

Transformasjon fra NFA (M) til DFA (M')

(\bar{M})

“The subset construction”

▪ DEFINISJONER

Gitt en mengde (et ”utplukk”) S av M-tilstander. Vi definerer:

- ϵ -tillukning av S: S selv, utvidet med alle M-tilstander man kan nå ved bare å bruke (en eller flere) ϵ -kanter fra S
- S_a : Den mengden av M-tilstander man kan nå fra S ved å følge én a-kant.

▪ IDÉ VED KONSTRUKSJONEN

- Hver tilstand S i M' er definert ved en mengde av M-tilstander
- Gitt streng α . I M' skal denne føre til tilstanden S definert ved *alle de M-tilstander som α kan føre fram til* ved forskjellige ikke-deterministiske valg i M.

Transformasjon fra NFA: M til DFA: M'

ALGORITME

1. Lag start-tilstanden i M' som ε -tillukningen av {start-tilstanden i M }
2. Om S er en tilstand i M' , så gjelder:

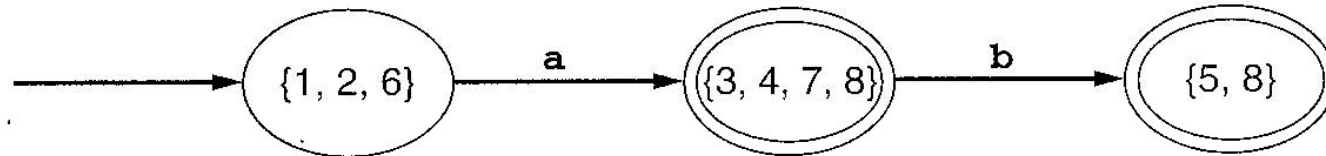
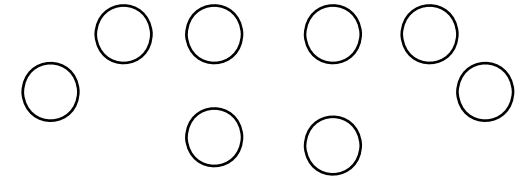
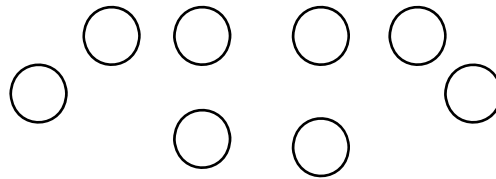
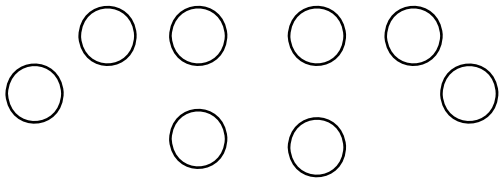
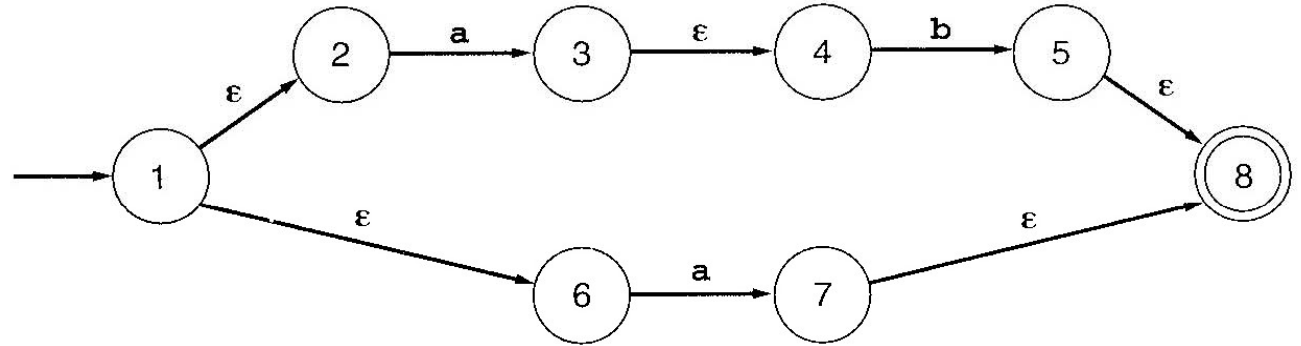


(Må eventuelt opprette ny M' -tilstand)

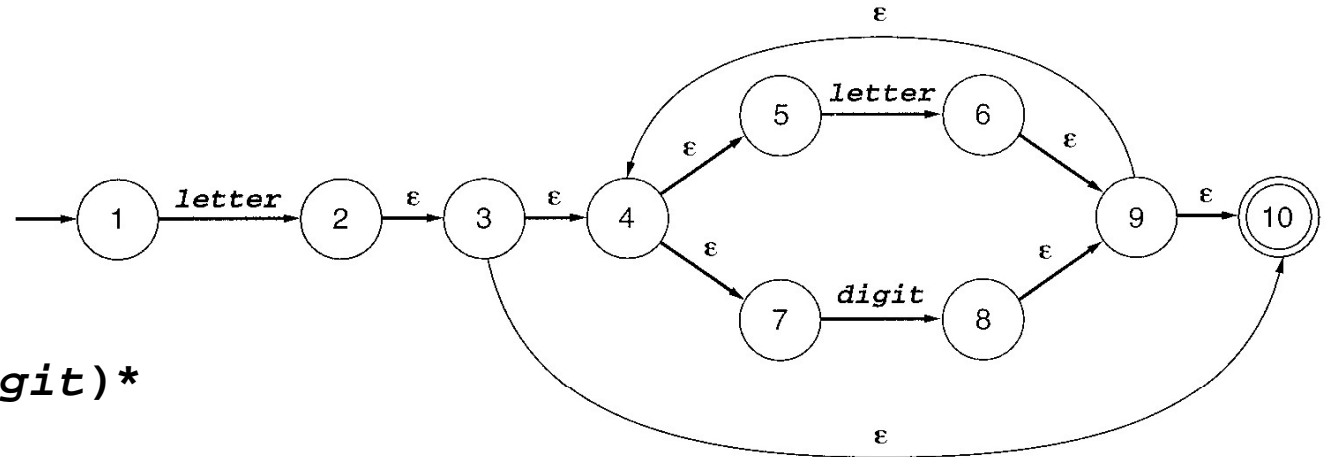
3. Gjenta steg 2 for alle S i M' og alle a i alfabetet til M' ikke får flere nye tilstander.
4. De aksepterende tilstander i M' er de S i M' som inneholder minst én av de aksepterende tilstander i M .

Eksempel 2.16

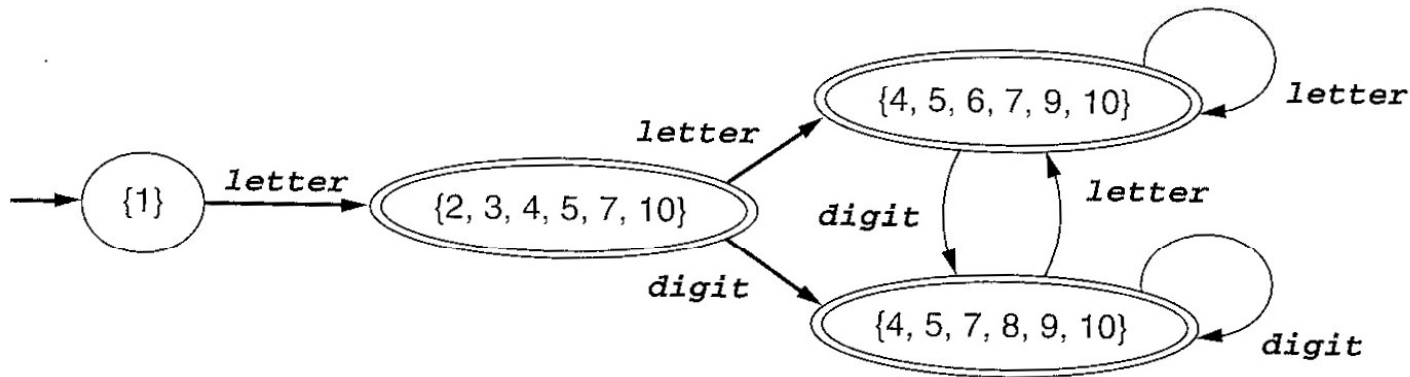
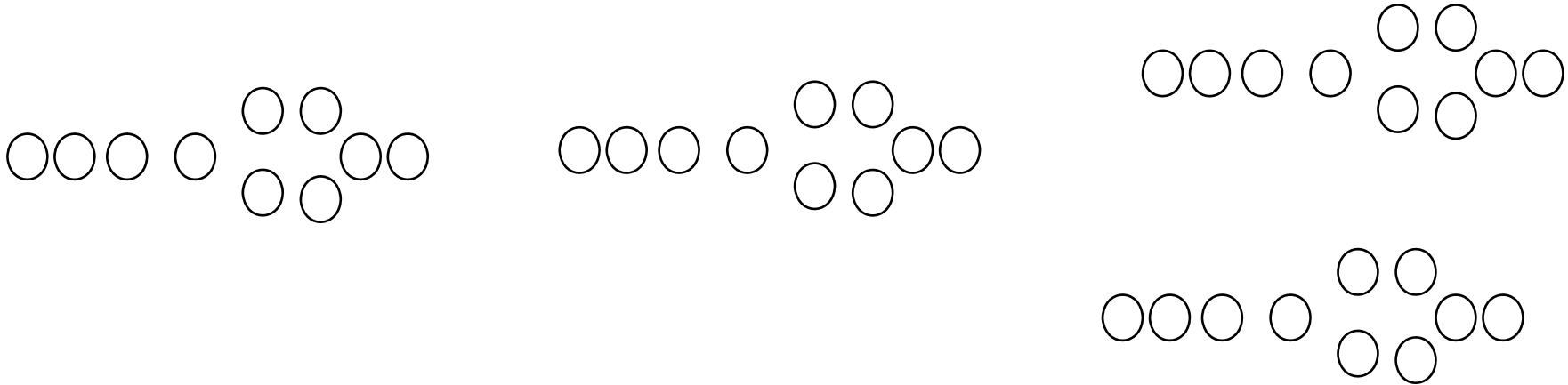
ab | a



Eksempel 2.17



*letter (letter | digit)**



Minimalisering av DFA

- Ved automatisk konstruksjon av DFA-er (f.eks. gjennom Thompson-konstr.) får vi ofte unødvendig mange tilstander.
- Skal se på algoritme som slår sammen tilstander så mye som mulig (uten å forandre språket som aksepteres).
- For denne algoritmen gjelder (vises ikke):
 - Alle DFA-er for samme språk transformerers til samme DFA
 - Denne har så få tilstander som mulig
- Som en bieffekt får vi derved en algoritme for å svare på:
 - Gitt to DFA-er. Aksepterer de samme språk?
 - Gitt to regulære uttrykk. Beskriver de samme språk?

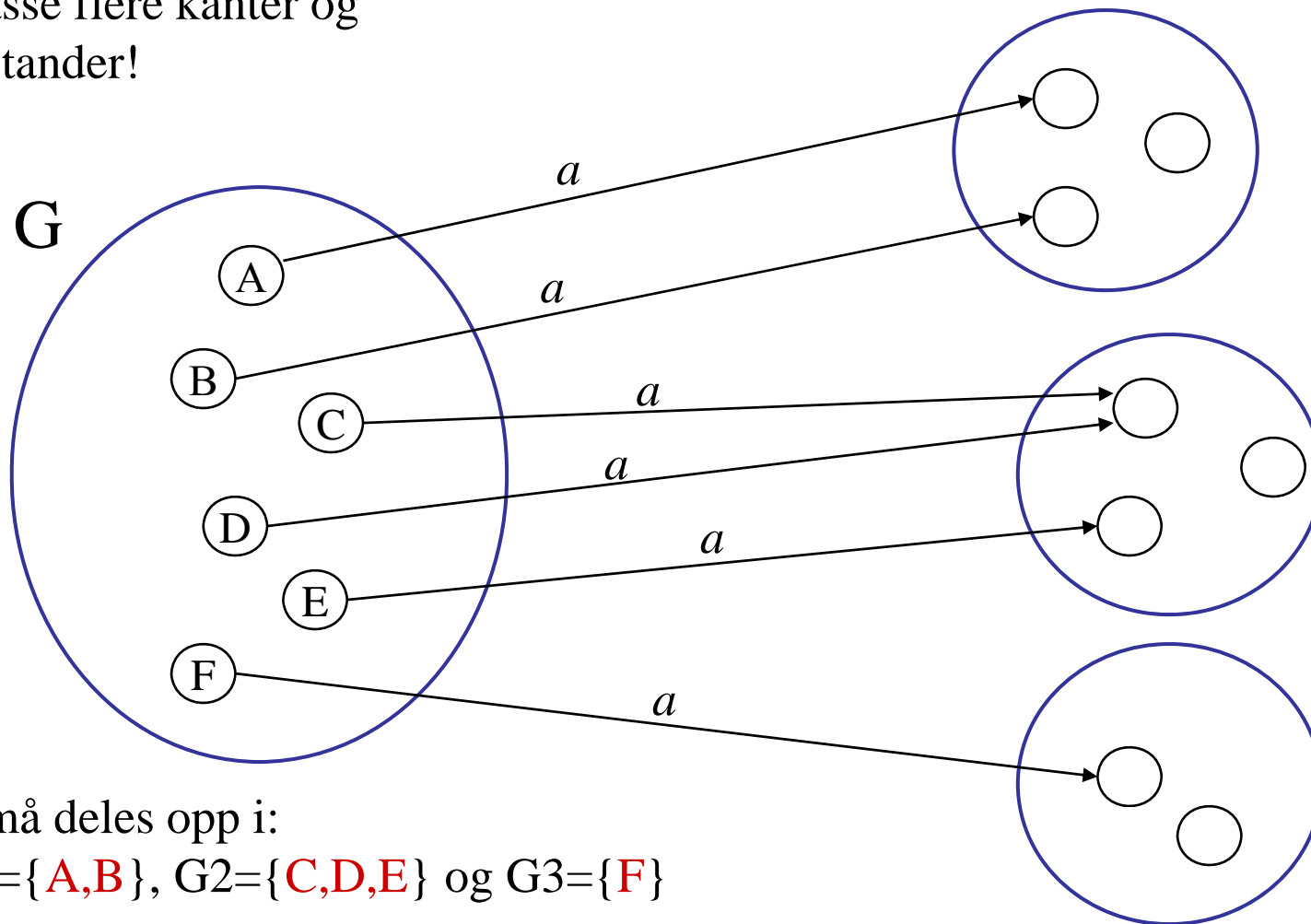
Algoritme for å minimalisere DFA

- Utvid til "fullstendig DFA" M , ved eventuelt å legge til "feil-tilstand"
- Algoritmen begynner med å gruppere sammen tilstander i M så optimistisk som mulig:
 - Alle aksepterende tilstander slås sammen
 - Alle ikke-aksepterende tilstander slås sammen
- Algoritmen splitter så opp disse gruppene der det er nødvendig.
STÈG: Se på gruppe $G = \{s_1, s_2, \dots, s_n\}$ og et symbol a .
 - Dersom $t(s_1, a), t(s_2, a), \dots, t(s_n, a)$ alle er i samme gruppe er alt OK.
 - Ellers må man splitte opp G i G_1, G_2, \dots, G_m , slik at kravet over gjelder for hver G_i . (Se tegning egen foil)
- Gjenta dette steget for alle (sammenslåtte) G og alle symboler a , inntil:
 - Enten oppsplittingen er blitt fullstendig (vi er tilbake til den opprinnelige DFA)
 - Eller det ikke blir mer oppsplitting
- Sett på transisjons-overgangene mellom de nye sammenslåtte tilstandene. Disse er nå entydig definerte.

Steget i Minimalisering av DFA

NB1: Bare a -kanter fra G -tilstander er tegnet. Det er masse flere kanter og tilstander!

NB2: En av disse kan være G selv:

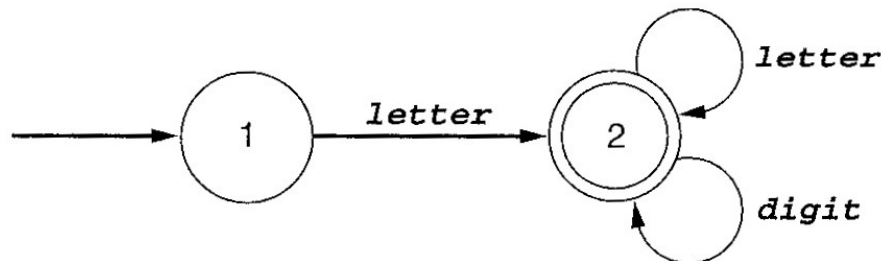
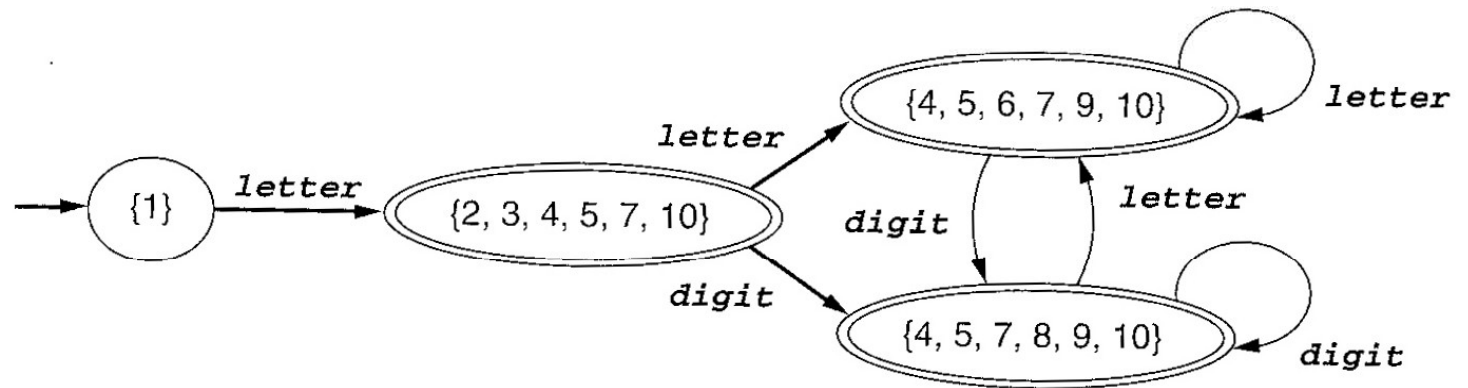


G må deles opp i:

$G1 = \{A, B\}$, $G2 = \{C, D, E\}$ og $G3 = \{F\}$

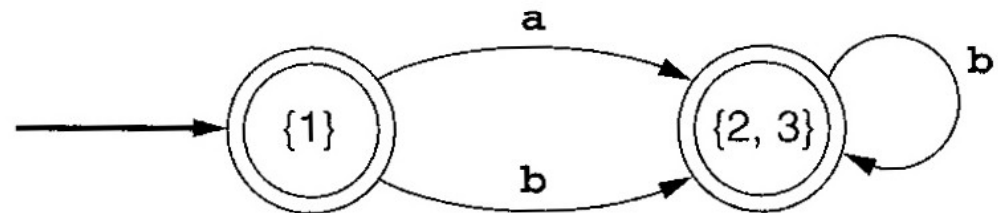
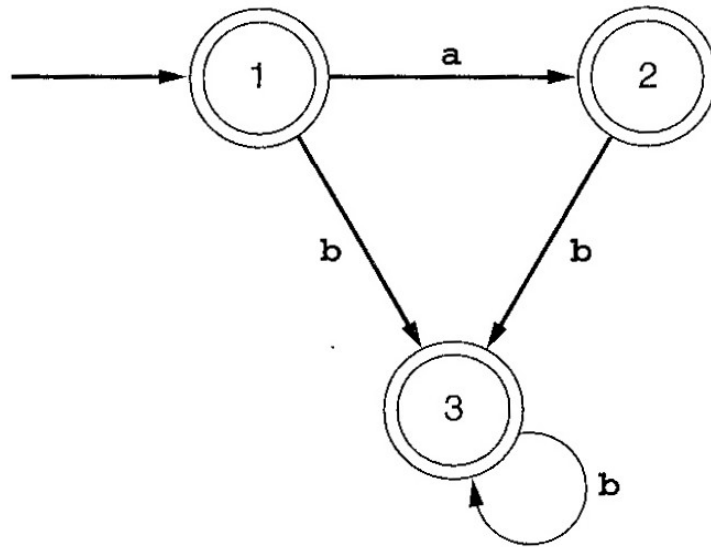
Eksempel 2.18

*letter (letter | digit)**



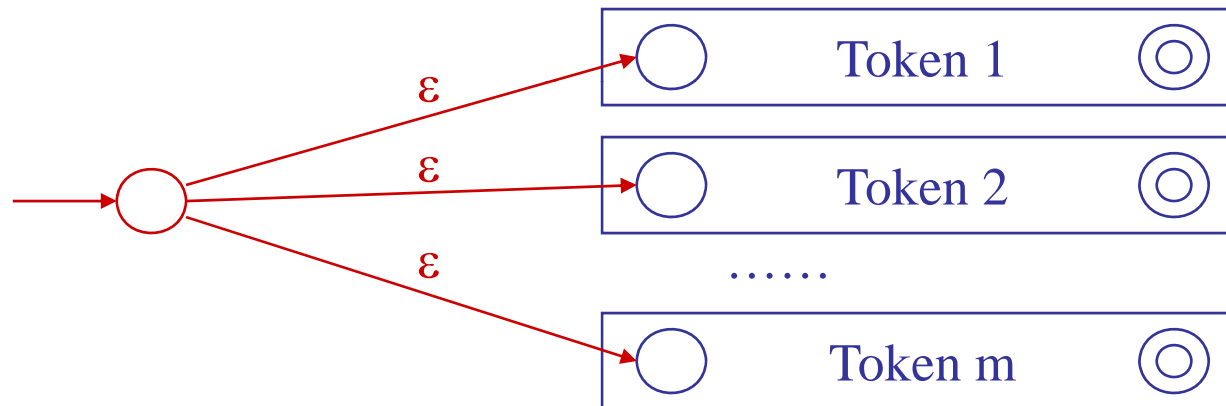
Eksempel 2.19

$(a \mid \varepsilon) b^*$



Hovedidé ved Lex/Flex og liknende

- Bruker setter opp regulær definisjon av hver token-klasse (samt tilhørende aksjoner som skal utføres når slikt token er funnet)
- Disse gjøres om til NFA-er (f.eks. ved Thompson konstr.)
- Disse enkelt-NFA-ene settes sammen til stor NFA slik:



- Denne gjøres om til DFA ved "subset construction".
- Gjør eventuelt minimalisering av DFA-en (med "litt forsiktighet", slik at token-klasser kan skilles)
- Implementer DFA-en, vanligvis tabell-drevet