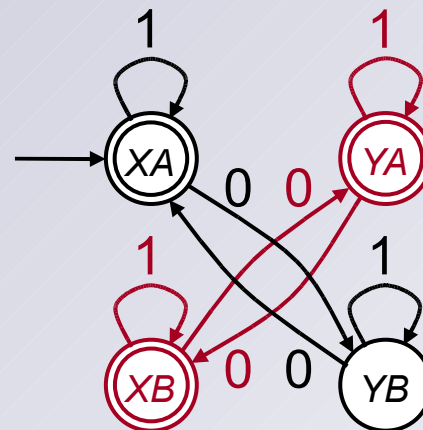
# Nøjes med opnåelige tilstande

- Produktkonstruktionen bruger  $Q = Q_1 \square Q_2$
- I praksis er hele tilstandsrummet sjældent nødvendigt
- Eksempel:



$M$ :

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



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



# Snitmængde og differens

Givet to regulære sprog,  $L_1$  og  $L_2$

1. er  $L_1 \cap L_2$  også regulært?

2. er  $L_1 - L_2$  også regulært?

- **Ja!** (dvs. klassen af regulære sprog er lukket under snit og differens)
- **Bevis:** produktkonstruktion som ved  $\cap$  men
  - for  $\cap$ , vælg  $A = \{ (p, q) \mid p \in A_1 \text{ og } q \in A_2 \}$
  - for  $-$ , vælg  $A = \{ (p, q) \mid p \in A_1 \text{ og } q \notin A_2 \}$

# Komplement

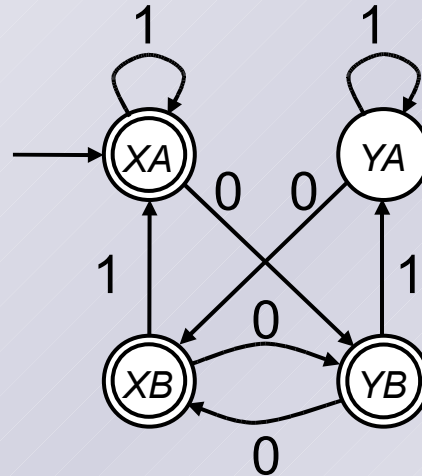
Givet et regulære sprog  $R$   
er  $R'$  ( $R$ s komplement) også regulært?

- **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  $M' = (Q, \Sigma, q_0, \mathbf{Q-A}, \delta)$
  - Derved gælder at  $L(M') = R'$

# Eksempel

$M$ :

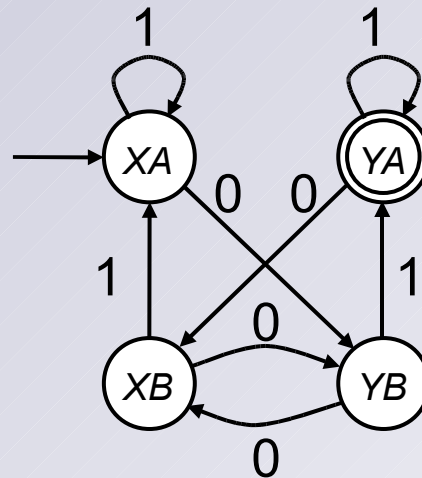
(strengene med lige antal 0'er  
eller ender med 0)



$M'$ :

$$L(M') = (L(M))'$$

(strengene med **ulige** antal 0'er  
**og** ender **ikke** med 0)



## Øvelser:

- [Martin] 3.33 (a-c)

# dRegAut Java-pakken

## Udleverede programdele:

- `FA.java`:  
repræsentation af FA'er
- `Alphabet.java`,  
`State.java`,  
`StateSymbolPair.java`,  
`AutomatonNotWellDefinedException.java`:  
hjælpeklasser 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; //  $\Sigma \times Q \rightarrow Q$   
    ...  
}
```

- **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)

# Automater til modellering og verifikation

Eksempel: *en jernbaneoverskæring*

- Tre komponenter:

- et **tog**
  - krydser vejen
  - kommunikerer med kontrolsystemet
- et **kontrolsystem**
  - styrer bommen
- en **bom**



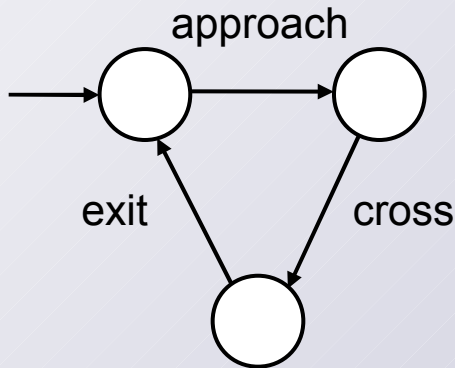
- Sikkerhedsegenskab:

*bommen er altid nede, når toget krydser vejen*

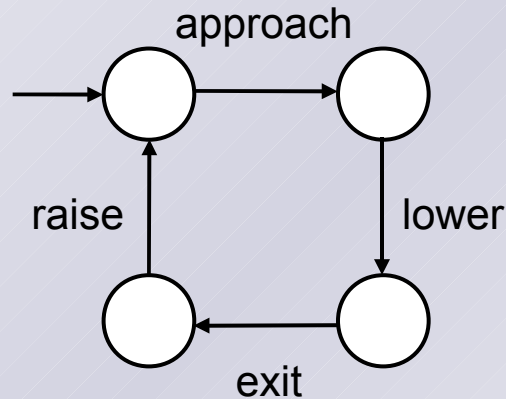


# Modellering af systemet

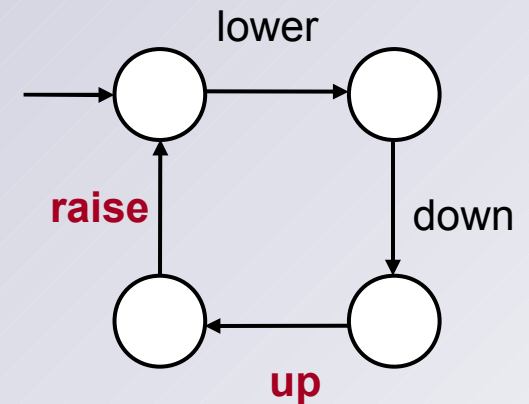
## TOG



## KONTROLSYSTEM



## BOM



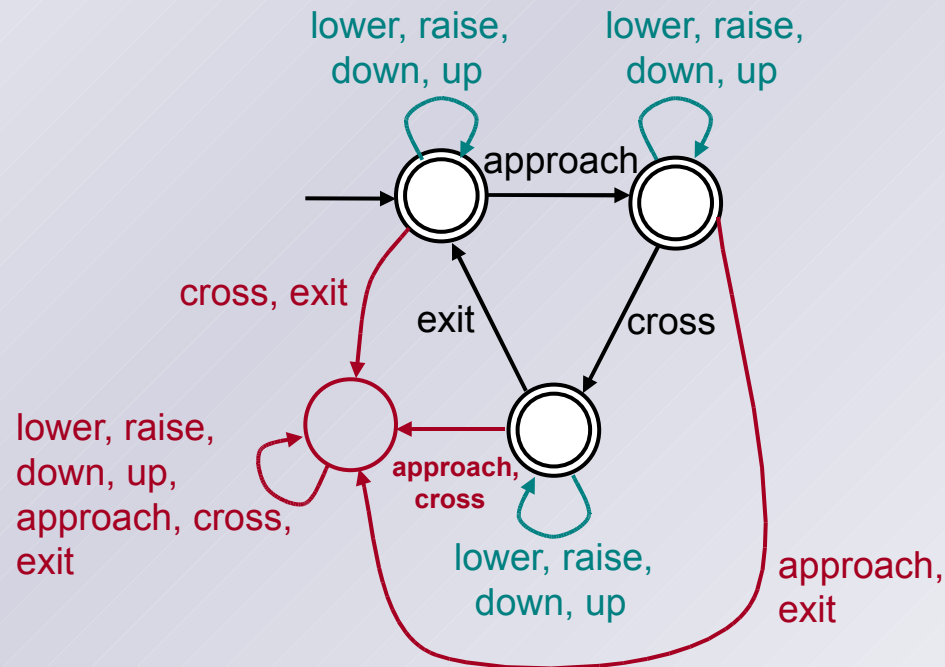
### Begivenheder:

- 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'er

Eksempel:

$M_{\text{TOG}} =$



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

# Kombination af komponenterne

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

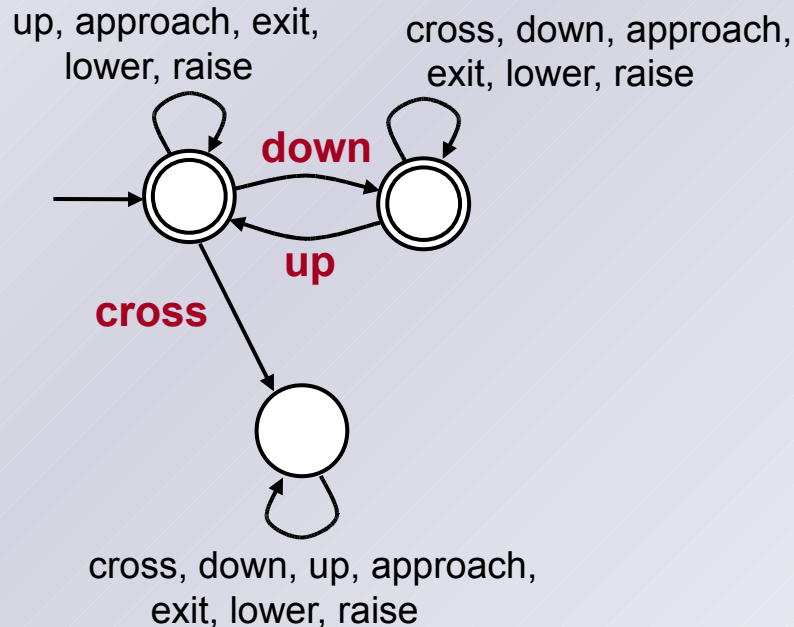
- Produktkonstruktion:

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

# Modellering af sikkerhedsegenskaben

– *bommen er altid nede, når toget krydser vejen*

$S =$



# Verifikation

- Korrekthed:  $L(M) \sqsubseteq (L(S))' = \emptyset$
- dvs. vi skal bruge
  - produktkonstruktion (igen)
  - komplement
  - algoritme til at afgøre om sproget for en given FA er tomt (3. seminar)
- hvis  $L(M) \sqsubseteq (L(S))' \not\sqsubseteq \emptyset$ :  
enhver streng i  $L(M) \sqsubseteq (L(S))'$  svarer til  
et **mod eksemp** (algoritme: 3. seminar)

# Verifikation med dRegAut-pakken

- Opbyg FA-objekter svarende til  $M_{\text{TOG}}$ ,  $M_{\text{KONTROL}}$ ,  $M_{\text{BOM}}$ , 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

# Plan

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



# Første del af java projekt:

- 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