Stockholms matematiska cirkel
Datorernas matematik

www.math-stockholm.se/cirkel

16.00–16.15: Fika
16.15–17.15: Föreläsning om kapitel 6
17.15–17.30: Rast
17.30–18.00: Utbildningsinformation

Översikt

1. Vad är matematik, egentligen?
2. Hur kan en dator räkna?
3. Tal med decimaler
4. Tärningen är kastad
5. Formella språk
6. **Tillståndsmaskiner**
7. Tillståndsmaskinernas språk

- Nästa **övning** är **12 mar** (nästa vecka) i sal **V3**
- Nästa **föreläsning** är **26 mar** i sal **F2**

- Glöm inte närvarolistan!

Kapitel 6 – Tillståndsmaskiner

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck

$(c \cup k)a(rl \cup ll^*e)$

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔
	carl

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔
	carl ✔

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔
	carl ✔
	kalle

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔
	carl ✔
	kalle ✔

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔
	carl ✔
	kalle ✔
	calle

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔
	carl ✔
	kalle ✔
	calle ✔

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔
	carl ✔
	kalle ✔
	calle ✔
	cale

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔
	carl ✔
	kalle ✔
	calle ✔
	cale ✔

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔
	carl ✔
	kalle ✔
	calle ✔
	cale ✔
	callllle

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔
	carl ✔
	kalle ✔
	calle ✔
	cale ✔
	callllle ✔

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔
	carl ✔
	kalle ✔
	calle ✔
	cale ✔
	callllle ✔
	kall

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔
	carl ✔
	kalle ✔
	calle ✔
	cale ✔
	callllle ✔
	kall ✘

Kapitel 6 – Tillståndsmaskiner

Förra gången: reguljära språk och reguljära uttryck

Reguljärt uttryck	Ord
$(c \cup k)a(rl \cup ll^*e)$	karl ✔
	carl ✔
	kalle ✔
	calle ✔
	cale ✔
	callllle ✔
	kall ✘

Hur avgör en dator
om ett ord ”matchar” ett reguljärt uttryck?

Datorn måste ha en **algoritm** att följa.

Datorn måste ha en **algoritm** att följa.

En **algoritm**

- består av en uppsättning instruktioner,

Datorn måste ha en **algoritm** att följa.

En **algoritm**

- består av en uppsättning instruktioner,
- som tar indata av en viss typ,

Datorn måste ha en **algoritm** att följa.

En **algoritm**

- består av en uppsättning instruktioner,
- som tar indata av en viss typ,
- och producerar utdata av en viss typ.

Datorn måste ha en **algoritm** att följa.

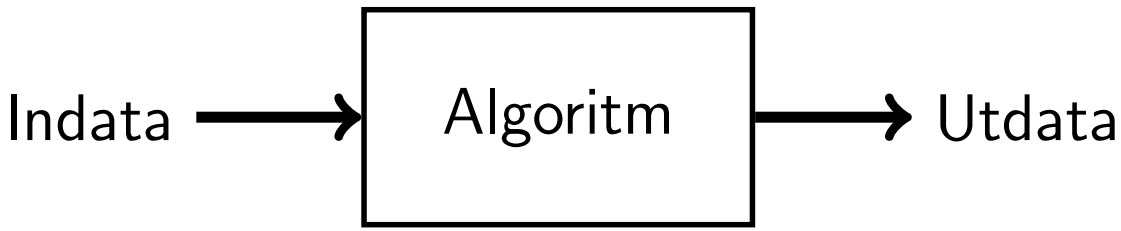
En **algoritm**

- består av en uppsättning instruktioner,
- som tar indata av en viss typ,
- och producerar utdata av en viss typ.
- Instruktionerna måste vara **beräkningsbara**.

Datorn måste ha en **algoritm** att följa.

En **algoritm**

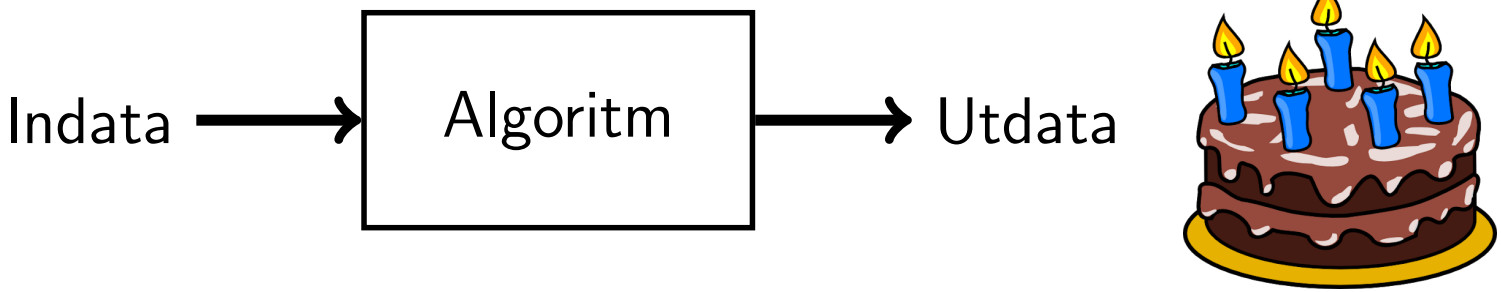
- består av en uppsättning instruktioner,
- som tar indata av en viss typ,
- och producerar utdata av en viss typ.
- Instruktionerna måste vara **beräkningsbara**.



Datorn måste ha en **algoritm** att följa.

En **algoritm**

- består av en uppsättning instruktioner,
- som tar indata av en viss typ,
- och producerar utdata av en viss typ.
- Instruktionerna måste vara **beräkningsbara**.



al-Khwarizmi
(ca. 780–850)

Exempel på algoritmer

- Beräkning av addition, subtraktion, multiplikation, division för hand
- Intervallhalveringsmetoden (kapitel 3)
- Pseudoslumptalsgeneratorer (kapitel 4)

al-Khwarizmi
(ca. 780–850)

Exempel på algoritmer

- Beräkning av addition, subtraktion, multiplikation, division för hand
- Intervallhalveringsmetoden (kapitel 3)
- Pseudoslumptalsgeneratorer (kapitel 4)

Vad betyder det att instruktionerna ska vara **beräkningsbara**?

al-Khwarizmi
(ca. 780–850)

Exempel på algoritmer

- Beräkning av addition, subtraktion, multiplikation, division för hand
- Intervallhalveringsmetoden (kapitel 3)
- Pseudoslumptalsgeneratorer (kapitel 4)

Vad betyder det att instruktionerna ska vara **beräkningsbara**?

Church–Turings tes:
En instruktion är beräkningsbar om den kan utföras av en *Turingmaskin*.

En *Turingmaskin* är en teoretisk datormaskin som kan simulera alla kända datorer.

En *Turingmaskin* är en teoretisk datormaskin som kan simulera alla kända datorer.

Vi kommer studera *tillståndsmaskiner*, som kan ses som en Turingmaskin utan minne.

En *Turingmaskin* är en teoretisk datormaskin som kan simulera alla kända datorer.

Vi kommer studera *tillståndsmaskiner*, som kan ses som en Turingmaskin utan minne.

För att avgöra om ett ord ”matchar” ett reguljärt uttryck kan datorn konstruera och köra en tillståndsmaskin.

En *Turingmaskin* är en teoretisk datormaskin som kan simulera alla kända datorer.

Vi kommer studera *tillståndsmaskiner*, som kan ses som en Turingmaskin utan minne.

För att avgöra om ett ord "matchar" ett reguljärt uttryck kan datorn konstruera och köra en tillståndsmaskin.

Innehåll

- **6.1** Deterministiska tillståndsmaskiner (DTM)
- **6.2** Icke-deterministiska tillståndsmaskiner (ITM)
- **6.3** Delmängdskonstruktionen

Kapitel 6.1 – DTM

En **deterministisk tillståndsmaskin** (DTM) består av:

Kapitel 6.1 – DTM

En **deterministisk tillståndsmaskin** (DTM) består av:

- En ändlig mängd tillstånd Ω

Kapitel 6.1 – DTM

En **deterministisk tillståndsmaskin** (DTM) består av:

- En ändlig mängd tillstånd Ω
- Ett utvalt starttillstånd $S_0 \in \Omega$

Kapitel 6.1 – DTM

En **deterministisk tillståndsmaskin** (DTM) består av:

- En ändlig mängd tillstånd Ω
- Ett utvalt starttillstånd $S_0 \in \Omega$
- En mängd accepterande tillstånd $F \subseteq \Omega$

Kapitel 6.1 – DTM

En **deterministisk tillståndsmaskin** (DTM) består av:

- En ändlig mängd tillstånd Ω
- Ett utvalt starttillstånd $S_0 \in \Omega$
- En mängd accepterande tillstånd $F \subseteq \Omega$
- En övergångsfunktion $\delta : \Omega \times \Sigma \to \Omega$
  (där Σ är ett bestämt alfabet)

Kapitel 6.1 – DTM

En **deterministisk tillståndsmaskin** (DTM) består av:

- En ändlig mängd tillstånd Ω
- Ett utvalt starttillstånd $S_0 \in \Omega$
- En mängd accepterande tillstånd $F \subseteq \Omega$
- En övergångsfunktion $\delta : \Omega \times \Sigma \to \Omega$
  (där Σ är ett bestämt alfabet)

Tillståndsmaskinen håller reda på vilket tillstånd den är i.

Kapitel 6.1 – DTM

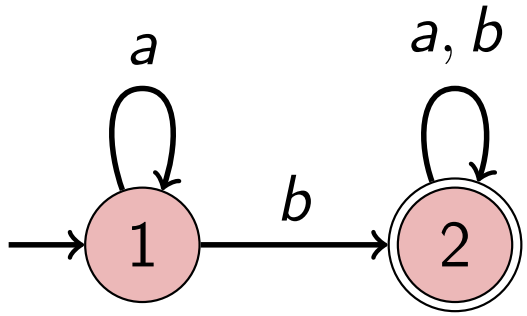
En **deterministisk tillståndsmaskin** (DTM) består av:

- En ändlig mängd tillstånd Ω
- Ett utvalt starttillstånd $S_0 \in \Omega$
- En mängd accepterande tillstånd $F \subseteq \Omega$
- En övergångsfunktion $\delta : \Omega \times \Sigma \to \Omega$
  (där Σ är ett bestämt alfabet)

Tillståndsmaskinen håller reda på vilket tillstånd den är i.

Med hjälp av övergångsfunktionen kan vi mata in ett tecken i tillståndsmaskinen och få den att byta tillstånd.

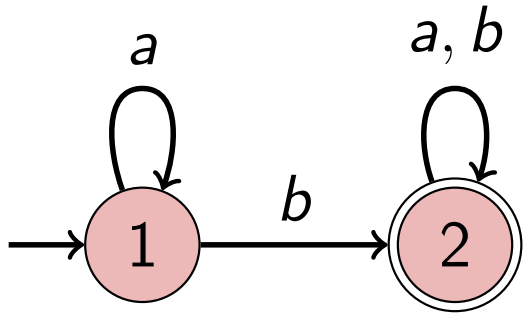
Exempel 6.1.1.



Figur 6.1

- $\Sigma = \{a, b\}$
- $\Omega = \{1, 2\}$
- $S_0 = 1$
- $F = \{2\}$

Exempel 6.1.1.



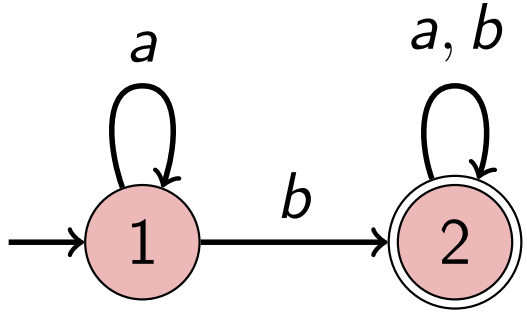
Figur 6.1

- $\Sigma = \{a, b\}$
- $\Omega = \{1, 2\}$
- $S_0 = 1$
- $F = \{2\}$

Tabell för övergångsfunktionen:

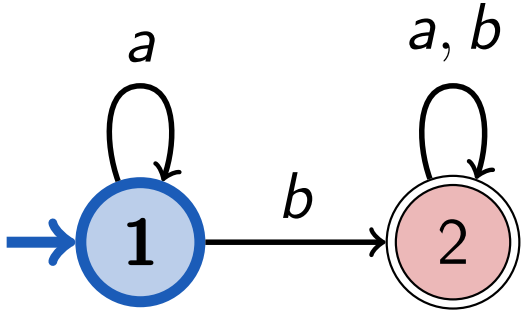
Ω	1	1	2	2
Σ	a	b	a	b
Ω	1	2	2	2

Vad händer om vi matar in ordet *aababa* i maskinen?



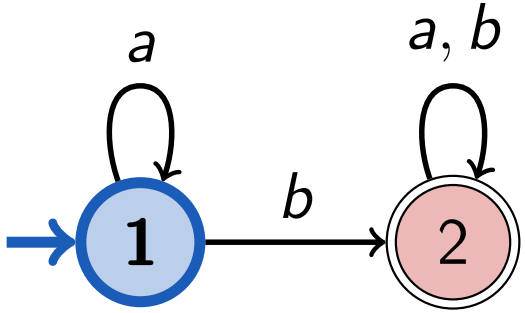
aababa

Vad händer om vi matar in ordet *aababa* i maskinen?



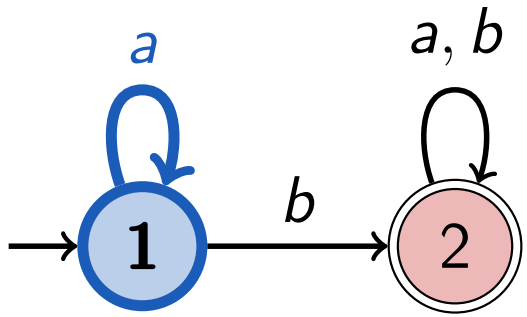
aababa

Vad händer om vi matar in ordet *aababa* i maskinen?



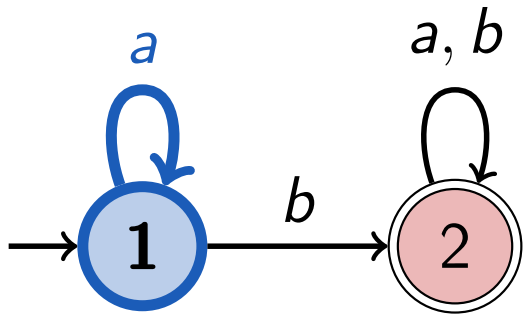
*a*ababa

Vad händer om vi matar in ordet *aababa* i maskinen?



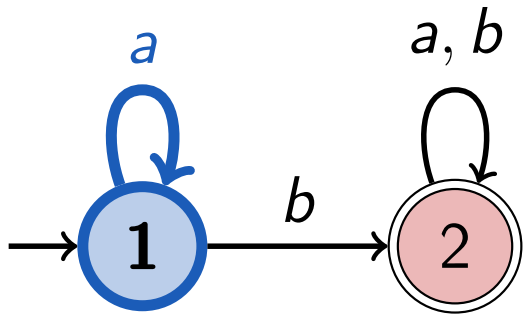
aababa

Vad händer om vi matar in ordet *aababa* i maskinen?



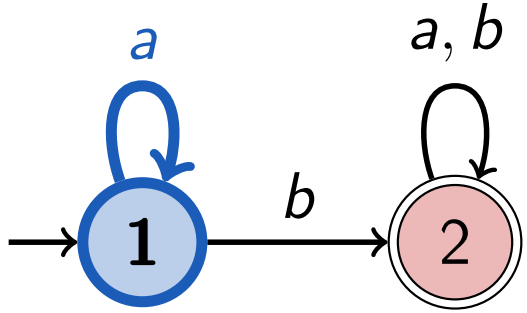
aababa

Vad händer om vi matar in ordet *aababa* i maskinen?



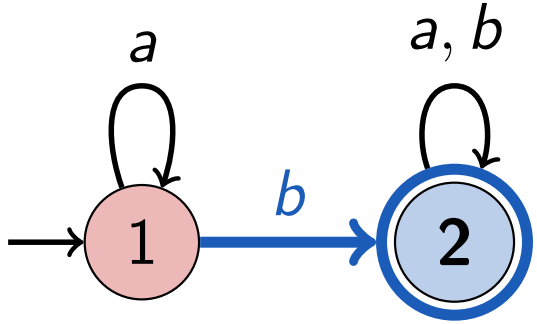
aababa

Vad händer om vi matar in ordet *aababa* i maskinen?



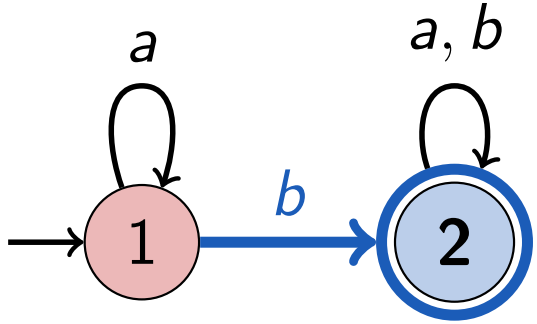
aababa

Vad händer om vi matar in ordet *aababa* i maskinen?



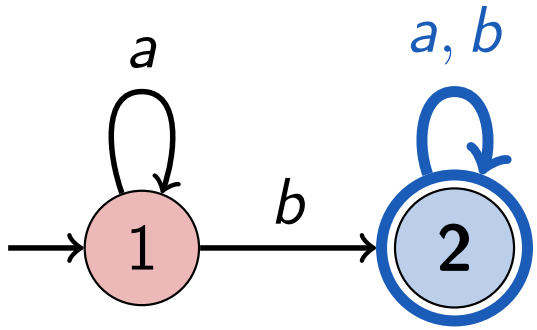
*aab*aba

Vad händer om vi matar in ordet *aababa* i maskinen?



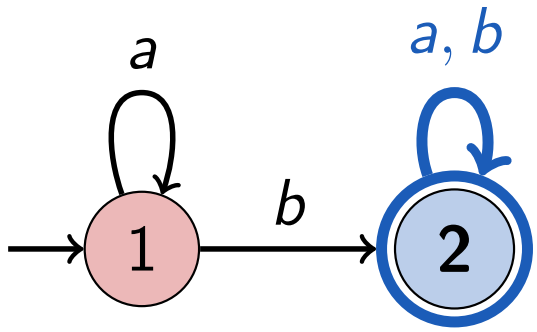
*aab**a**ba*

Vad händer om vi matar in ordet *aababa* i maskinen?



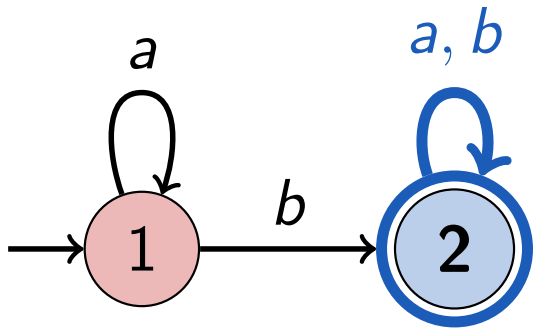
aababa

Vad händer om vi matar in ordet *aababa* i maskinen?



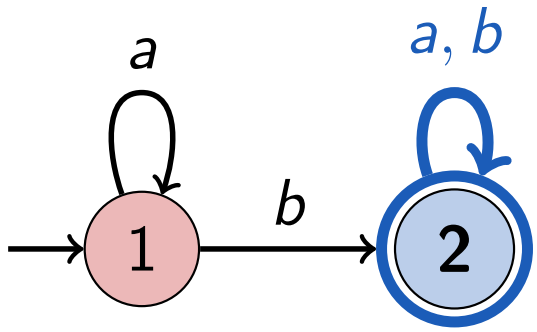
*aabab**b**a*

Vad händer om vi matar in ordet *aababa* i maskinen?



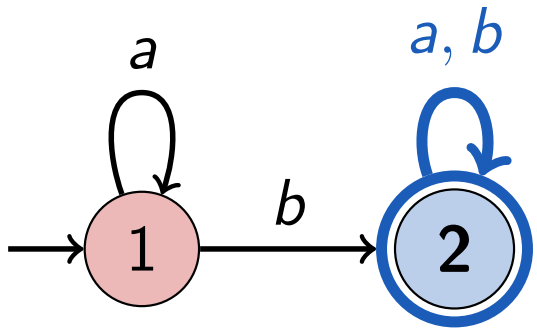
aababa

Vad händer om vi matar in ordet *aababa* i maskinen?



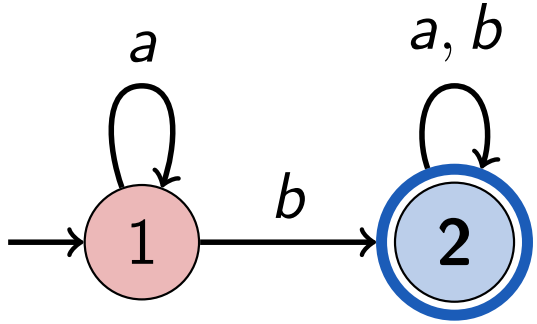
*aababa***a**

Vad händer om vi matar in ordet *aababa* i maskinen?



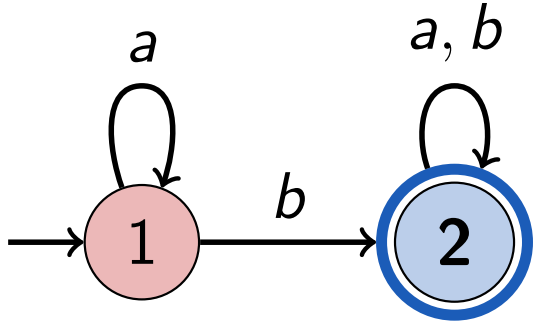
aababa

Vad händer om vi matar in ordet *aababa* i maskinen?



aababa

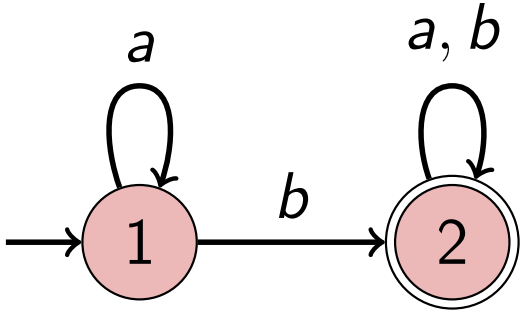
Vad händer om vi matar in ordet *aababa* i maskinen?



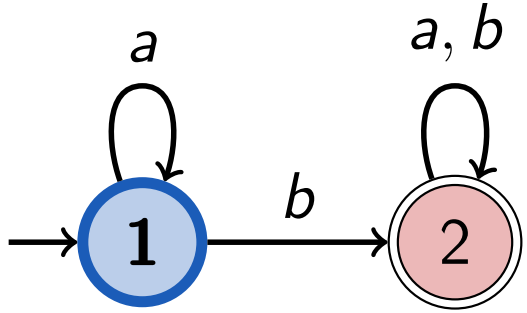
aababa

Eftersom vi slutar i ett accepterande tillstånd (2) kommer maskinen acceptera detta ord – det "matchar"

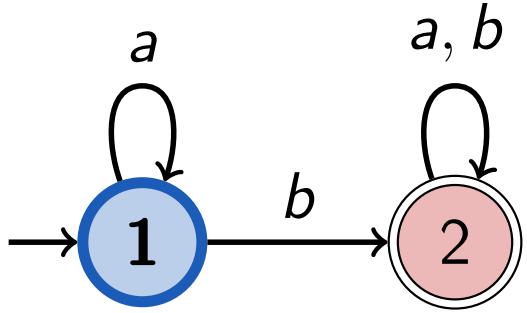
Vad händer om vi matar in ordet *aaa* i maskinen?



Vad händer om vi matar in ordet *aaa* i maskinen?

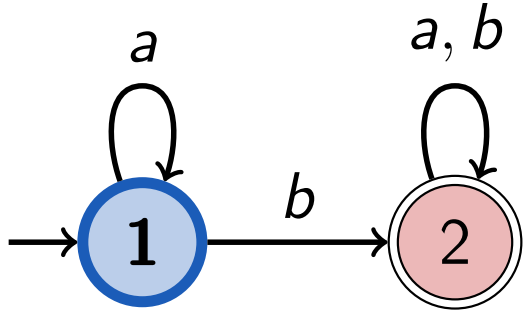


Vad händer om vi matar in ordet *aaa* i maskinen?



Vi slutar i tillstånd 1 som **inte** är accepterande,
så *aaa* accepteras inte – det ”matchar” inte

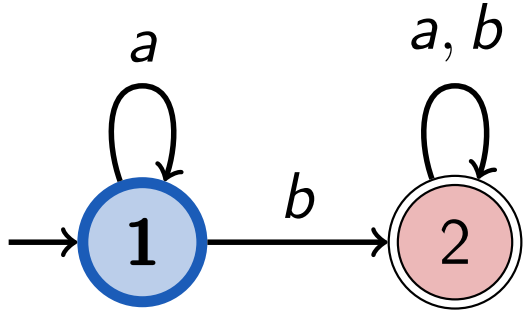
Vad händer om vi matar in ordet *aaa* i maskinen?



Vi slutar i tillstånd 1 som **inte** är accepterande,
så *aaa* accepteras inte – det ”matchar” inte

Vilka ord accepteras av denna maskin?

Vad händer om vi matar in ordet *aaa* i maskinen?

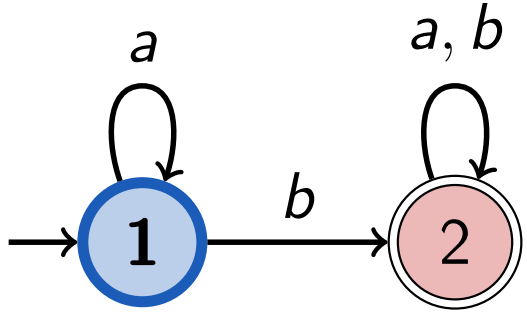


Vi slutar i tillstånd 1 som **inte** är accepterande,
så *aaa* accepteras inte – det ”matchar” inte

Vilka ord accepteras av denna maskin?

Alla ord som innehåller minst ett *b*

Vad händer om vi matar in ordet *aaa* i maskinen?



Vi slutar i tillstånd 1 som **inte** är accepterande,
så *aaa* accepteras inte – det ”matchar” inte

Vilka ord accepteras av denna maskin?

Alla ord som innehåller minst ett b

Maskinen motsvarar alltså det reguljära uttrycket $a^*b(a \cup b)^*$

En matematisk beskrivning av vad vi nyss gjorde:

En matematisk beskrivning av vad vi nyss gjorde:

Definition 6.1.5. Den **utvidgade övergångsfunktionen**

$$\delta^* : \Omega \times \Sigma^* \to \Omega$$

beskriver vad som händer när man matar in ett ord.

En matematisk beskrivning av vad vi nyss gjorde:

Definition 6.1.5. Den **utvidgade övergångsfunktionen**

$$\delta^* : \Omega \times \Sigma^* \to \Omega$$

beskriver vad som händer när man matar in ett ord.

Den definieras rekursivt:

$$\begin{cases} \delta^*(S, \varepsilon) = S \\ \delta^*(S, \sigma w) = \delta^*(\delta(S, \sigma), w) \end{cases}$$

En matematisk beskrivning av vad vi nyss gjorde:

Definition 6.1.5. Den **utvidgade övergångsfunktionen**

$$\delta^* : \Omega \times \Sigma^* \to \Omega$$

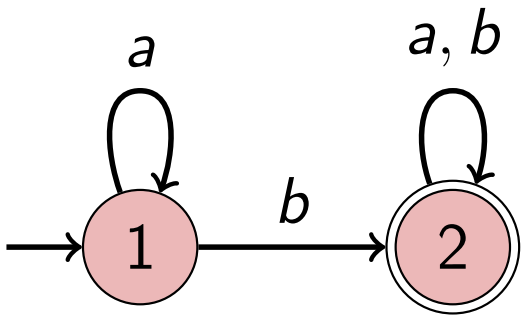
beskriver vad som händer när man matar in ett ord.

Den definieras rekursivt:

$$\begin{cases} \delta^*(S, \varepsilon) = S \\ \delta^*(S, \sigma w) = \delta^*(\delta(S, \sigma), w) \end{cases}$$

Om $\delta^*(S_1, w) = S_2$ så driver ordet w maskinen från S_1 till S_2

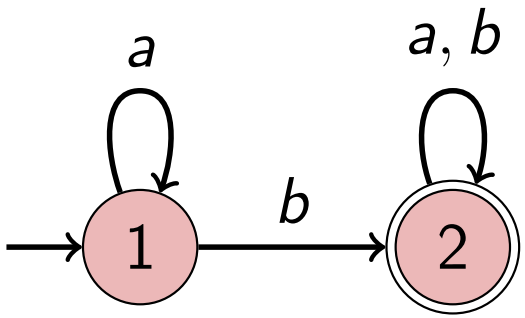
Exempel.



$$\begin{cases} \delta^*(S, \varepsilon) = S \\ \delta^*(S, \sigma w) = \delta^*(\delta(S, \sigma), w) \end{cases}$$

$$\delta^*(1, aba) =$$

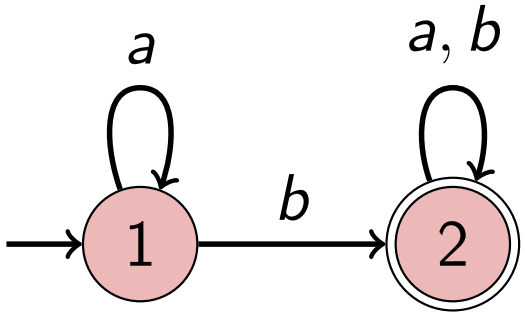
Exempel.



$$\begin{cases} \delta^*(S, \varepsilon) = S \\ \delta^*(S, \sigma w) = \delta^*(\delta(S, \sigma), w) \end{cases}$$

$$\delta^*(1, aba) = \delta^*(\delta(1, a), ba)$$

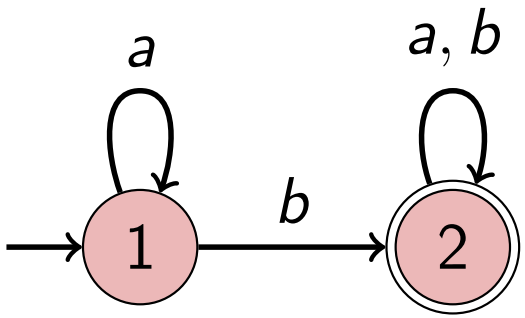
Exempel.



$$\begin{cases} \delta^*(S, \varepsilon) = S \\ \delta^*(S, \sigma w) = \delta^*(\delta(S, \sigma), w) \end{cases}$$

$$\delta^*(1, aba) = \delta^*(\delta(1, a), ba) = \delta^*(1, ba)$$

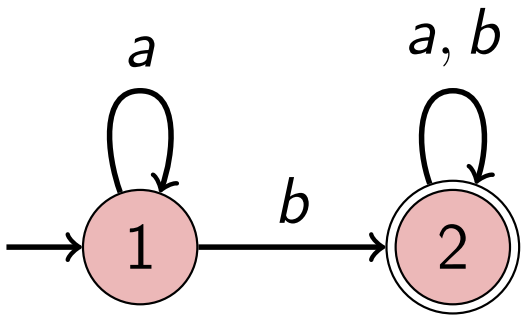
Exempel.



$$\begin{cases} \delta^*(S, \varepsilon) = S \\ \delta^*(S, \sigma w) = \delta^*(\delta(S, \sigma), w) \end{cases}$$

$$\begin{aligned} \delta^*(1, aba) &= \delta^*(\delta(1, a), ba) = \delta^*(1, ba) \\ &= \delta^*(\delta(1, b), a) \end{aligned}$$

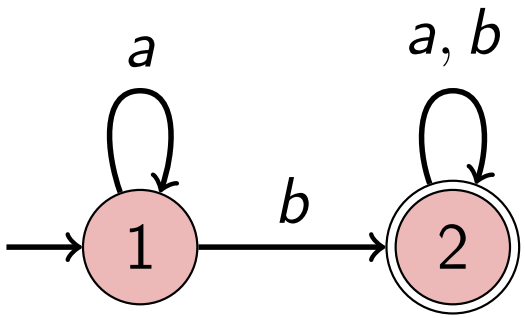
Exempel.



$$\begin{cases} \delta^*(S, \varepsilon) = S \\ \delta^*(S, \sigma w) = \delta^*(\delta(S, \sigma), w) \end{cases}$$

$$\begin{aligned} \delta^*(1, aba) &= \delta^*(\delta(1, a), ba) = \delta^*(1, ba) \\ &= \delta^*(\delta(1, b), a) = \delta^*(2, a) \end{aligned}$$

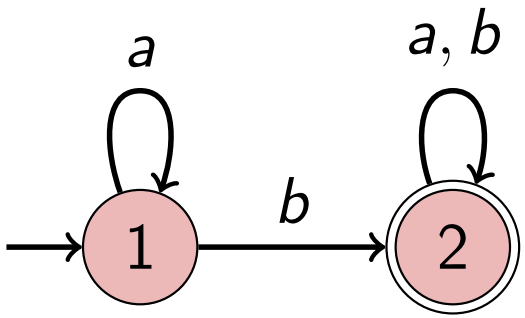
Exempel.



$$\begin{cases} \delta^*(S, \varepsilon) = S \\ \delta^*(S, \sigma w) = \delta^*(\delta(S, \sigma), w) \end{cases}$$

$$\begin{aligned} \delta^*(1, aba) &= \delta^*(\delta(1, a), ba) = \delta^*(1, ba) \\ &= \delta^*(\delta(1, b), a) = \delta^*(2, a) \\ &= \delta^*(\delta(2, a), \varepsilon) \end{aligned}$$

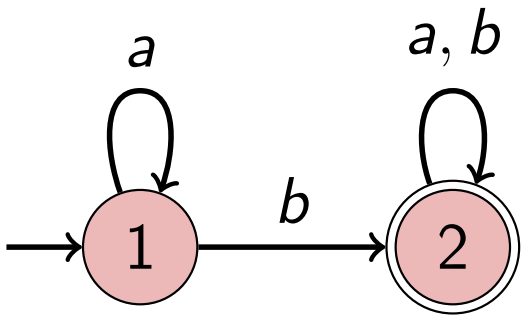
Exempel.



$$\begin{cases} \delta^*(S, \varepsilon) = S \\ \delta^*(S, \sigma w) = \delta^*(\delta(S, \sigma), w) \end{cases}$$

$$\begin{aligned} \delta^*(1, aba) &= \delta^*(\delta(1, a), ba) = \delta^*(1, ba) \\ &= \delta^*(\delta(1, b), a) = \delta^*(2, a) \\ &= \delta^*(\delta(2, a), \varepsilon) = \delta^*(2, \varepsilon) \end{aligned}$$

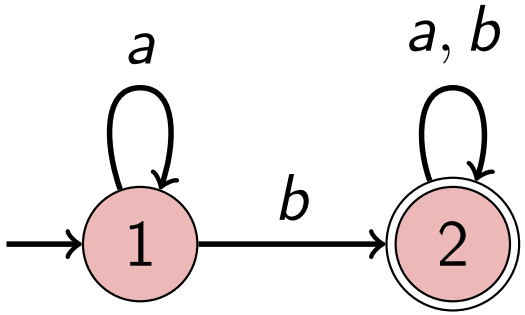
Exempel.



$$\begin{cases} \delta^*(S, \varepsilon) = S \\ \delta^*(S, \sigma w) = \delta^*(\delta(S, \sigma), w) \end{cases}$$

$$\begin{aligned} \delta^*(1, aba) &= \delta^*(\delta(1, a), ba) = \delta^*(1, ba) \\ &= \delta^*(\delta(1, b), a) = \delta^*(2, a) \\ &= \delta^*(\delta(2, a), \varepsilon) = \delta^*(2, \varepsilon) = 2 \end{aligned}$$

Exempel.



$$\begin{cases} \delta^*(S, \varepsilon) = S \\ \delta^*(S, \sigma w) = \delta^*(\delta(S, \sigma), w) \end{cases}$$

$$\begin{aligned} \delta^*(1, aba) &= \delta^*(\delta(1, a), ba) = \delta^*(1, ba) \\ &= \delta^*(\delta(1, b), a) = \delta^*(2, a) \\ &= \delta^*(\delta(2, a), \varepsilon) = \delta^*(2, \varepsilon) = 2 \end{aligned}$$

Eftersom δ^* är en **funktion** blir resultatet samma varje gång.
Det är därför maskinerna kallas **deterministiska**.

Sats 6.1.11. $\delta^*(S, wu) = \delta^*(\delta^*(S, w), u)$

Sats 6.1.11. $\delta^*(S, wu) = \delta^*(\delta^*(S, w), u)$

Definition 6.1.7. En DTM M har **språket**

$$L(M) = \{w \in \Sigma^* \mid \delta^*(S_0, w) \in F\}$$

det vill säga mängden av alla ord som accepteras.

Sats 6.1.11. $\delta^*(S, wu) = \delta^*(\delta^*(S, w), u)$

Definition 6.1.7. En DTM M har **språket**

$$L(M) = \{w \in \Sigma^* \mid \delta^*(S_0, w) \in F\}$$

det vill säga mängden av alla ord som accepteras.

M **avgör** ett språk L om $L = L(M)$.

Sats 6.1.11. $\delta^*(S, wu) = \delta^*(\delta^*(S, w), u)$

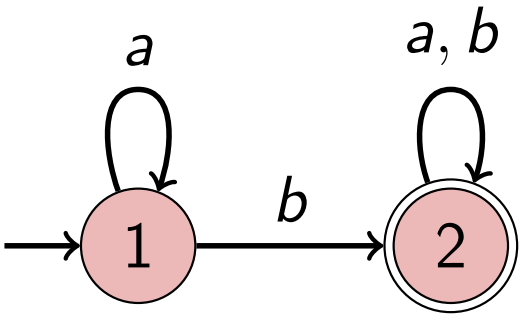
Definition 6.1.7. En DTM M har **språket**

$$L(M) = \{w \in \Sigma^* \mid \delta^*(S_0, w) \in F\}$$

det vill säga mängden av alla ord som accepteras.

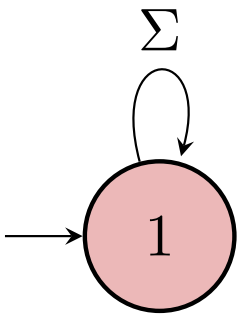
M **avgör** ett språk L om $L = L(M)$.

Exempel 6.1.8.



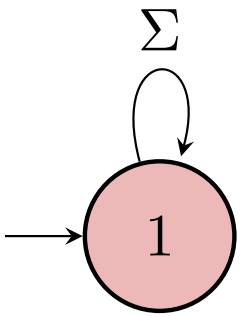
avgör språket $a^*b(a \cup b)^*$

Exempel 6.1.10.



Figur 6.6

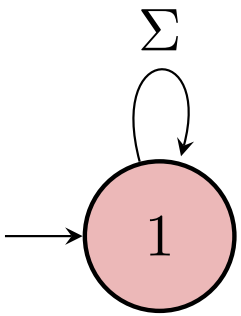
Exempel 6.1.10.



Figur 6.6

Accepterar inga ord

Exempel 6.1.10.

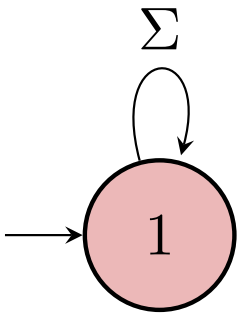


Figur 6.6

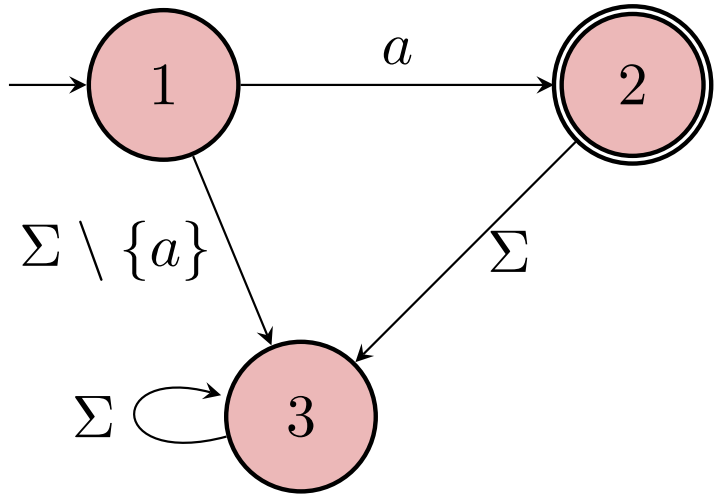
Accepterar inga ord

Avgör språket Ø

Exempel 6.1.10.



Figur 6.6

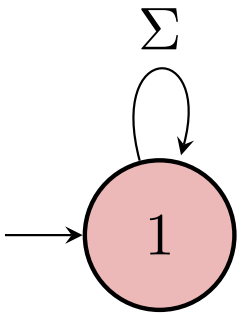


Figur 6.7

Accepterar inga ord

Avgör språket Ø

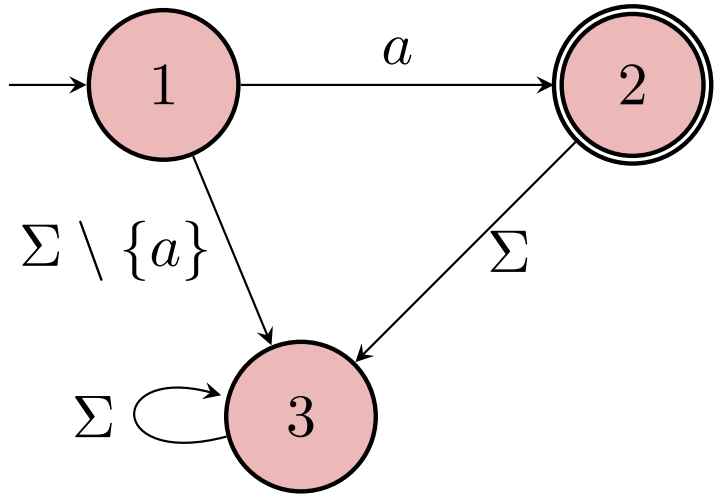
Exempel 6.1.10.



Figur 6.6

Accepterar inga ord

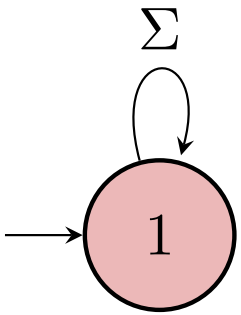
Avgör språket ∅



Figur 6.7

Accepterar bara ordet *a*

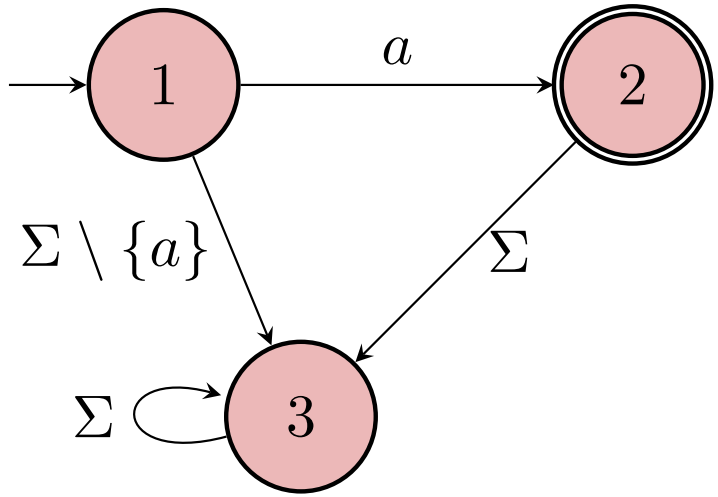
Exempel 6.1.10.



Figur 6.6

Accepterar inga ord

Avgör språket $\emptyset$



Figur 6.7

Accepterar bara ordet a

Avgör språket $\{a\}$

Kapitel 6.2 – ITM

Problem: Givet ett språk, konstruera en tillståndsmaskin som avgör språket

Kapitel 6.2 – ITM

Problem: Givet ett språk, konstruera en tillståndsmaskin som avgör språket

Detta blir lättare om vi släpper lite på reglerna för tillståndsmaskiner.

Kapitel 6.2 – ITM

Problem: Givet ett språk, konstruera en tillståndsmaskin som avgör språket

Detta blir lättare om vi släpper lite på reglerna för tillståndsmaskiner.

Leder till **icke-deterministiska tillståndsmaskiner** (ITM):

Kapitel 6.2 – ITM

Problem: Givet ett språk, konstruera en tillståndsmaskin som avgör språket

Detta blir lättare om vi släpper lite på reglerna för tillståndsmaskiner.

Leder till **icke-deterministiska tillståndsmaskiner** (ITM):

- Maskinen kan byta tillstånd **utan** att konsumera ett tecken. Kallas ε-övergång (tomma ordet).

Kapitel 6.2 – ITM

Problem: Givet ett språk, konstruera en tillståndsmaskin som avgör språket

Detta blir lättare om vi släpper lite på reglerna för tillståndsmaskiner.

Leder till **icke-deterministiska tillståndsmaskiner** (ITM):

- Maskinen kan byta tillstånd **utan** att konsumera ett tecken. Kallas ε-övergång (tomma ordet).
- Tillstånd kan ha flera olika övergångar för samma tecken. Behöver inte finnas övergångar för alla tecken – maskinen kan ”hänga sig”.

Kapitel 6.2 – ITM

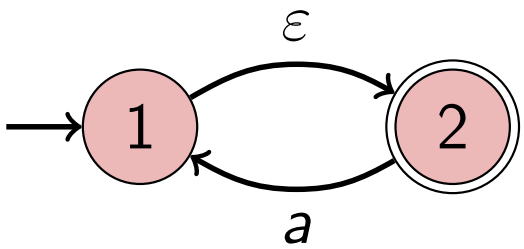
Problem: Givet ett språk, konstruera en tillståndsmaskin som avgör språket

Detta blir lättare om vi släpper lite på reglerna för tillståndsmaskiner.

Leder till **icke-deterministiska tillståndsmaskiner** (ITM):

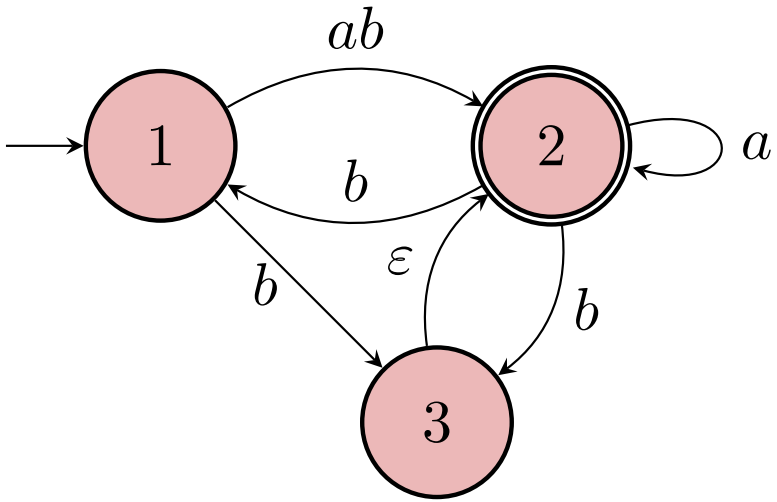
- Maskinen kan byta tillstånd **utan** att konsumera ett tecken. Kallas ε-övergång (tomma ordet).
- Tillstånd kan ha flera olika övergångar för samma tecken. Behöver inte finnas övergångar för alla tecken – maskinen kan ”hänga sig”.
- Det får finnas flera starttillstånd.

Exempel.



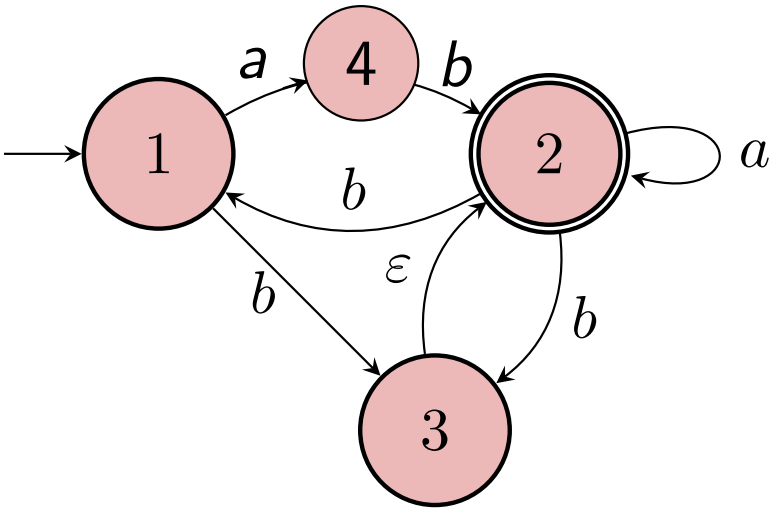
$$\Sigma = \{a, b\}$$

Exempel 6.2.1.



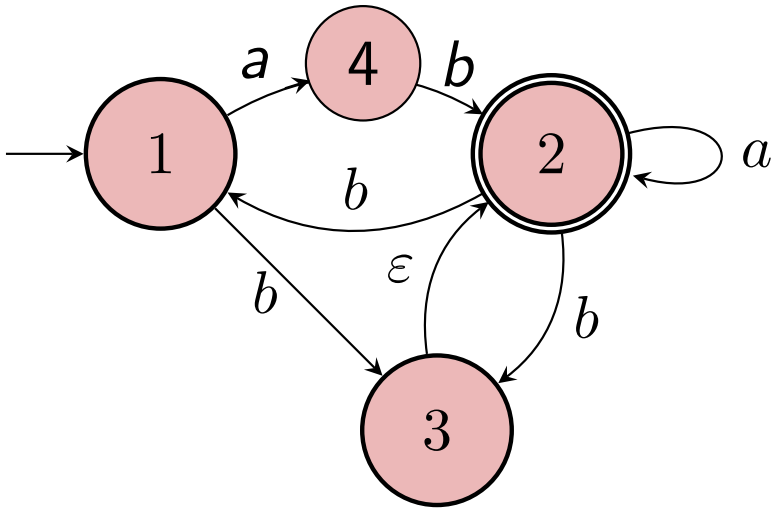
$$\Sigma = \{a, b\}$$

Exempel 6.2.1.



$$\Sigma = \{a, b\}$$

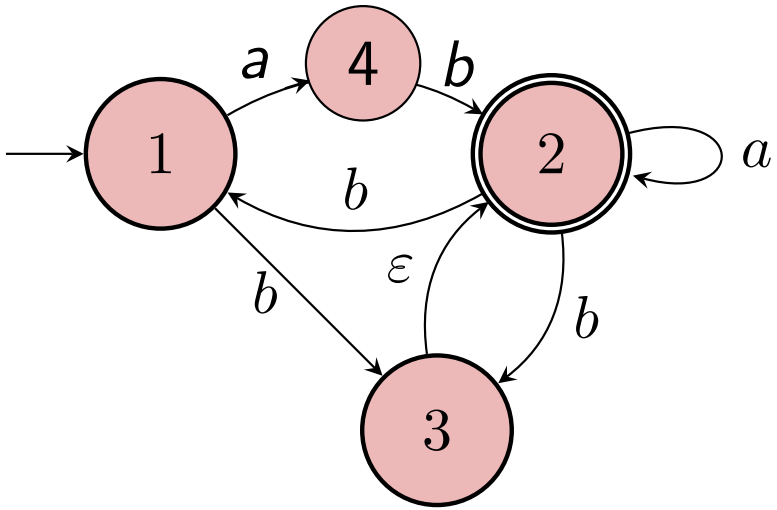
Exempel 6.2.1.



$\Sigma = \{a, b\}$

Mata in ordet *abbab*

Exempel 6.2.1.

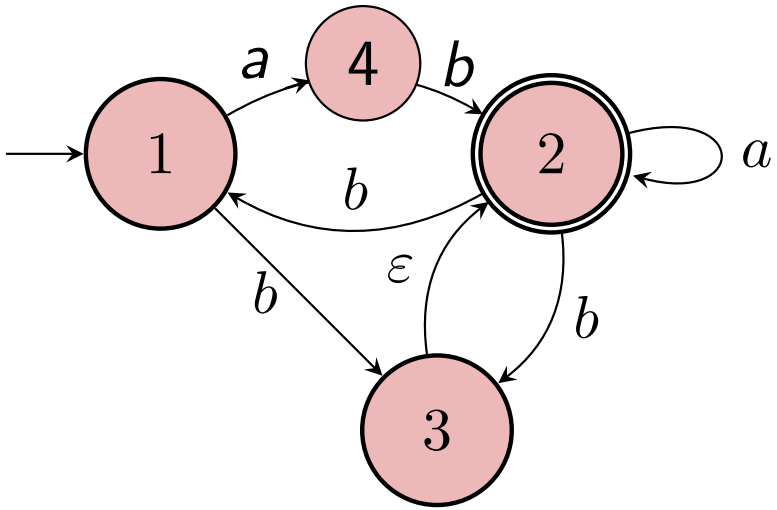


$\Sigma = \{a, b\}$

Mata in ordet *abbab*

Finns flera möjliga resultat – på tavlan!

Exempel 6.2.1.

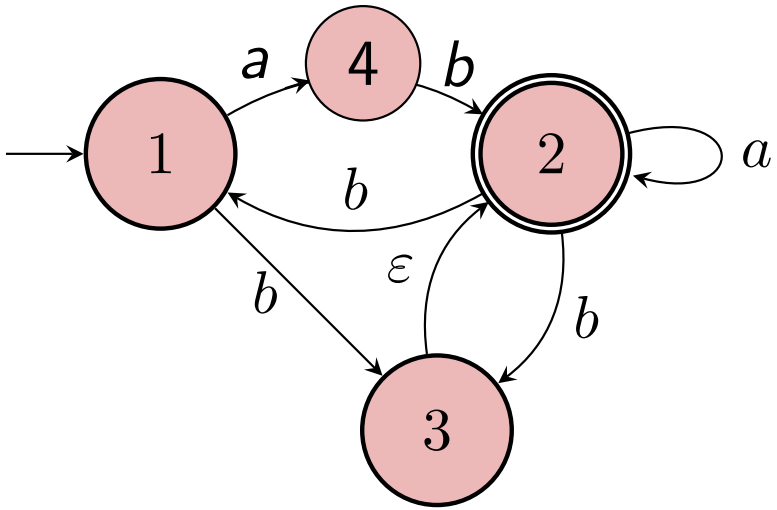


$\Sigma = \{a, b\}$

Mata in ordet *abbab*

Finns flera möjliga resultat – på tavlan!
abbab **accepteras** eftersom minst en möjlighet accepteras

Exempel 6.2.1.



$\Sigma = \{a, b\}$

Mata in ordet *abbab*

Finns flera möjliga resultat – på tavlan!
abbab **accepteras** eftersom minst en möjlighet accepteras

Obs: maskinen **hänger sig** om ett ord börjar på *aa* – accepteras ej

En ITM:s beräkning är inte entydig – det finns flera möjligheter.

En ITM:s beräkning är inte entydig – det finns flera möjligheter.

Språket för en ITM består av de ord som **kan** driva maskinen från ett starttillstånd till ett accepterande tillstånd.

En ITM:s beräkning är inte entydig – det finns flera möjligheter.

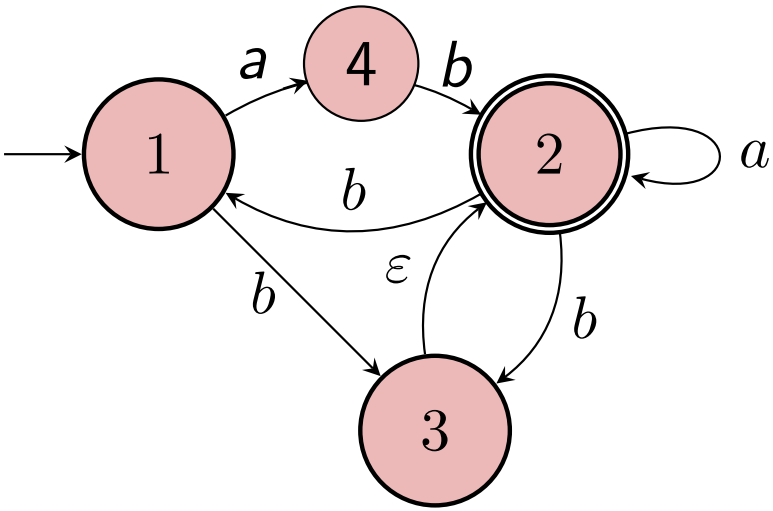
Språket för en ITM består av de ord som **kan** driva maskinen från ett starttillstånd till ett accepterande tillstånd.

Det räcker alltså att en möjlighet accepteras.

En ITM:s beräkning är inte entydig – det finns flera möjligheter.

Språket för en ITM består av de ord som **kan** driva maskinen från ett starttillstånd till ett accepterande tillstånd.

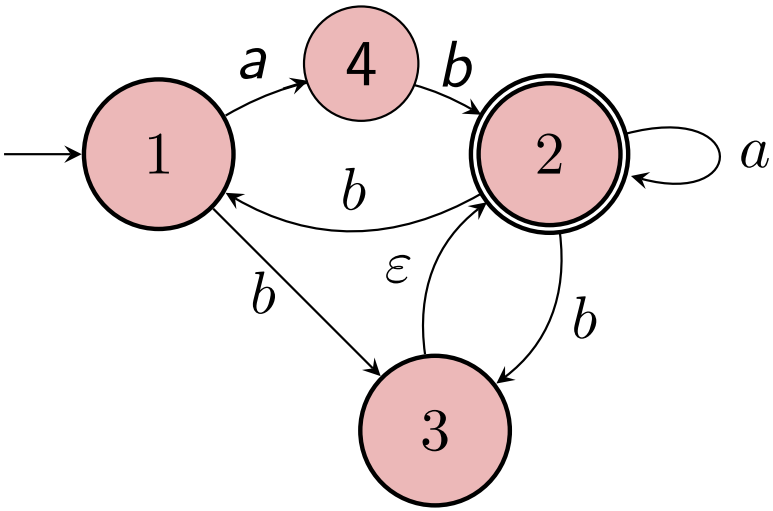
Det räcker alltså att en möjlighet accepteras.



En ITM:s beräkning är inte entydig – det finns flera möjligheter.

Språket för en ITM består av de ord som **kan** driva maskinen från ett starttillstånd till ett accepterande tillstånd.

Det räcker alltså att en möjlighet accepteras.



har språket $(b \cup ab)(a \cup b)^*$

Kapitel 6.3 – Delmängdskonstruktionen

Kapitel 6.3 – Delmängdskonstruktionen

Alla DTM:er är också en ITM, men det omvända gäller inte.

Kapitel 6.3 – Delmängdskonstruktionen

Alla DTM:er är också en ITM, men det omvända gäller inte.

Finns det språk som bara ITM:er kan avgöra?

Kapitel 6.3 – Delmängdskonstruktionen

Alla DTM:er är också en ITM, men det omvända gäller inte.

Finns det språk som bara ITM:er kan avgöra?

Nej!

Kapitel 6.3 – Delmängdskonstruktionen

Alla DTM:er är också en ITM, men det omvända gäller inte.

Finns det språk som bara ITM:er kan avgöra?

Nej! För varje ITM kan man konstruera en DTM som avgör samma språk!

Kapitel 6.3 – Delmängdskonstruktionen

Alla DTM:er är också en ITM, men det omvända gäller inte.

Finns det språk som bara ITM:er kan avgöra?

Nej! För varje ITM kan man konstruera en DTM som avgör samma språk!

Metoden för att göra detta kallas **delmängdskonstruktionen**

Definition.
Om w är ett ord och X en mängd tillstånd i en ITM,

Definition.
Om w är ett ord och X en mängd tillstånd i en ITM, låt $w(X)$ vara mängden av tillstånd som kan nås från ett tillstånd i X genom att mata in ordet w.

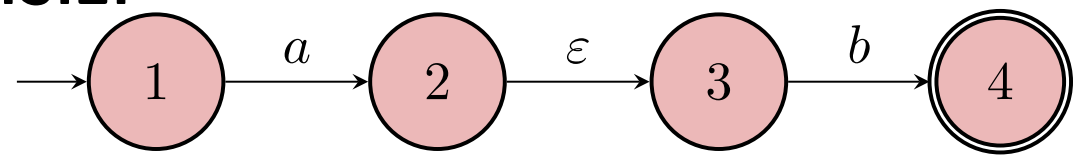
Definition.
Om w är ett ord och X en mängd tillstånd i en ITM, låt $w(X)$ vara mängden av tillstånd som kan nås från ett tillstånd i X genom att mata in ordet w.

($w(X)$ kallas "w-tillslutningen av X")

Definition.
Om w är ett ord och X en mängd tillstånd i en ITM, låt $w(X)$ vara mängden av tillstånd som kan nås från ett tillstånd i X genom att mata in ordet w.

($w(X)$ kallas "w-tillslutningen av X")

Exempel 6.3.2.

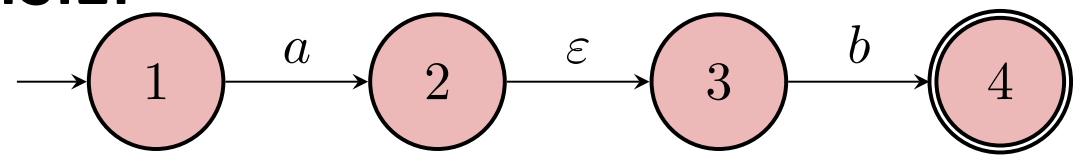


Figur 6.15

Definition.
Om w är ett ord och X en mängd tillstånd i en ITM, låt $w(X)$ vara mängden av tillstånd som kan nås från ett tillstånd i X genom att mata in ordet w.

($w(X)$ kallas "w-tillslutningen av X")

Exempel 6.3.2.



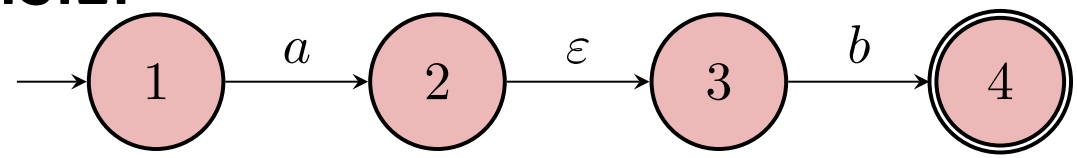
Figur 6.15

- $a(\{1\}) =$

Definition.
Om w är ett ord och X en mängd tillstånd i en ITM, låt $w(X)$ vara mängden av tillstånd som kan nås från ett tillstånd i X genom att mata in ordet w.

($w(X)$ kallas "w-tillslutningen av X")

Exempel 6.3.2.



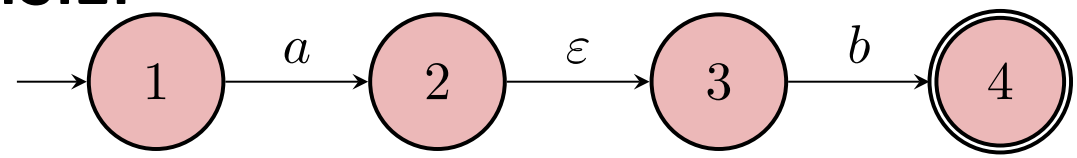
Figur 6.15

- $a(\{1\}) = \{2, 3\}$

Definition.
Om w är ett ord och X en mängd tillstånd i en ITM, låt $w(X)$ vara mängden av tillstånd som kan nås från ett tillstånd i X genom att mata in ordet w.

($w(X)$ kallas "w-tillslutningen av X")

Exempel 6.3.2.



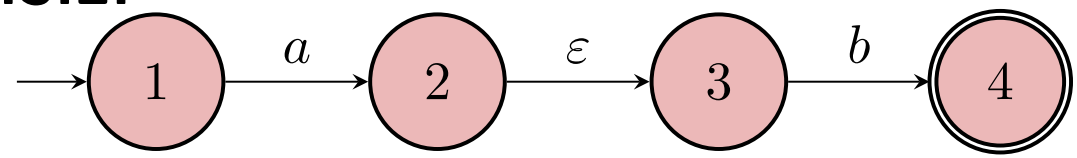
Figur 6.15

- $a(\{1\}) = \{2, 3\}$
- $b(\{1\}) =$

Definition.
Om w är ett ord och X en mängd tillstånd i en ITM, låt $w(X)$ vara mängden av tillstånd som kan nås från ett tillstånd i X genom att mata in ordet w.

($w(X)$ kallas "w-tillslutningen av X")

Exempel 6.3.2.



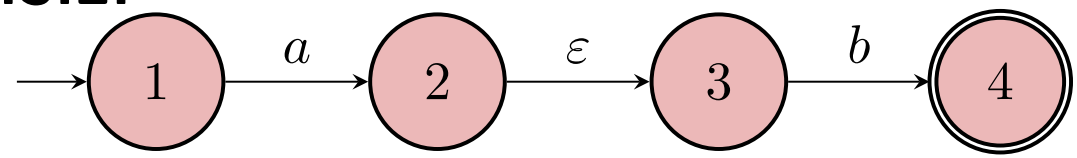
Figur 6.15

- $a(\{1\}) = \{2, 3\}$
- $b(\{1\}) = \emptyset$

Definition.
Om w är ett ord och X en mängd tillstånd i en ITM, låt $w(X)$ vara mängden av tillstånd som kan nås från ett tillstånd i X genom att mata in ordet w.

($w(X)$ kallas "w-tillslutningen av X")

Exempel 6.3.2.



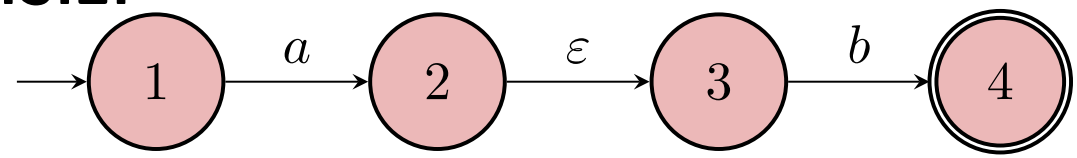
Figur 6.15

- $a(\{1\}) = \{2, 3\}$
- $b(\{1\}) = \emptyset$ men $b(\{1, 2, 3\}) =$

Definition.
Om w är ett ord och X en mängd tillstånd i en ITM, låt $w(X)$ vara mängden av tillstånd som kan nås från ett tillstånd i X genom att mata in ordet w.

($w(X)$ kallas "w-tillslutningen av X")

Exempel 6.3.2.



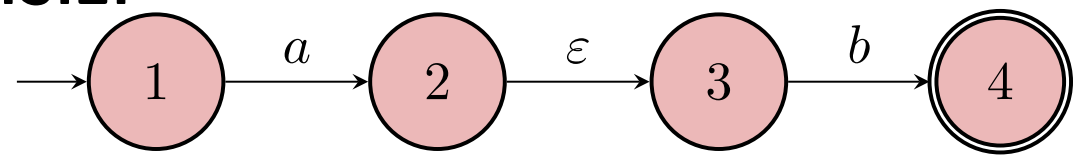
Figur 6.15

- $a(\{1\}) = \{2, 3\}$
- $b(\{1\}) = \emptyset$ men $b(\{1, 2, 3\}) = \{4\}$

Definition.
Om w är ett ord och X en mängd tillstånd i en ITM, låt $w(X)$ vara mängden av tillstånd som kan nås från ett tillstånd i X genom att mata in ordet w.

($w(X)$ kallas "w-tillslutningen av X")

Exempel 6.3.2.



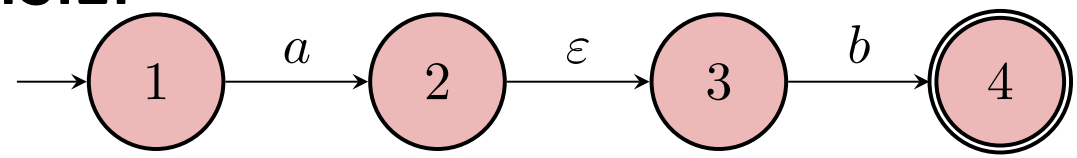
Figur 6.15

- $a(\{1\}) = \{2, 3\}$
- $b(\{1\}) = \emptyset$ men $b(\{1, 2, 3\}) = \{4\}$
- $ab(\{1\}) =$

Definition.
Om w är ett ord och X en mängd tillstånd i en ITM, låt $w(X)$ vara mängden av tillstånd som kan nås från ett tillstånd i X genom att mata in ordet w.

($w(X)$ kallas "w-tillslutningen av X")

Exempel 6.3.2.



Figur 6.15

- $a(\{1\}) = \{2, 3\}$
- $b(\{1\}) = \emptyset$ men $b(\{1, 2, 3\}) = \{4\}$
- $ab(\{1\}) = \{4\}$

Delmängdskonstruktionen.

Varje tillstånd i den konstruerade DTM:en kommer bestå av en **delmängd** av tillstånd i den ursprungliga ITM:en

Delmängdskonstruktionen (Algoritm 6.3.4).

Delmängdskonstruktionen (Algoritm 6.3.4).

1. Starttillståndet i DTM:en är $\varepsilon(S_0)$

Delmängdskonstruktionen (Algoritm 6.3.4).

1. Starttillståndet i DTM:en är $\varepsilon(S_0)$
2. Välj ett tillstånd X som saknar övergångar i DTM:en

Delmängdskonstruktionen (Algoritm 6.3.4).

1. Starttillståndet i DTM:en är $\varepsilon(S_0)$
2. Välj ett tillstånd X som saknar övergångar i DTM:en
3. För varje σ i alfabetet, beräkna $\sigma(X)$.
   Om $\sigma(X)$ saknas i DTM:en, lägg till det.

Delmängdskonstruktionen (Algoritm 6.3.4).

1. Starttillståndet i DTM:en är $\varepsilon(S_0)$
2. Välj ett tillstånd X som saknar övergångar i DTM:en
3. För varje σ i alfabetet, beräkna $\sigma(X)$.
   Om $\sigma(X)$ saknas i DTM:en, lägg till det.
4. Lägg till en σ-övergång från X till $\sigma(X)$

Delmängdskonstruktionen (Algoritm 6.3.4).

1. Starttillståndet i DTM:en är $\varepsilon(S_0)$
2. Välj ett tillstånd X som saknar övergångar i DTM:en
3. För varje σ i alfabetet, beräkna $\sigma(X)$.
   Om $\sigma(X)$ saknas i DTM:en, lägg till det.
4. Lägg till en σ-övergång från X till $\sigma(X)$
5. Upprepa steg 1–4 tills alla tillstånd har övergångar

Delmängdskonstruktionen (Algoritm 6.3.4).

1. Starttillståndet i DTM:en är $\varepsilon(S_0)$
2. Välj ett tillstånd X som saknar övergångar i DTM:en
3. För varje σ i alfabetet, beräkna $\sigma(X)$.
   Om $\sigma(X)$ saknas i DTM:en, lägg till det.
4. Lägg till en σ-övergång från X till $\sigma(X)$
5. Upprepa steg 1–4 tills alla tillstånd har övergångar

Alla tillstånd som innehåller ett accepterande tillstånd från ITM:en blir accepterande i DTM:en.

Delmängdskonstruktionen (Algoritm 6.3.4).

1. Starttillståndet i DTM:en är $\varepsilon(S_0)$
2. Välj ett tillstånd X som saknar övergångar i DTM:en
3. För varje σ i alfabetet, beräkna $\sigma(X)$.
   Om $\sigma(X)$ saknas i DTM:en, lägg till det.
4. Lägg till en σ-övergång från X till $\sigma(X)$
5. Upprepa steg 1–4 tills alla tillstånd har övergångar

Alla tillstånd som innehåller ett accepterande tillstånd från ITM:en blir accepterande i DTM:en.

Exempel 6.3.5. På tavlan!

Sats 6.3.7. För varje tillstånd X i delmängdskonstruktionen gäller $\delta^*(X, w) = w(X)$

Sats 6.3.8. Delmängdskonstruktionen av en ITM avgör samma språk som den ursprungliga maskinen.

Stockholms matematiska cirkel
Datorernas matematik

www.math-stockholm.se/cirkel

16.00–16.15: Fika
16.15–17.15: Föreläsning om kapitel 6
17.15–17.30: Rast
17.30–18.00: Utbildningsinformation