

DD1361

Programmeringsparadigm

Formella Språk & Syntaxanalys  
Föreläsning 2

Per Austrin

2015-11-06

# Snabb repetition

Tre huvudkoncept förra gången:

## *Formellt språk*

- en mängd strängar
- t.ex. “obligatoriska progp-labbarna”, “binära strängar med udda antal ettor”, “syntaktiskt korrekta java-program”

## *Reguljärt uttryck*

- ett “verktyg” för att beskriva ett formellt språk
- i praktiken: *regex*, har extra funktionalitet som t.ex. *bakåtreferenser* (skilj på “*regex*” och “*reguljära uttryck*”!)

## *Ändlig automat, DFA*

- en abstrakt “maskin” för att beskriva ett formellt språk,
- en sorts algoritm för att *känna igen* ett formellt språk

Idag

Reguljära uttryck vs automater

Reguljära språk, begränsningar

Grammatiker

Stackautomater

Idag

## Reguljära uttryck vs automater

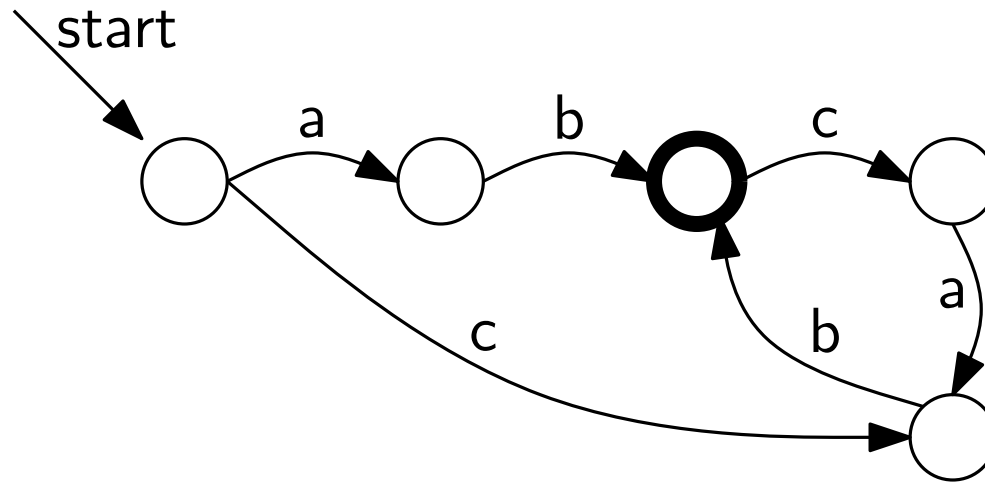
Reguljära språk, begränsningar

Grammatiker

Stackautomater

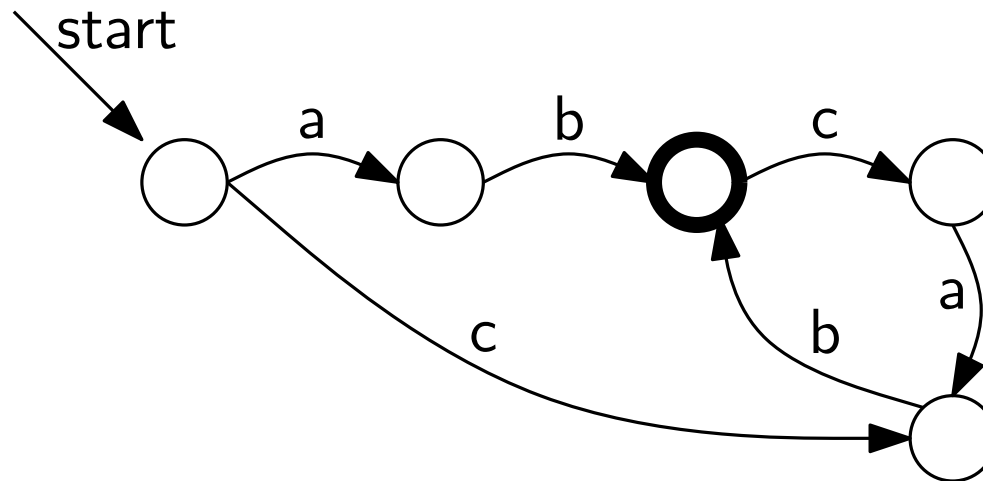
# Automat till reguljärt uttryck

Förra gången tittade vi på den här automaten



# Automat till reguljärt uttryck

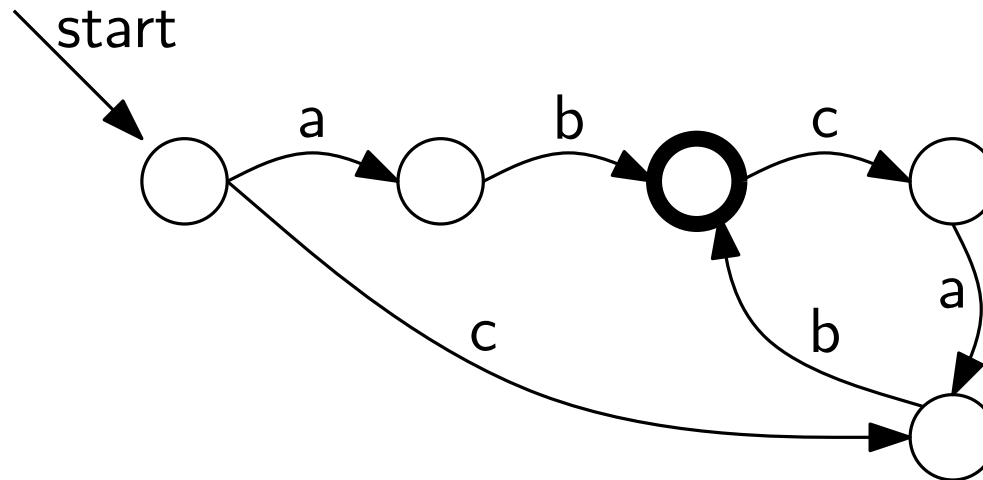
Förra gången tittade vi på den här automaten



Kan vi beskriva samma formella språk med ett reguljärt uttryck?

# Automat till reguljärt uttryck

Förra gången tittade vi på den här automaten



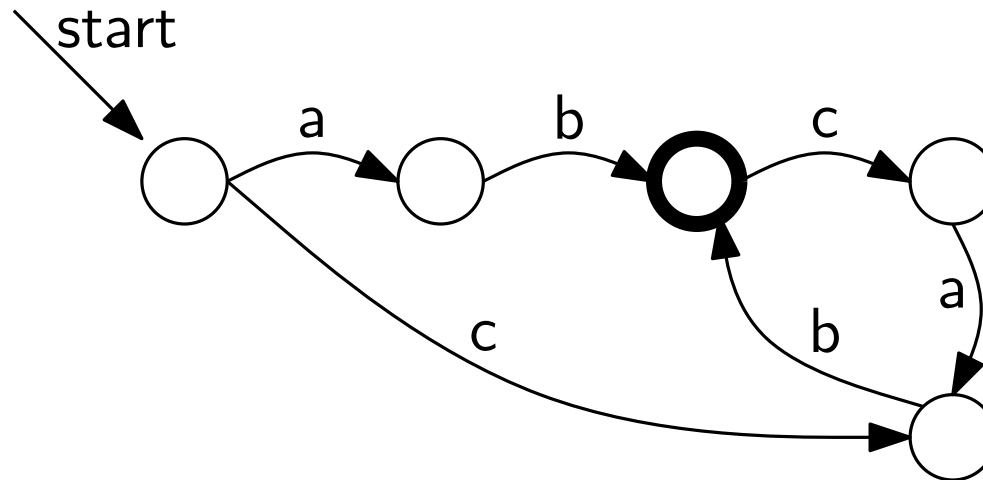
Kan vi beskriva samma formella språk med ett reguljärt uttryck?

Strängarna vi vill acceptera är:

1. Börja med antingen **ab** eller **cb** för att komma till accept-tillstånd
2. Sedan valfritt antal **cab** (varv i loopen)

# Automat till reguljärt uttryck

Förra gången tittade vi på den här automaten



Kan vi beskriva samma formella språk med ett reguljärt uttryck?

Strängarna vi vill acceptera är:

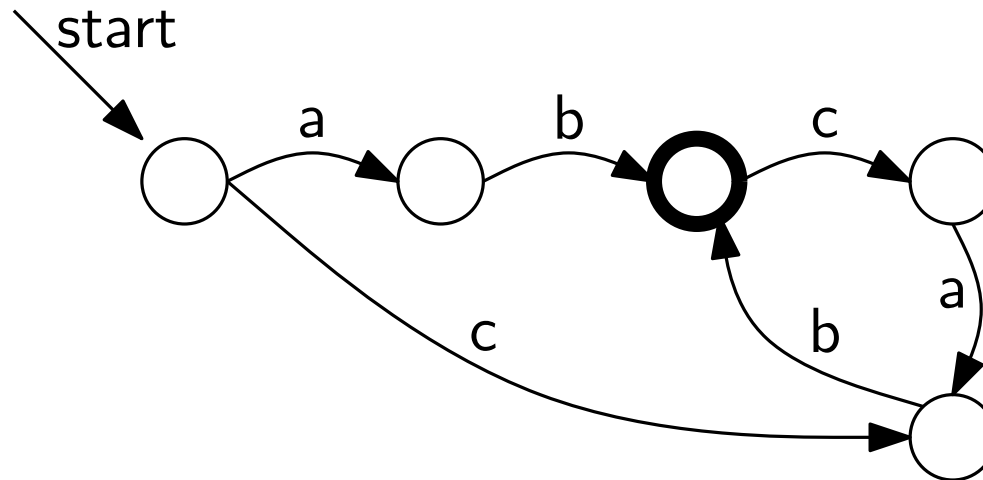
1. Börja med antingen **ab** eller **cb** för att komma till accept-tillstånd
2. Sedan valfritt antal **cab** (varv i loopen)

Reguljärt uttryck:  **$(ab|cb)(cab)^*$**



# Automat till reguljärt uttryck

Förra gången tittade vi på den här automaten



Kan vi beskriva samma formella språk med ett reguljärt uttryck?

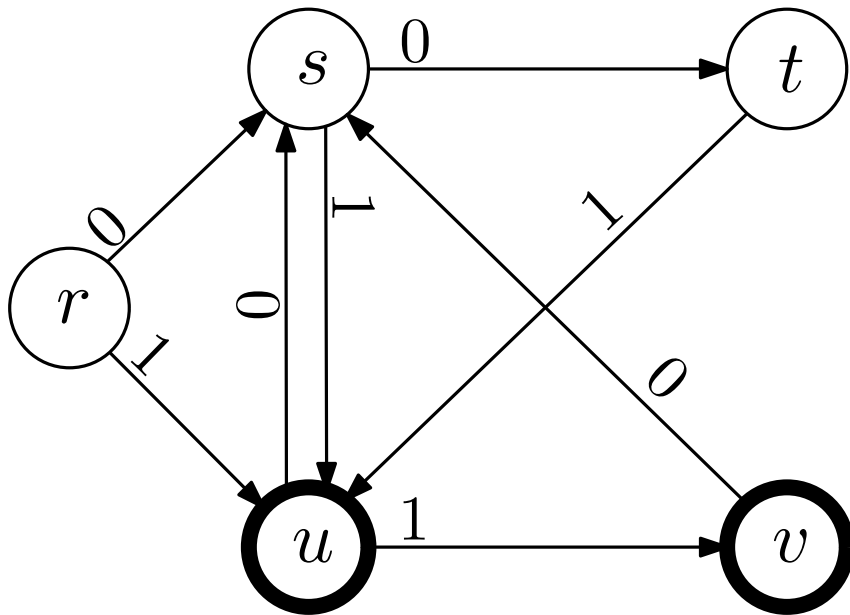
Strängarna vi vill acceptera är:

1. Börja med antingen **ab** eller **cb** för att komma till accept-tillstånd
2. Sedan valfritt antal **cab** (varv i loopen)

Reguljärt uttryck:  **$(ab|cb)(cab)^*$**

Mer kompakt:  **$[ac]b(cab)^*$**

# Från automater till reguljära uttryck

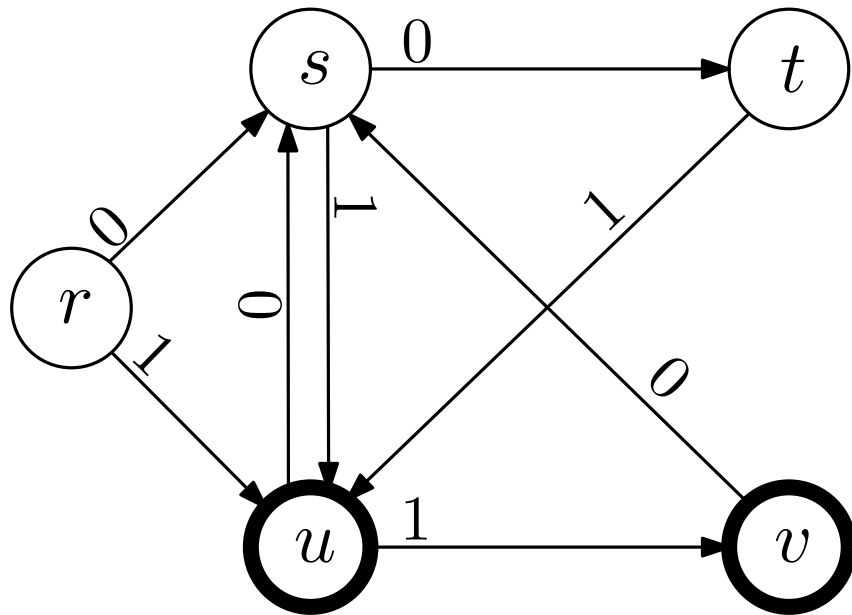


Starttillstånd  $r$

Accepterande tillstånd  $u$  och  $v$

Kan vi skriva ett reguljärt uttryck för den här automaten?

# Från automater till reguljära uttryck



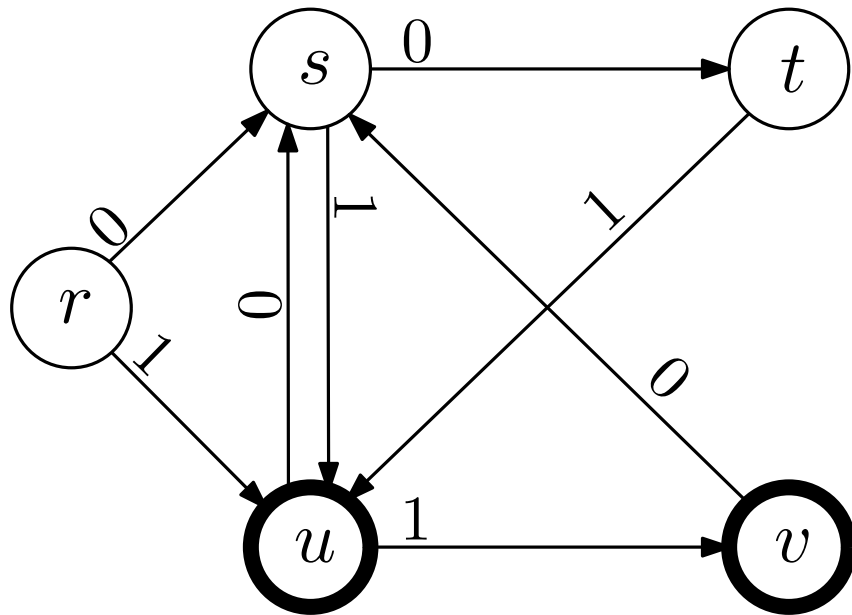
Starttillstånd  $r$

Accepterande tillstånd  $u$  och  $v$

Kan vi skriva ett reguljärt uttryck för den här automaten?

Steg 1: förstå vilka strängar som automaten känner igen

# Från automater till reguljära uttryck



Starttillstånd  $r$

Accepterande tillstånd  $u$  och  $v$

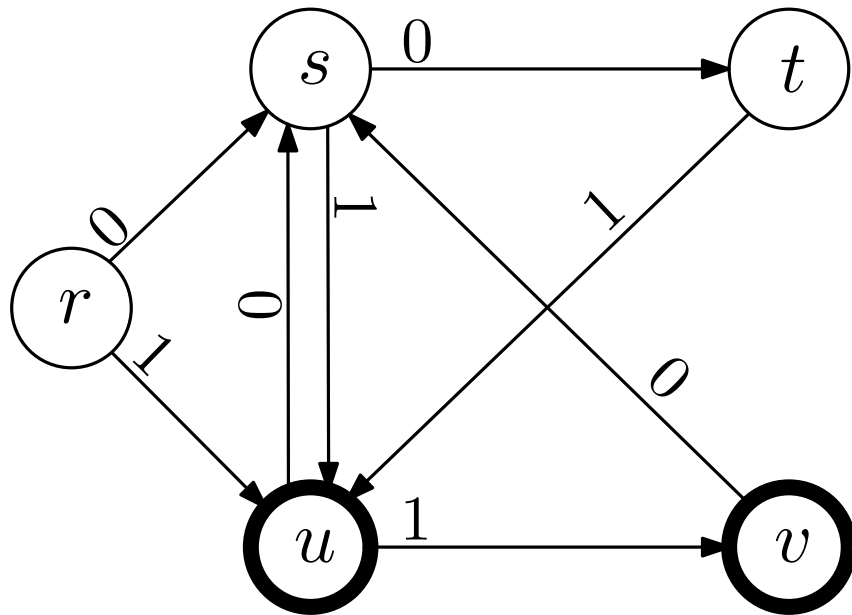
Kan vi skriva ett reguljärt uttryck för den här automaten?

Steg 1: förstå vilka strängar som automaten känner igen

Ungefärlig tankeprocess (repetition från förra föreläsningen):

1.  $s$  och  $t$  kommer man bara till när man läst en nolla.
2.  $u$  och  $v$  kommer man bara till när man läst en etta
3.  $t$  resp.  $v$  kommer man bara till när man läst två nollor resp. ettor i rad, och får då inte läsa en tredje

# Från automater till reguljära uttryck



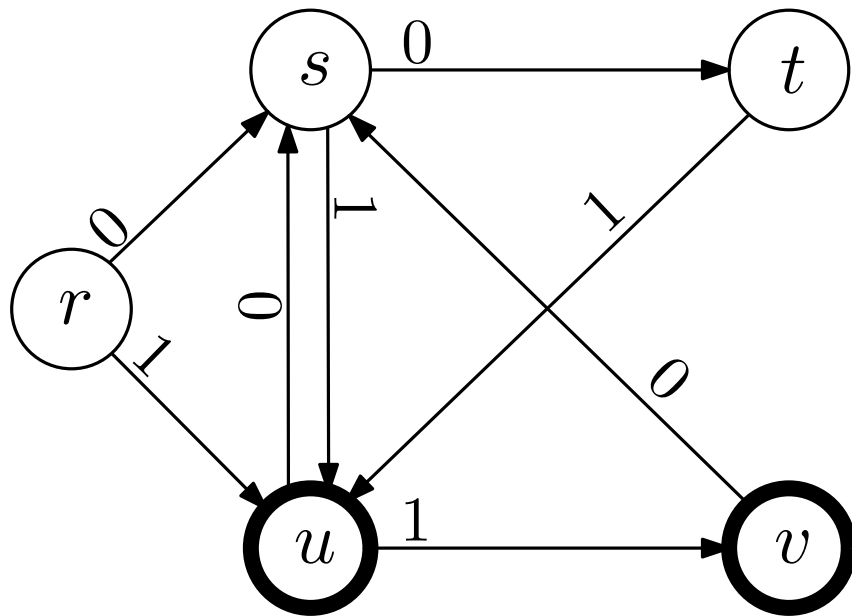
Starttillstånd  $r$

Accepterande tillstånd  $u$  och  $v$

Kan vi skriva ett reguljärt uttryck för den här automaten?

Steg 1: förstå vilka strängar som automaten känner igen  
...alla strängar där det aldrig förekommer mer än två nollor/ettor i rad, och som slutar på en etta

# Från automater till reguljära uttryck



Starttillstånd  $r$

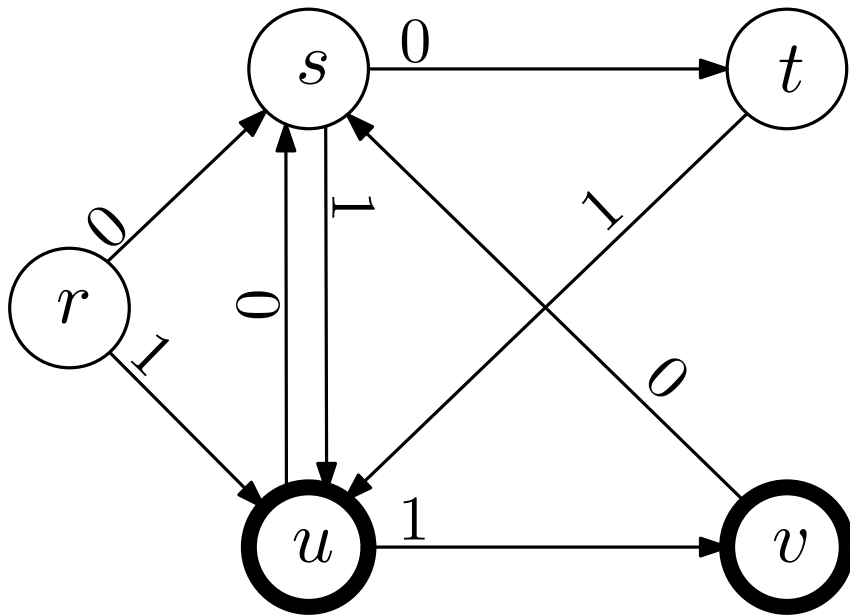
Accepterande tillstånd  $u$  och  $v$

Kan vi skriva ett reguljärt uttryck för den här automaten?

Steg 1: förstå vilka strängar som automaten känner igen  
...alla strängar där det aldrig förekommer mer än två nollor/ettor i rad, och som slutar på en etta

Steg 2, enklare variant: tillåt inte ens två nollor/ettor i rad

# Från automater till reguljära uttryck



Starttillstånd  $r$

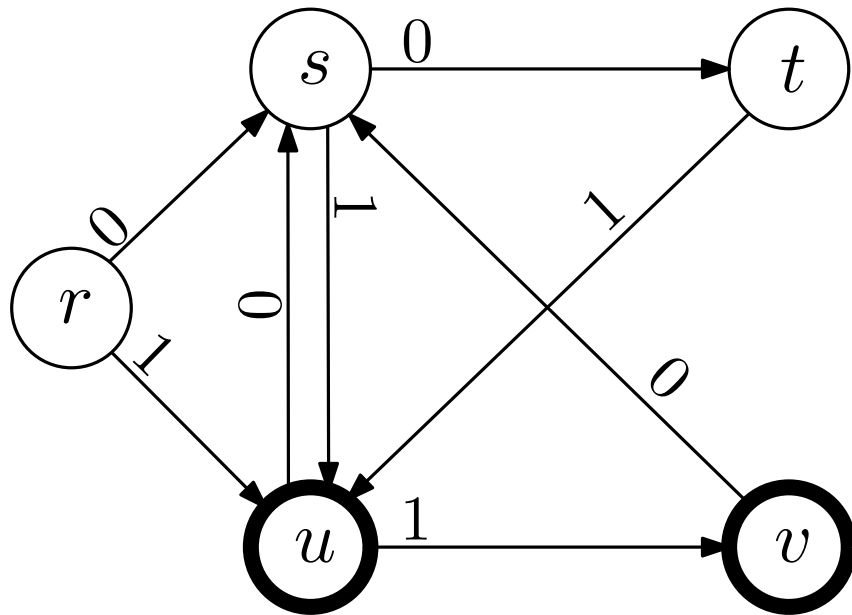
Accepterande tillstånd  $u$  och  $v$

Kan vi skriva ett reguljärt uttryck för den här automaten?

Steg 1: förstå vilka strängar som automaten känner igen  
...alla strängar där det aldrig förekommer mer än två nollor/ettor i rad, och som slutar på en etta

Steg 2, enklare variant: tillåt inte ens två nollor/ettor i rad  
 $0?(10)^*1$

# Från automater till reguljära uttryck



Starttillstånd  $r$

Accepterande tillstånd  $u$  och  $v$

Kan vi skriva ett reguljärt uttryck för den här automaten?

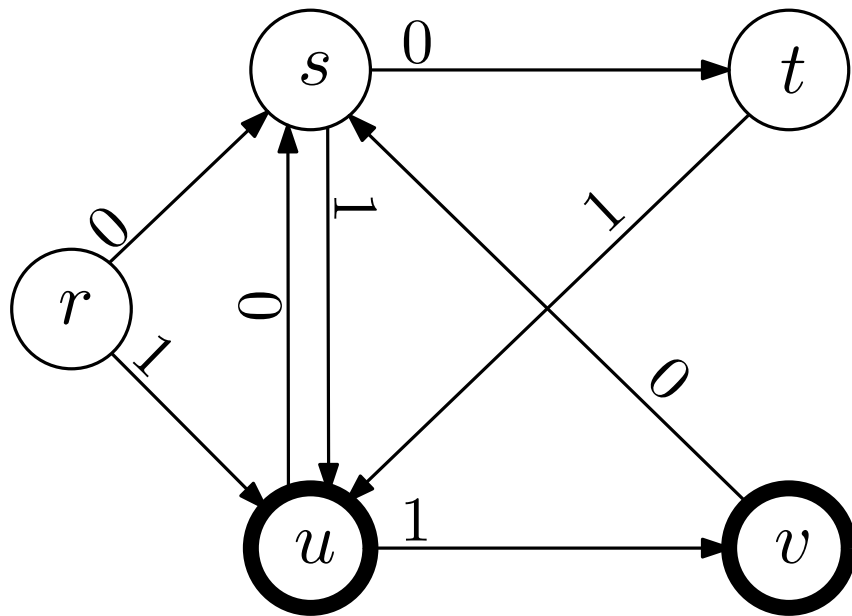
Steg 1: förstå vilka strängar som automaten känner igen  
...alla strängar där det aldrig förekommer mer än två nollor/ettor i rad, och som slutar på en etta

Steg 2, enklare variant: tillåt inte ens två nollor/ettor i rad  
 $0?(10)*1$

Steg 3: tillåt varje tecken att upprepas en gång. Byt ut 0 mot  $0?0$  och liknande för 1



# Från automater till reguljära uttryck



Starttillstånd  $r$

Accepterande tillstånd  $u$  och  $v$

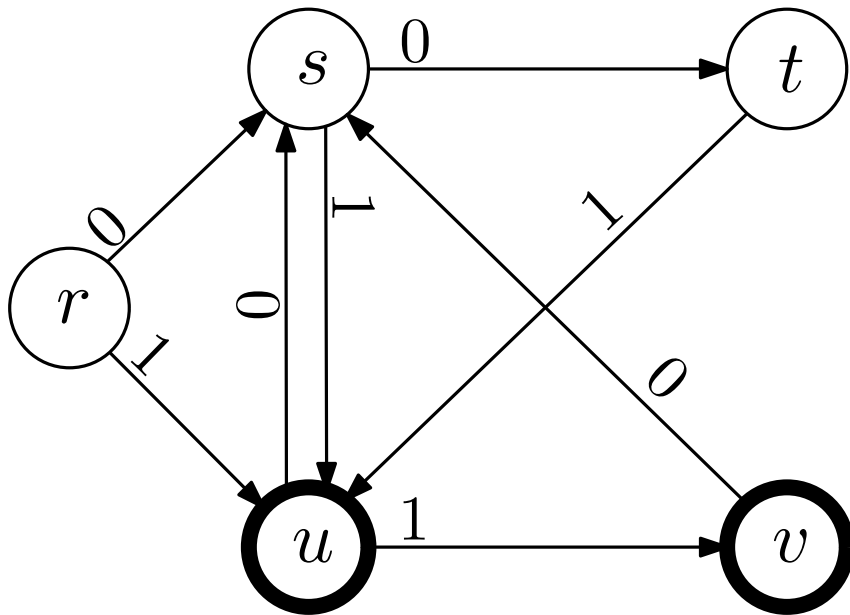
Kan vi skriva ett reguljärt uttryck för den här automaten?

Steg 1: förstå vilka strängar som automaten känner igen  
...alla strängar där det aldrig förekommer mer än två nollor/ettor i rad, och som slutar på en etta

Steg 2, enklare variant: tillåt inte ens två nollor/ettor i rad  
 $0?(10)*1$

Steg 3: tillåt varje tecken att upprepas en gång. Byt ut 0 mot  $0?0$  och liknande för 1  
 $0?0?(1?10?0)*1?1$

# Från automater till reguljära uttryck



Starttillstånd  $r$

Accepterande tillstånd  $u$  och  $v$

Kan vi skriva ett reguljärt uttryck för den här automaten?

Lösning:

$0?0?(1?10?0)*1?1$

# Automater till reguljära uttryck

Våra konstruktioner av reguljära uttryck från automater har varit väldigt ad hoc och inte det minsta metodiska.

# Automater till reguljära uttryck

Våra konstruktioner av reguljära uttryck från automater har varit väldigt ad hoc och inte det minsta metodiska.

Kan man alltid konvertera en automat till ett reguljärt uttryck?

# Automater till reguljära uttryck

Våra konstruktioner av reguljära uttryck från automater har varit väldigt ad hoc och inte det minsta metodiska.

Kan man alltid konvertera en automat till ett reguljärt uttryck?

Förvånande(?): **Ja, det kan man!**

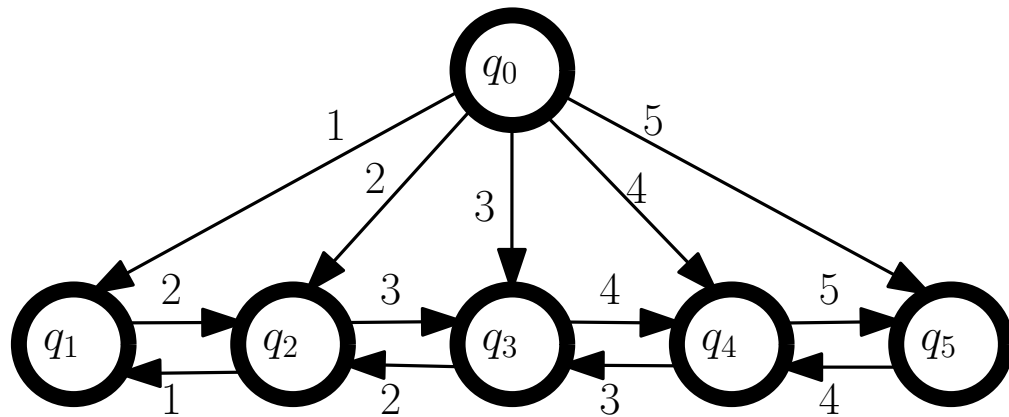
# Automater till reguljära uttryck

Våra konstruktioner av reguljära uttryck från automater har varit väldigt ad hoc och inte det minsta metodiska.

Kan man alltid konvertera en automat till ett reguljärt uttryck?

Förvånande(?): **Ja, det kan man!**

...men det reguljära uttrycket kan bli väldigt komplicerat



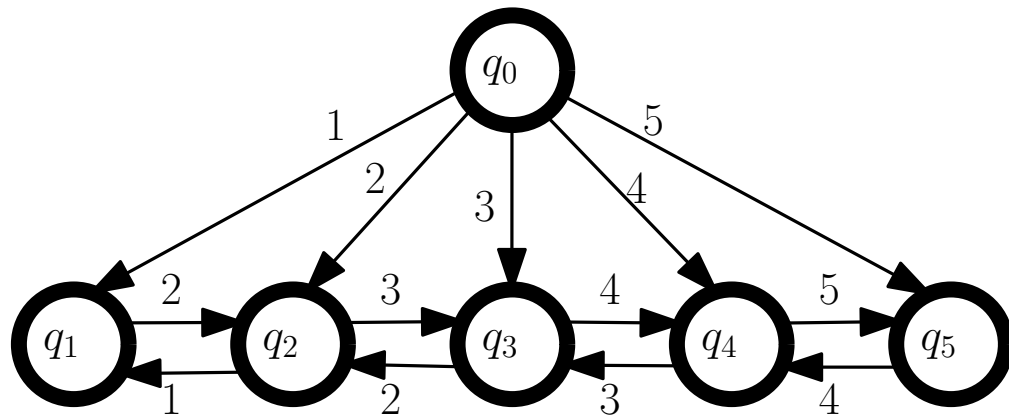
# Automater till reguljära uttryck

Våra konstruktioner av reguljära uttryck från automater har varit väldigt ad hoc och inte det minsta metodiska.

Kan man alltid konvertera en automat till ett reguljärt uttryck?

Förvånande(?): **Ja, det kan man!**

...men det reguljära uttrycket kan bli väldigt komplicerat



Alla strängar på tecknen  $\{1, 2, 3, 4, 5\}$  där varje siffra diffar med exakt 1 från föregående.

Reguljärt uttryck för samma språk?

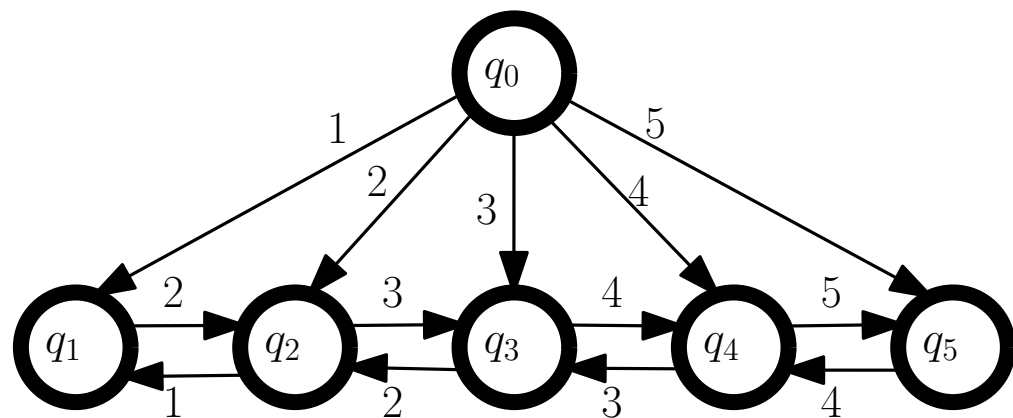
# Automater till reguljära uttryck

Våra konstruktioner av reguljära uttryck från automater har varit väldigt ad hoc och inte det minsta metodiska.

Kan man alltid konvertera en automat till ett reguljärt uttryck?

Förvånande(?): **Ja, det kan man!**

...men det reguljära uttrycket kan bli väldigt komplicerat



Alla strängar på tecknen  $\{1, 2, 3, 4, 5\}$  där varje siffra diffar med exakt 1 från föregående.

Reguljärt uttryck för samma språk?

Jag har ingen aning, men jag vet att det existerar...



# Reguljära uttryck till automater

Reguljärt uttryck från förra föreläsningen för lokal del av e-post-adress:

$[A-Za-z0-9] ([A-Za-z0-9.]^* [A-Za-z0-9]) ?$

# Reguljära uttryck till automater

Reguljärt uttryck från förra föreläsningen för lokal del av e-post-adress:

$[A-Za-z0-9] ([A-Za-z0-9.]^* [A-Za-z0-9]) ?$

Kan vi konstruera en automat som känner igen samma språk?

(kom ihåg: "känna igen samma språk" = "beskriver samma mängd strängar")

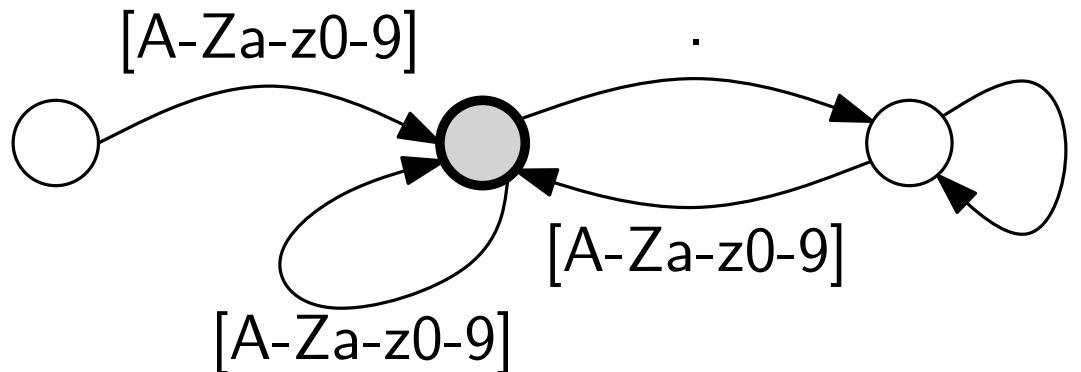
# Reguljära uttryck till automater

Reguljärt uttryck från förra föreläsningen för lokal del av e-post-adress:

$[A-Za-z0-9] ([A-Za-z0-9.]^* [A-Za-z0-9]) ?$

Kan vi konstruera en automat som känner igen samma språk?

(kom ihåg: "känna igen samma språk" = "beskriver samma mängd strängar")



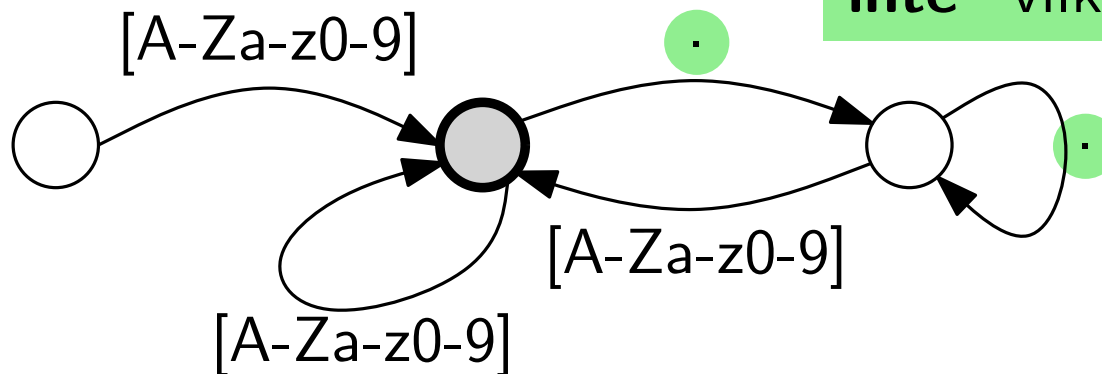
# Reguljära uttryck till automater

Reguljärt uttryck från förra föreläsningen för lokal del av e-post-adress:

$[A-Za-z0-9] ([A-Za-z0-9.]^* [A-Za-z0-9]) ?$

Kan vi konstruera en automat som känner igen samma språk?

(kom ihåg: "känna igen samma språk" = "beskriver samma mängd strängar")



OBS: punkterna här betyder "nästa tecken är en punkt", **inte** "vilket tecken som helst"

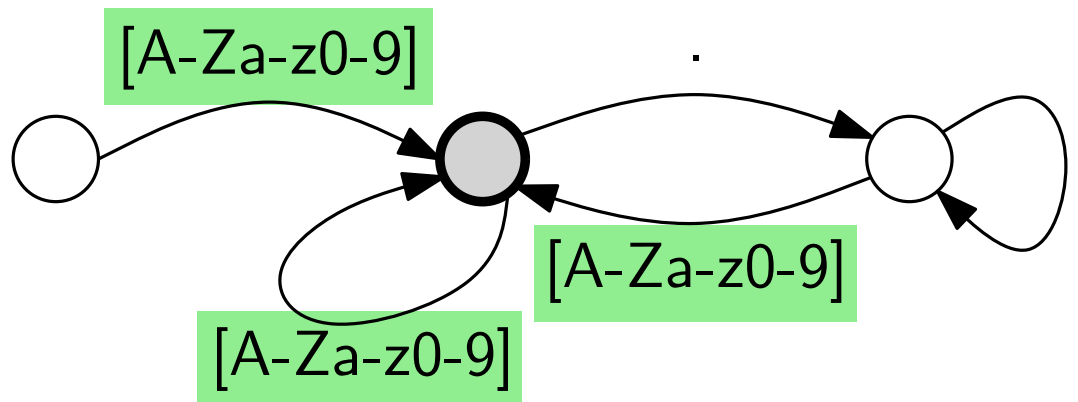
# Reguljära uttryck till automater

Reguljärt uttryck från förra föreläsningen för lokal del av e-post-adress:

$[A-Za-z0-9] ([A-Za-z0-9.]^* [A-Za-z0-9]) ?$

Kan vi konstruera en automat som känner igen samma språk?

(kom ihåg: "känna igen samma språk" = "beskriver samma mängd strängar")



Tolkas som "en pil för varje tecken i teckenintervallet"

# Reguljära uttryck till automater

Kan man alltid konvertera ett reguljärt uttryck till en automat?

# Reguljära uttryck till automater

Kan man alltid konvertera ett reguljärt uttryck till en automat?

**Ja, det kan man!**

# Reguljära uttryck till automater

Kan man alltid konvertera ett reguljärt uttryck till en automat?

**Ja, det kan man!**

(...men automaten kan bli väldigt stor)



# Reguljära uttryck till automater

Kan man alltid konvertera ett reguljärt uttryck till en automat?

**Ja, det kan man!**

(...men automaten kan bli väldigt stor)

Konverteringen går via en annan typ av automat som heter *NFA (Nondeterministic Finite Automaton)*, icke-deterministisk ändlig automat (som vi inte kommer prata om alls i den här kursen...)

# Reguljära uttryck till automater

Kan man alltid konvertera ett reguljärt uttryck till en automat?

**Ja, det kan man!**

(...men automaten kan bli väldigt stor)

Konverteringen går via en annan typ av automat som heter *NFA (Nondeterministic Finite Automaton)*, icke-deterministisk ändlig automat (som vi inte kommer prata om alls i den här kursen...)

**Praktisk betydelse:**

ger snabb algoritm (linjär tid) för att matcha reguljära uttryck!

# Reguljära uttryck till automater

Kan man alltid konvertera ett reguljärt uttryck till en automat?

**Ja, det kan man!**

(...men automaten kan bli väldigt stor)

Konverteringen går via en annan typ av automat som heter *NFA (Nondeterministic Finite Automaton)*, icke-deterministisk ändlig automat (som vi inte kommer prata om alls i den här kursen...)

**Praktisk betydelse:**

ger snabb algoritm (linjär tid) för att matcha reguljära uttryck!

**Brasklapp:**

många implementationer för reguljära uttryck använder en exponentiell backtracking-algoritm

# Reguljära uttryck till automater

Kan man alltid konvertera ett reguljärt uttryck till en automat?

**Ja, det kan man!**

(...men automaten kan bli väldigt stor)

Konverteringen går via en annan typ av automat som heter *NFA (Nondeterministic Finite Automaton)*, icke-deterministisk ändlig automat (som vi inte kommer prata om alls i den här kursen...)

**Praktisk betydelse:**

ger snabb algoritm (linjär tid) för att matcha reguljära uttryck!

**Brasklapp:**

många implementationer för reguljära uttryck använder en exponentiell backtracking-algoritm

(behövs för att hantera t.ex. bakåttreferenser som finns i regex)

Idag

Reguljära uttryck vs automater

Reguljära språk, begränsningar

Grammatiker

Stackautomater

# Reguljära språk

**Fundamentalt:** Reguljära uttryck och DFA är lika kraftfulla – kan beskriva exakt samma klass av språk

# Reguljära språk

**Fundamentalt:** Reguljära uttryck och DFA är lika kraftfulla – kan beskriva exakt samma klass av språk

De språk som man kan beskriva med reguljära uttryck eller DFA kallas för *reguljära språk*

# Reguljära språk

**Fundamentalt:** Reguljära uttryck och DFA är lika kraftfulla – kan beskriva exakt samma klass av språk

De språk som man kan beskriva med reguljära uttryck eller DFA kallas för *reguljära språk*

Exempel:

- språket “giltiga e-post-adresser” är reguljärt
- språket “binära strängar med udda antal ettor” är reguljärt
- alla ändliga språk är reguljära, t.ex. “namnen på de obligatoriska progplabbarna”



# Reguljära språk

**Fundamentalt:** Reguljära uttryck och DFA är lika kraftfulla – kan beskriva exakt samma klass av språk

De språk som man kan beskriva med reguljära uttryck eller DFA kallas för *reguljära språk*

Exempel:

- språket “giltiga e-post-adresser” är reguljärt
- språket “binära strängar med udda antal ettor” är reguljärt
- alla ändliga språk är reguljära, t.ex. “namnen på de obligatoriska progp-labbarna”

Men vad är exempel på språk som inte är reguljära?

# Begränsningar hos reguljära uttryck

Finns det språk man inte kan beskriva med reguljära uttryck (eller automater)?

Och hur kan man isåfall bevisa en sådan sak, att det inte existerar något reguljärt uttryck, hur listig man än är??

# Begränsningar hos reguljära uttryck

Finns det språk man inte kan beskriva med reguljära uttryck (eller automater)?

Och hur kan man isåfall bevisa en sådan sak, att det inte existerar något reguljärt uttryck, hur listig man än är??

**Typexempel:**

$$L = \{0^n 1^n \mid n \geq 0\}$$

# Begränsningar hos reguljära uttryck

Finns det språk man inte kan beskriva med reguljära uttryck (eller automater)?

Och hur kan man isåfall bevisa en sådan sak, att det inte existerar något reguljärt uttryck, hur listig man än är??

**Typexempel:**

$$L = \{0^n 1^n \mid n \geq 0\}$$

$0^n$  = strängen som består av  $n$  stycken nollor

# Begränsningar hos reguljära uttryck

Finns det språk man inte kan beskriva med reguljära uttryck (eller automater)?

Och hur kan man isåfall bevisa en sådan sak, att det inte existerar något reguljärt uttryck, hur listig man än är??

## **Typexempel:**

$$L = \{0^n 1^n \mid n \geq 0\}$$

## **Intuition:**

Efter att en automat läst alla nollor måste den komma ihåg exakt hur många det var. Men automaten har inget minne så den kan inte komma ihåg det.

# Begränsningar hos reguljära uttryck

Finns det språk man inte kan beskriva med reguljära uttryck (eller automater)?

Och hur kan man isåfall bevisa en sådan sak, att det inte existerar något reguljärt uttryck, hur listig man än är??

**Typexempel:**

$$L = \{0^n 1^n \mid n \geq 0\}$$

**Intuition: (ej bevis!)**

Efter att en automat läst alla nollor måste den komma ihåg exakt hur många det var. Men automaten har inget minne så den kan inte komma ihåg det.

**Sanning med modifikation:** automaten kan använda de olika tillstånden för att minnas litegrann av vad den sett

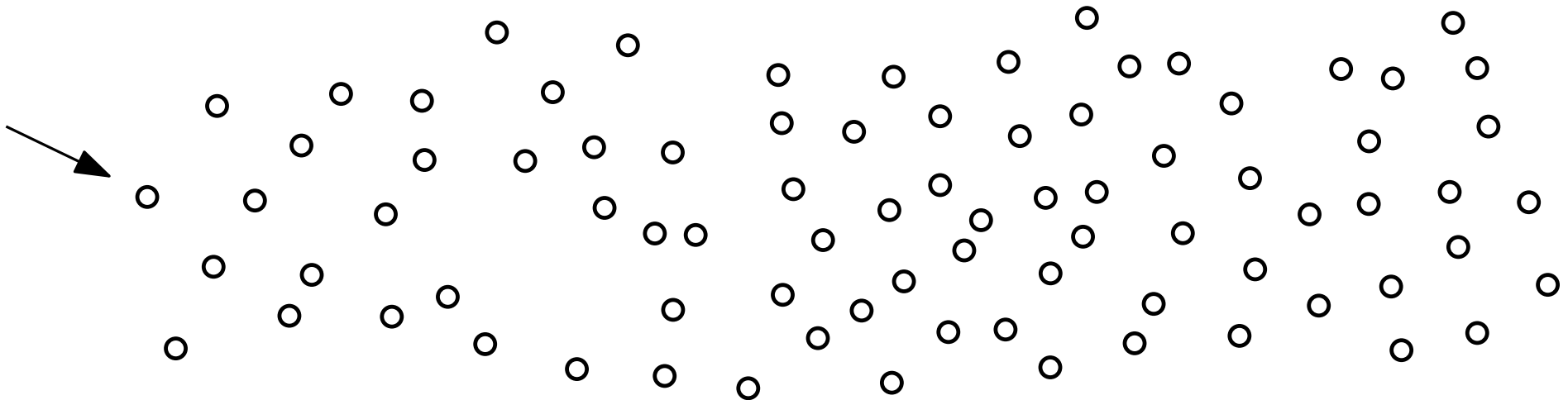
Bevis att  $\{0^n 1^n \mid n \geq 0\}$  inte är reguljärt

Antag att det finns en automat för  $\{0^n 1^n \mid n \geq 0\}$

Bevis att  $\{0^n 1^n \mid n \geq 0\}$  inte är reguljärt

Antag att det finns en automat för  $\{0^n 1^n \mid n \geq 0\}$

Den har ett ändligt antal tillstånd, säg  $t$  stycken



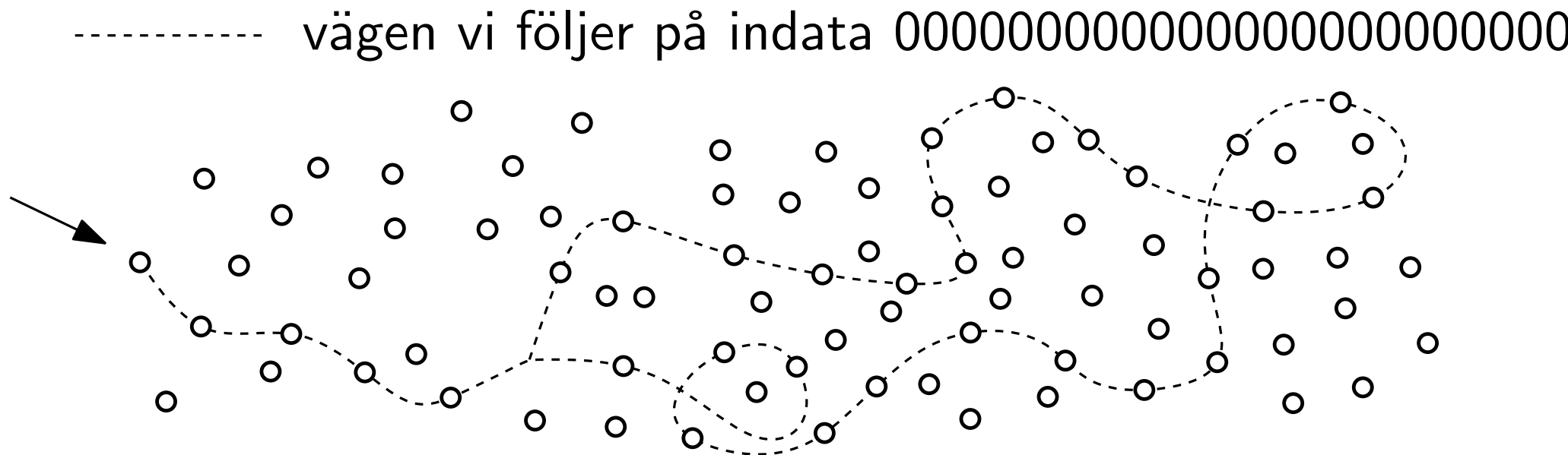


# Bevis att $\{0^n 1^n \mid n \geq 0\}$ inte är reguljärt

Antag att det finns en automat för  $\{0^n 1^n \mid n \geq 0\}$

Den har ett ändligt antal tillstånd, säg  $t$  stycken

Betrakta vad som händer när vi kör automaten på en sträng med en massa nollor



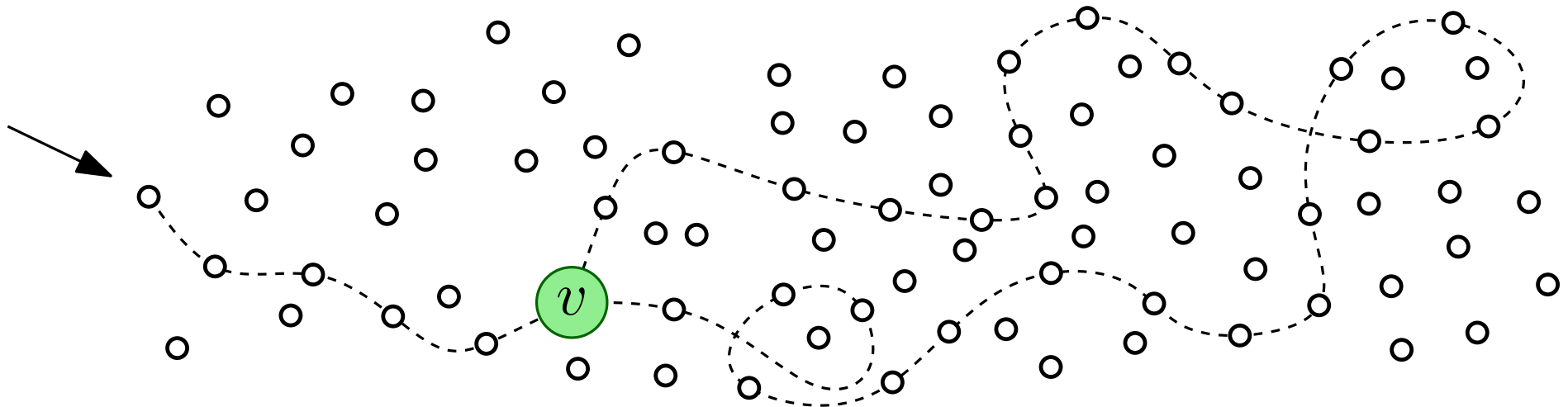
# Bevis att $\{0^n 1^n \mid n \geq 0\}$ inte är reguljärt

Antag att det finns en automat för  $\{0^n 1^n \mid n \geq 0\}$

Den har ett ändligt antal tillstånd, säg  $t$  stycken

Betrakta vad som händer när vi kör automaten på en sträng med en massa nollor

Efter högst  $t$  steg måste vi komma tillbaka till något tillstånd  $v$  som vi redan besökt (varför)?

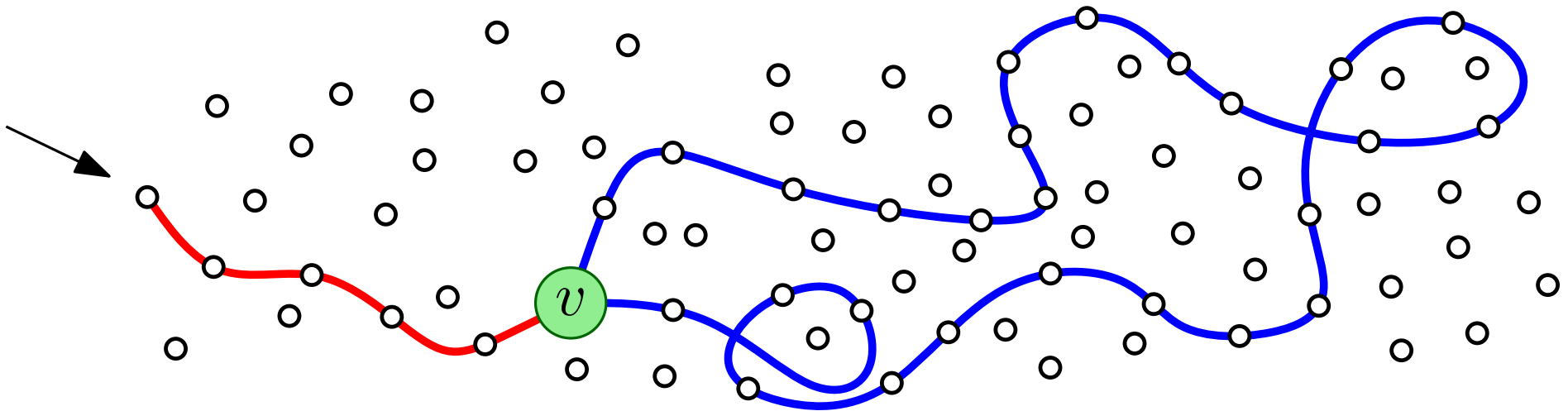


Bevis att  $\{0^n 1^n \mid n \geq 0\}$  inte är reguljärt

Säg att vi besöker  $v$  efter  $s$  steg och  $s + r$  steg på indata 0000000

—  $s$  steg längs "0"-övergångar tills vi kommer till  $v$

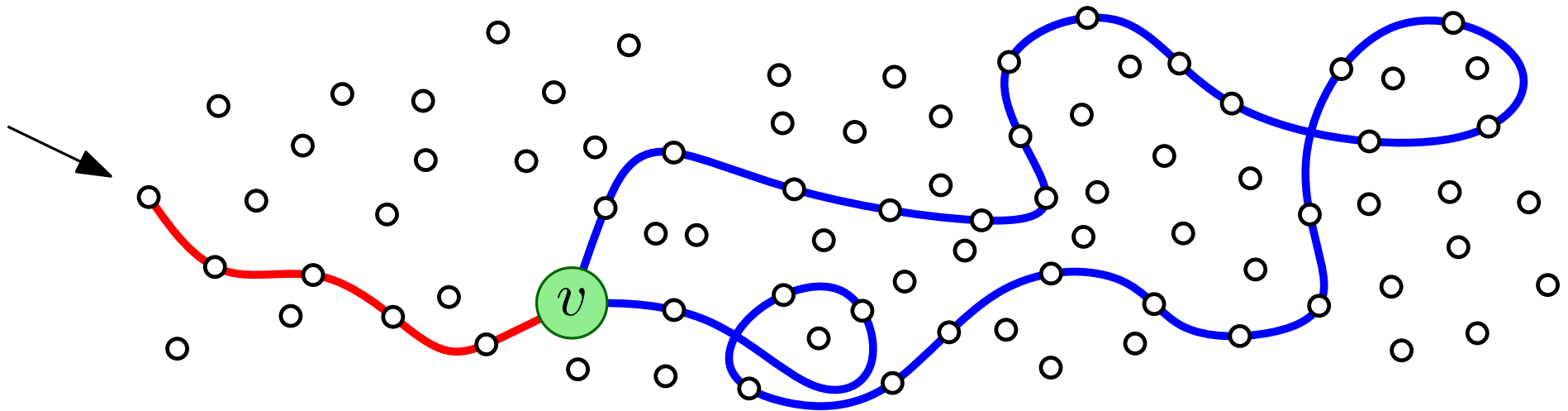
— en loop på  $r$  steg som tar oss tillbaka till  $v$



Bevis att  $\{0^n 1^n \mid n \geq 0\}$  inte är reguljärt

Säg att vi besöker  $v$  efter  $s$  steg och  $s + r$  steg på indata 0000000

Hur kommer automaten bete sig på indata  $0^s 1^s$ ?



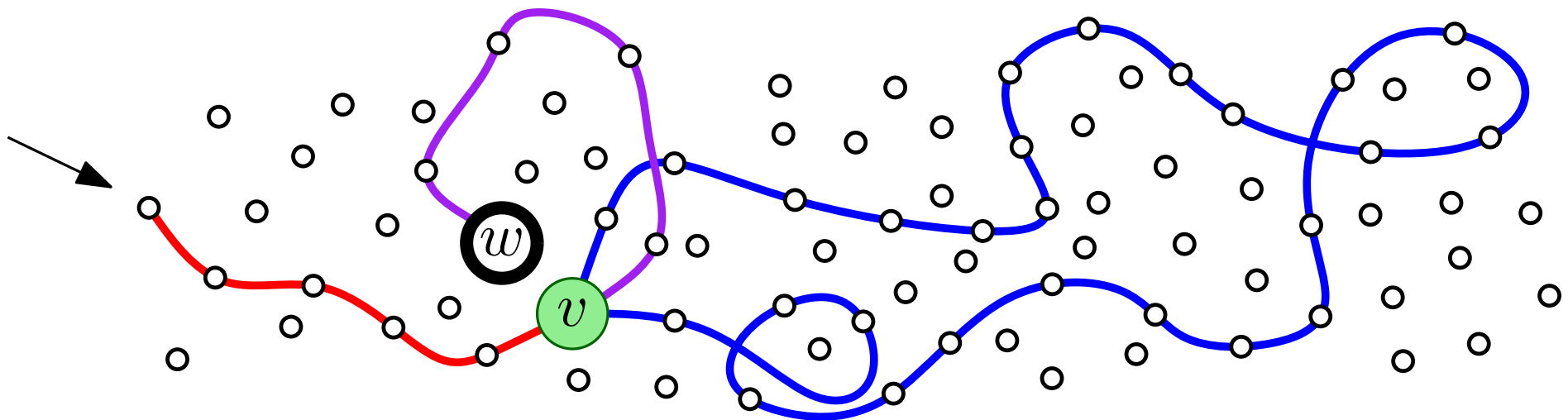
# Bevis att $\{0^n 1^n \mid n \geq 0\}$ inte är reguljärt

Säg att vi besöker  $v$  efter  $s$  steg och  $s + r$  steg på indata 0000000

Hur kommer automaten bete sig på indata  $0^s 1^s$ ?

Den kommer ta  $s$  steg till  $v$ , sedan  $s$  steg till något accepterande tillstånd  $w$  (varför?)

—  $s$  steg längs "1"-övergångar



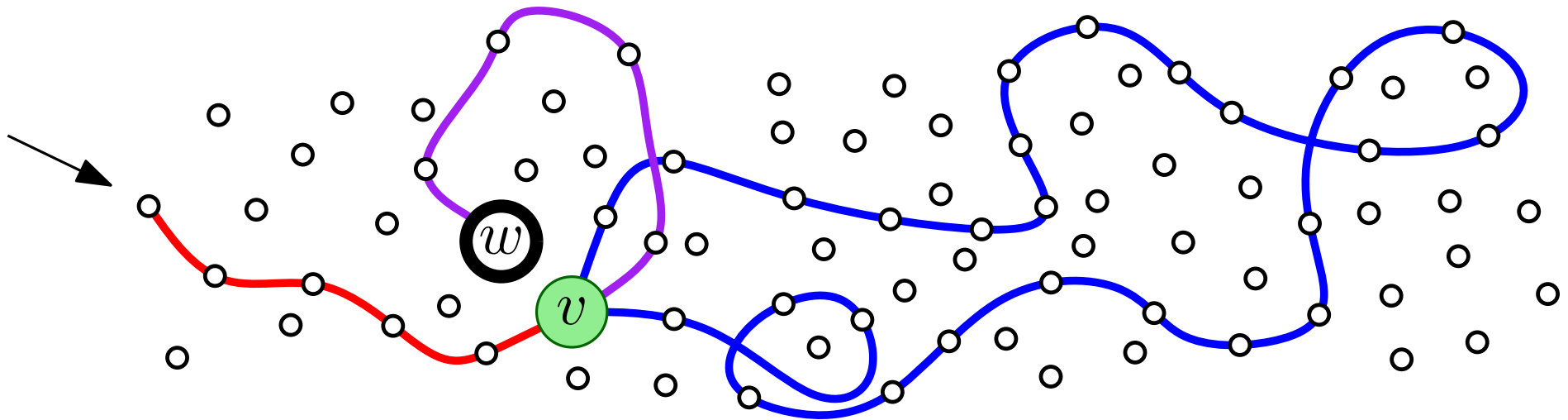
# Bevis att $\{0^n 1^n \mid n \geq 0\}$ inte är reguljärt

Säg att vi besöker  $v$  efter  $s$  steg och  $s + r$  steg på indata 0000000

Hur kommer automaten bete sig på indata  $0^s 1^s$ ?

Den kommer ta  $s$  steg till  $v$ , sedan  $s$  steg till något accepterande tillstånd  $w$  (varför?)

Hur kommer automaten bete sig på indata  $0^{s+r} 1^s$ ?



# Bevis att $\{0^n 1^n \mid n \geq 0\}$ inte är reguljärt

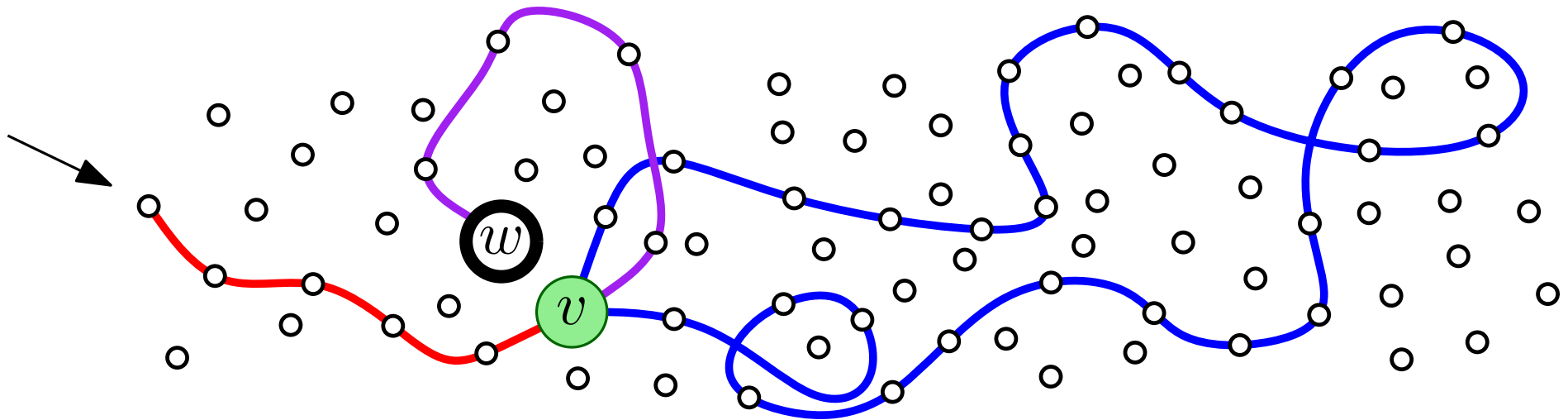
Säg att vi besöker  $v$  efter  $s$  steg och  $s + r$  steg på indata 0000000

Hur kommer automaten bete sig på indata  $0^s 1^s$ ?

Den kommer ta  $s$  steg till  $v$ , sedan  $s$  steg till något accepterande tillstånd  $w$  (varför?)

Hur kommer automaten bete sig på indata  $0^{s+r} 1^s$ ?

Den kommer ta  $s$  steg till  $v$ , sedan  $r$  steg för ett varv i loopen, sedan  $s$  steg till  $w$  (varför?)



# Bevis att $\{0^n 1^n \mid n \geq 0\}$ inte är reguljärt

Säg att vi besöker  $v$  efter  $s$  steg och  $s + r$  steg på indata 0000000

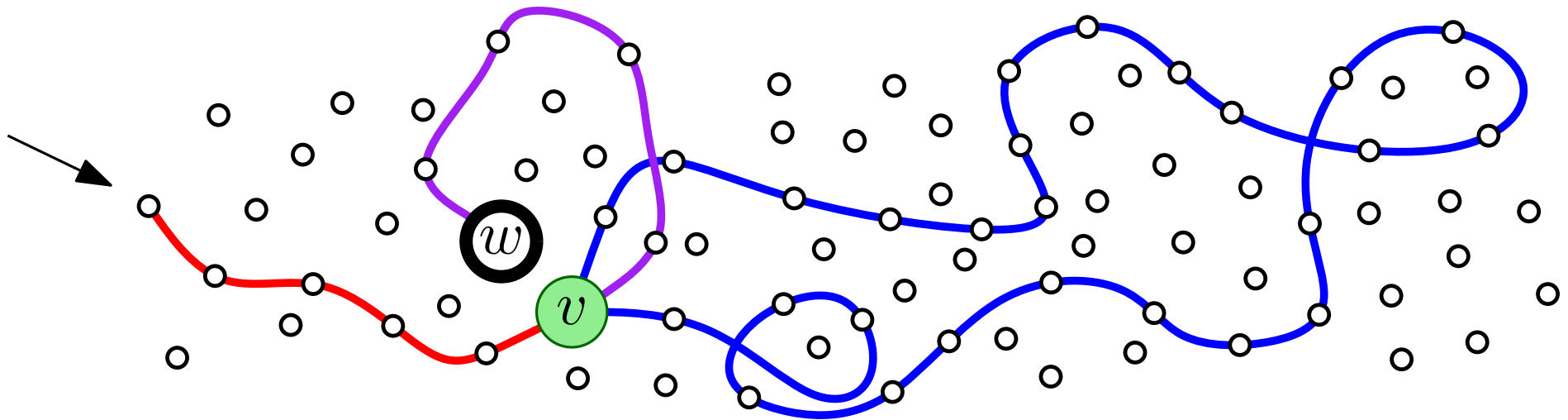
Hur kommer automaten bete sig på indata  $0^s 1^s$ ?

Den kommer ta  $s$  steg till  $v$ , sedan  $s$  steg till något accepterande tillstånd  $w$  (varför?)

Hur kommer automaten bete sig på indata  $0^{s+r} 1^s$ ?

Den kommer ta  $s$  steg till  $v$ , sedan  $r$  steg för ett varv i loopen, sedan  $s$  steg till  $w$  (varför?)

$\implies$  automaten accepterar strängen  $0^{s+r} 1^s$ , **motsägelse!**





# Icke-reguljära språk

Beviset vi just såg kan generaliseras för att visa att många andra språk inte heller är reguljära.

# Icke-reguljära språk

Beviset vi just såg kan generaliseras för att visa att många andra språk inte heller är reguljära.

## Exempel:

- Balanserade parentesuttryck
- Palindrom (t.ex. “abba”, “nitalarbralatin”)
- Strängar på formen  $xx$  för någon sträng  $x \in \{0, 1\}^n$  (t.ex. “aa”, “hejhej”, “01110010111001”)

# Icke-reguljära språk

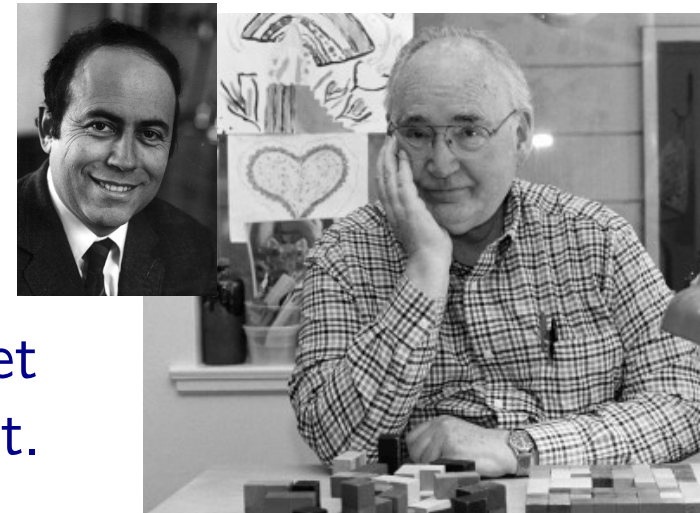
Beviset vi just såg kan generaliseras för att visa att många andra språk inte heller är reguljära.

## Exempel:

- Balanserade parentesuttryck
- Palindrom (t.ex. “abba”, “nitalarbralatin”)
- Strängar på formen  $xx$  för någon sträng  $x \in \{0, 1\}^n$  (t.ex. “aa”, “hejhej”, “01110010111001”)

Detta kallas för *pumping-lemmat* (för reguljära språk), och upptäcktes av Michael Rabin och Dana Scott 1959

Samma artikeln gav dem Turing-priset (“datavetenskapens Nobelpris”) 1976, för konceptet “icke-determinism” som de introducerade samtidigt.



# Icke-reguljära språk

Beviset vi just såg kan generaliseras för att visa att många andra språk inte heller är reguljära.

## Exempel:

- Balanserade parentesuttryck
- Palindrom (t.ex. “abba”, “nitalarbralatin”)
- Strängar på formen  $xx$  för någon sträng  $x \in \{0, 1\}^n$  (t.ex. “aa”, “hejhej”, “01110010111001”)

Men... kan vi inte känna igen detta med regexet “([01]\*)\1”?

# Icke-reguljära språk

Beviset vi just såg kan generaliseras för att visa att många andra språk inte heller är reguljära.

## Exempel:

- Balanserade parentesuttryck
- Palindrom (t.ex. “abba”, “nitalarbralatin”)
- Strängar på formen  $xx$  för någon sträng  $x \in \{0, 1\}^n$  (t.ex. “aa”, “hejhej”, “01110010111001”)

Men... kan vi inte känna igen detta med regexet “([01]\*)\1”?

Jo. Detta är ett exempel på hur regexar är kraftfullare (ger möjlighet att uttrycka fler saker) än reguljära uttryck.

## Egenskaper hos reguljära språk

Antag att  $A$  och  $B$  är reguljära språk över  $\Sigma$ . Då gäller följande:

# Egenskaper hos reguljära språk

Antag att  $A$  och  $B$  är reguljära språk över  $\Sigma$ . Då gäller följande:

- $\overline{A} = \Sigma^* - A$ , dvs,  $A$ -komplement, är reguljärt.

**Bevisidé:** byt mellan accepterande och icke-accepterande tillstånd i DFA för  $A$  (obs: fail-tillståndet viktigt!)

# Egenskaper hos reguljära språk

Antag att  $A$  och  $B$  är reguljära språk över  $\Sigma$ . Då gäller följande:

- $\overline{A} = \Sigma^* - A$ , dvs,  $A$ -komplement, är reguljärt.  
**Bevisidé:** byt mellan accepterande och icke-accepterande tillstånd i DFA för  $A$  (obs: fail-tillståndet viktigt!)
- $A \cup B$  är reguljärt.  
**Bevisidé:** beskrivs av  $R_A | R_B$ .



# Egenskaper hos reguljära språk

Antag att  $A$  och  $B$  är reguljära språk över  $\Sigma$ . Då gäller följande:

- $\overline{A} = \Sigma^* - A$ , dvs,  $A$ -komplement, är reguljärt.

**Bevisidé:** byt mellan accepterande och icke-accepterande tillstånd i DFA för  $A$  (obs: fail-tillståndet viktigt!)

- $A \cup B$  är reguljärt.

**Bevisidé:** beskrivs av  $R_A | R_B$ .

- $A \cap B$  är reguljärt.

**Bevisidé:** använd  $A \cap B = \overline{(\overline{A} \cup \overline{B})}$ .

# Egenskaper hos reguljära språk

Antag att  $A$  och  $B$  är reguljära språk över  $\Sigma$ . Då gäller följande:

- $\bar{A} = \Sigma^* - A$ , dvs,  $A$ -komplement, är reguljärt.

**Bevisidé:** byt mellan accepterande och icke-accepterande tillstånd i DFA för  $A$  (obs: fail-tillståndet viktigt!)

- $A \cup B$  är reguljärt.

**Bevisidé:** beskrivs av  $R_A | R_B$ .

- $A \cap B$  är reguljärt.

**Bevisidé:** använd  $A \cap B = \overline{(\bar{A} \cup \bar{B})}$ .

- $A - B$  är reguljärt.

**Bevisidé:** använd  $A - B = A \cap \bar{B}$ .

# Egenskaper hos reguljära språk

Antag att  $A$  och  $B$  är reguljära språk över  $\Sigma$ . Då gäller följande:

- $\bar{A} = \Sigma^* - A$ , dvs,  $A$ -komplement, är reguljärt.

**Bevisidé:** byt mellan accepterande och icke-accepterande tillstånd i DFA för  $A$  (obs: fail-tillståndet viktigt!)

- $A \cup B$  är reguljärt.

**Bevisidé:** beskrivs av  $R_A | R_B$ .

- $A \cap B$  är reguljärt.

**Bevisidé:** använd  $A \cap B = \overline{(\bar{A} \cup \bar{B})}$ .

- $A - B$  är reguljärt.

**Bevisidé:** använd  $A - B = A \cap \bar{B}$ .

Kan användas för att visa att språk är reguljära utan att konstruera automat eller reguljärt uttryck. T.ex. "alla strängar som uppfyller krav  $X$  och  $Y$  men inte  $Z$ ", där  $X$ ,  $Y$ ,  $Z$  är reguljära

Idag

Reguljära uttryck vs automater

Reguljära språk, begränsningar

Grammatiker

Stackautomater

# Grammatiker

Vi ska nu titta på ett annat sätt att beskriva språk på, som är mycket mer kraftfullt/uttrycksfullt

# Grammatiker

Vi ska nu titta på ett annat sätt att beskriva språk på, som är mycket mer kraftfullt/uttrycksfullt

Varför kan inte reguljära uttryck beskriva balanserade parentesuttryck?

# Grammatiker

Vi ska nu titta på ett annat sätt att beskriva språk på, som är mycket mer kraftfullt/uttrycksfullt

Varför kan inte reguljära uttryck beskriva balanserade parentesuttryck?

På sätt och vis kan man säga att det delvis beror på att balanserade parentesuttryck har en rekursiv struktur.

# Grammatik för parentesuttryck

Vi kan definiera balanserade parentesuttryck matematiskt enligt följande:



# Grammatik för parentesuttryck

Vi kan definiera balanserade parentesuttryck matematiskt enligt följande:

- **Basfall:**

tomma strängen är ett balanserat parentesuttryck

# Grammatik för parentesuttryck

Vi kan definiera balanserade parentesuttryck matematiskt enligt följande:

- **Basfall:**

tomma strängen är ett balanserat parentesuttryck

- **Induktiv definition:**

om " $x$ " och " $y$ " är balanserade parentesuttryck så är " $(x)y$ " ett balanserat parentesuttryck

# Grammatik för parentesuttryck

Vi kan definiera balanserade parentesuttryck matematiskt enligt följande:

- **Basfall:**

tomma strängen är ett balanserat parentesuttryck

- **Induktiv definition:**

om " $x$ " och " $y$ " är balanserade parentesuttryck så är " $(x)y$ " ett balanserat parentesuttryck

Grammatiker är ett sätt att beskriva språk med den här typen av definitioner

# Grammatik för parentesuttryck

Vi kan definiera balanserade parentesuttryck matematiskt enligt följande:

- **Basfall:**

tomma strängen är ett balanserat parentesuttryck

- **Induktiv definition:**

om “ $x$ ” och “ $y$ ” är balanserade parentesuttryck så är “ $(x)y$ ” ett balanserat parentesuttryck

Grammatiker är ett sätt att beskriva språk med den här typen av definitioner

En grammatik för balanserade parentesuttryck:

$$\text{Expr} \rightarrow \epsilon \mid (\text{Expr})\text{Expr}$$

# Grammatik för parentesuttryck

Vi kan definiera balanserade parentesuttryck matematiskt enligt följande:

- **Basfall:**

tomma strängen är ett balanserat parentesuttryck

- **Induktiv definition:**

om “ $x$ ” och “ $y$ ” är balanserade parentesuttryck så är “ $(x)y$ ” ett balanserat parentesuttryck

Grammatiker är ett sätt att beskriva språk med den här typen av definitioner

En grammatik för balanserade parentesuttryck:

$$\text{Expr} \rightarrow \epsilon \mid (\text{Expr})\text{Expr}$$

Utläses “Ett Expr är antingen tomma strängen, eller så är det vänsterparentes följt av ett Expr följt av högerparentes följt av ett Expr”

# Beståndsdelar i en grammatik

**Slutsymboler** (eng. terminals): de symboler som ingår i indata (tänk på det som tecken tills vidare)

# Beståndsdelar i en grammatik

**Slutsymboler** (eng. terminals): de symboler som ingår i indata (tänk på det som tecken tills vidare)

**Icke-slutsymboler** (eng. non-terminals): olika typer av delsträngar som kan förekomma i indata, t.ex.: en for-loop.

# Beståndsdelar i en grammatik

**Slutsymboler** (eng. terminals): de symboler som ingår i indata (tänk på det som tecken tills vidare)

**Icke-slutsymboler** (eng. non-terminals): olika typer av delsträngar som kan förekomma i indata, t.ex.: en for-loop.

**Produktionsregler** (eng. production rules): regler för hur varje icke-slutsymbol kan formas, i termer av slutsymboler och andra icke-slutsymboler



# Beståndsdelar i en grammatik

**Slutsymboler** (eng. terminals): de symboler som ingår i indata (tänk på det som tecken tills vidare)

**Icke-slutsymboler** (eng. non-terminals): olika typer av delsträngar som kan förekomma i indata, t.ex.: en for-loop.

**Produktionsregler** (eng. production rules): regler för hur varje icke-slutsymbol kan formas, i termer av slutsymboler och andra icke-slutsymboler

## Exempel

$$S \rightarrow B \mid AA$$

$$A \rightarrow cA \mid dB$$

$$B \rightarrow aSa \mid \epsilon$$

Icke-slutsymboler S, A, B. Slutsymboler a, c, d.

# Beståndsdelar i en grammatik

**Slutsymboler** (eng. terminals): de symboler som ingår i indata (tänk på det som tecken tills vidare)

**Icke-slutsymboler** (eng. non-terminals): olika typer av delsträngar som kan förekomma i indata, t.ex.: en for-loop.

**Produktionsregler** (eng. production rules): regler för hur varje icke-slutsymbol kan formas, i termer av slutsymboler och andra icke-slutsymboler

## Exempel

$$S \rightarrow B \mid AA$$

$$A \rightarrow cA \mid dB$$

$$B \rightarrow aSa \mid \epsilon$$

Ett  $S$  är antingen ett  $B$   
eller ett  $A$  följt av ett  $A$

Icke-slutsymboler  $S$ ,  $A$ ,  $B$ . Slutsymboler  $a$ ,  $c$ ,  $d$ .

# Beståndsdelar i en grammatik

**Slutsymboler** (eng. terminals): de symboler som ingår i indata (tänk på det som tecken tills vidare)

**Icke-slutsymboler** (eng. non-terminals): olika typer av delsträngar som kan förekomma i indata, t.ex.: en for-loop.

**Produktionsregler** (eng. production rules): regler för hur varje icke-slutsymbol kan formas, i termer av slutsymboler och andra icke-slutsymboler

## Exempel

$$S \rightarrow B \mid AA$$

$$A \rightarrow cA \mid dB$$

$$B \rightarrow aSa \mid \epsilon$$

Ett  $A$  är antingen ett  $c$  följt av ett  $A$  eller ett  $d$  följt av ett  $B$

Icke-slutsymboler  $S, A, B$ . Slutsymboler  $a, c, d$ .

# Beståndsdelar i en grammatik

**Slutsymboler** (eng. terminals): de symboler som ingår i indata (tänk på det som tecken tills vidare)

**Icke-slutsymboler** (eng. non-terminals): olika typer av delsträngar som kan förekomma i indata, t.ex.: en for-loop.

**Produktionsregler** (eng. production rules): regler för hur varje icke-slutsymbol kan formas, i termer av slutsymboler och andra icke-slutsymboler

## Exempel

$$S \rightarrow B \mid AA$$

$$A \rightarrow cA \mid dB$$

$$B \rightarrow aSa \mid \epsilon$$

Ett  $B$  är antingen ett  $a$  följt av ett  $S$  följt av ett  $a$ , eller tomma strängen,  $\epsilon$

Icke-slutsymboler  $S$ ,  $A$ ,  $B$ . Slutsymboler  $a$ ,  $c$ ,  $d$ .

# Grammatik för programmeringsspråk

Grammatiker är tillräckligt kraftfulla för att beskriva syntaxen för ens favoritprogramspråk.

# Grammatik för programmeringsspråk

Grammatiker är tillräckligt kraftfulla för att beskriva syntaxen för ens favoritprogramspråk.

```
Statement → begin StatementList end |  
           if Conditional then Statement else Statement |  
           Assignment |  
           FunctionCall |  
           ...
```

```
StatementList → StatementList ; Statement |  
              Statement
```

```
Assignment → Variable = Expression
```

```
FunctionCall → FunctionName (ParameterList)
```

# Grammatik för programmeringsspråk

Grammatiker är tillräckligt kraftfulla för att beskriva syntaxen för ens favoritprogramspråk.

```
Statement → begin StatementList end |  
           if Conditional then Statement else Statement |  
           Assignment |  
           FunctionCall |  
           ...
```

```
StatementList → StatementList ; Statement |  
              Statement
```

```
Assignment → Variable = Expression
```

```
FunctionCall → FunctionName (ParameterList)
```

...plus många många fler produktionsregler och symboler

Googlar man på “grammar for [språk] syntax” brukar man hitta en formell grammatik för språket

# Exempel

$S \rightarrow B \mid AA$

$A \rightarrow cA \mid dB$

$B \rightarrow aSa \mid \epsilon$

Startsymbol S



# Exempel

$S \rightarrow B \mid AA$

Startsymbol S

$A \rightarrow cA \mid dB$

$B \rightarrow aSa \mid \epsilon$

Hur hitta strängar som ligger i språket?

# Exempel

$S \rightarrow B \mid AA$

Startsymbol S

$A \rightarrow cA \mid dB$

$B \rightarrow aSa \mid \epsilon$

Hur hitta strängar som ligger i språket?

Använd produktionsreglerna!

# Exempel

$S \rightarrow B \mid AA$

Startsymbol S

$A \rightarrow cA \mid dB$

$B \rightarrow aSa \mid \epsilon$

Hur hitta strängar som ligger i språket?

Använd produktionsreglerna!

$S \rightarrow B \rightarrow aSa \rightarrow aBa \rightarrow aa$

# Exempel

$S \rightarrow B \mid AA$

Startsymbol S

$A \rightarrow cA \mid dB$

$B \rightarrow aSa \mid \epsilon$

Hur hitta strängar som ligger i språket?

Använd produktionsreglerna!

$S \rightarrow B \rightarrow aSa \rightarrow aBa \rightarrow aa$

$S \rightarrow B \rightarrow aSa \rightarrow aAAa \rightarrow acAAa \rightarrow acdBAa \rightarrow acdAa \rightarrow acddBa \rightarrow acdda$

# Exempel

$S \rightarrow B \mid AA$       Startsymbol S

$A \rightarrow cA \mid dB$

$B \rightarrow aSa \mid \epsilon$

Hur hitta strängar som ligger i språket?

Använd produktionsreglerna!

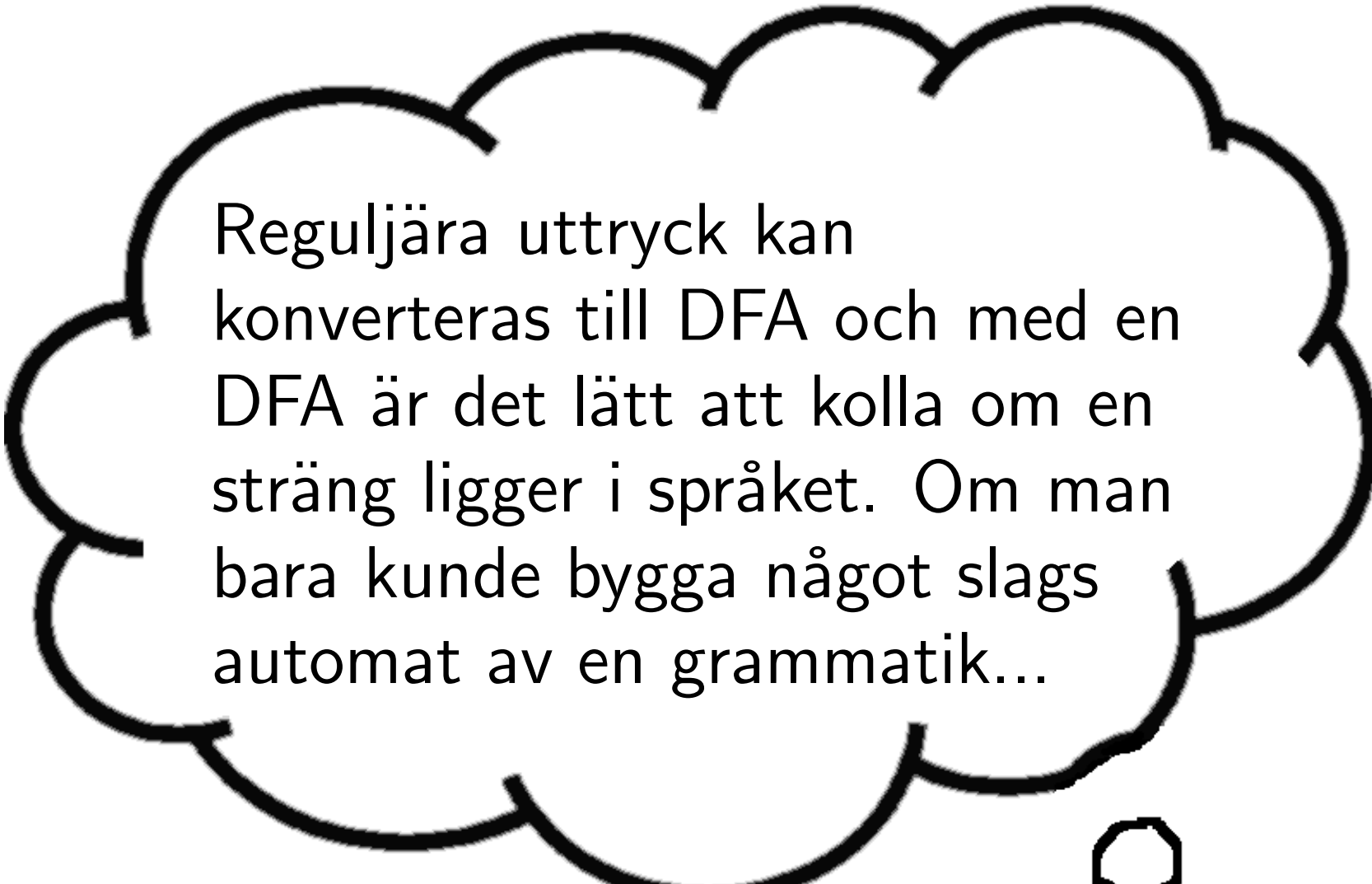
$S \rightarrow B \rightarrow aSa \rightarrow aBa \rightarrow aa$

$S \rightarrow B \rightarrow aSa \rightarrow aAAa \rightarrow acAAa \rightarrow acdBAa \rightarrow acdAa \rightarrow acddBa \rightarrow acdda$

Ofta vill vi dock gå åt andra hållet:

Givet en sträng, ligger den i språket?

## Exempel



Reguljära uttryck kan konverteras till DFA och med en DFA är det lätt att kolla om en sträng ligger i språket. Om man bara kunde bygga något slags automat av en grammatik...

Ofta vill vi dock gå åt andra hållet:  
Givet en sträng, ligger den i språket?



Idag

Reguljära uttryck vs automater

Reguljära språk, begränsningar

Grammatiker

Stackautomater

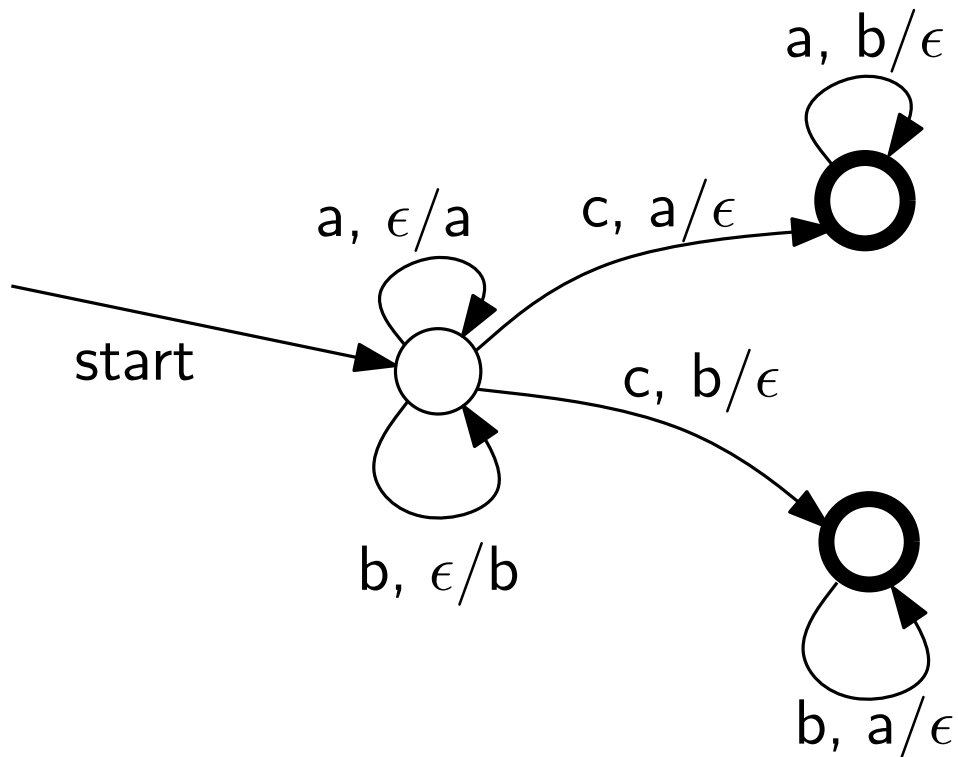
# Stackautomater

Stackautomat (PDA, Push-Down Automaton):  
som DFA, men har ett **obegränsat** minne i form av en stack



# Stackautomater

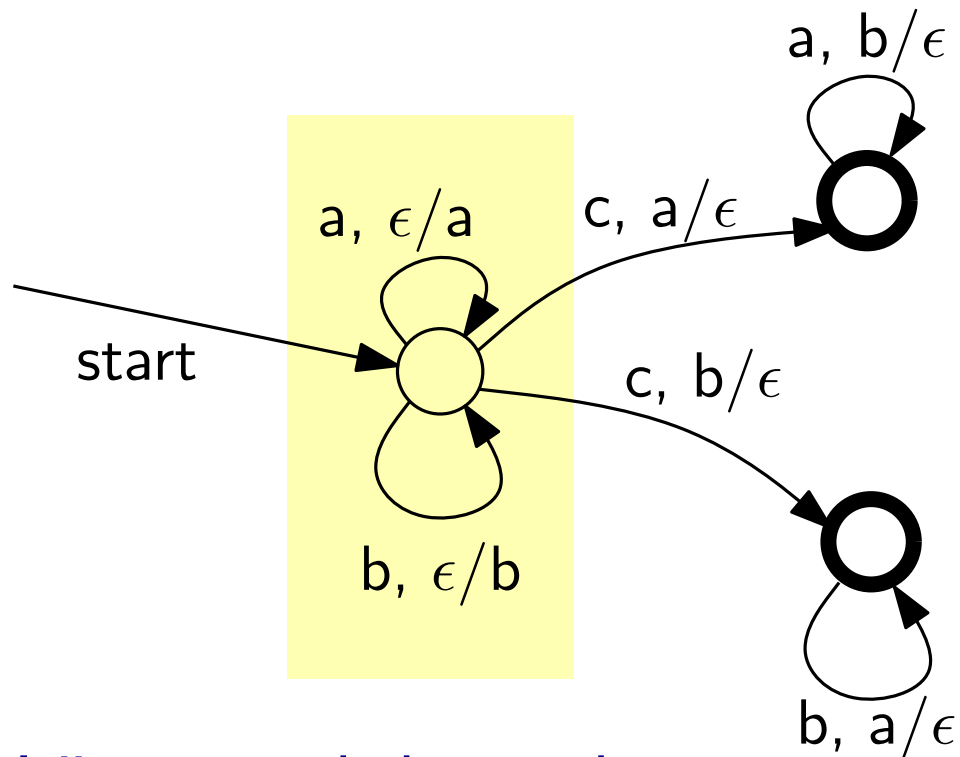
Stackautomat (PDA, Push-Down Automaton):  
som DFA, men har ett **obegränsat** minne i form av en stack



$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$

# Stackautomater

Stackautomat (PDA, Push-Down Automaton):  
som DFA, men har ett **obegränsat** minne i form av en stack

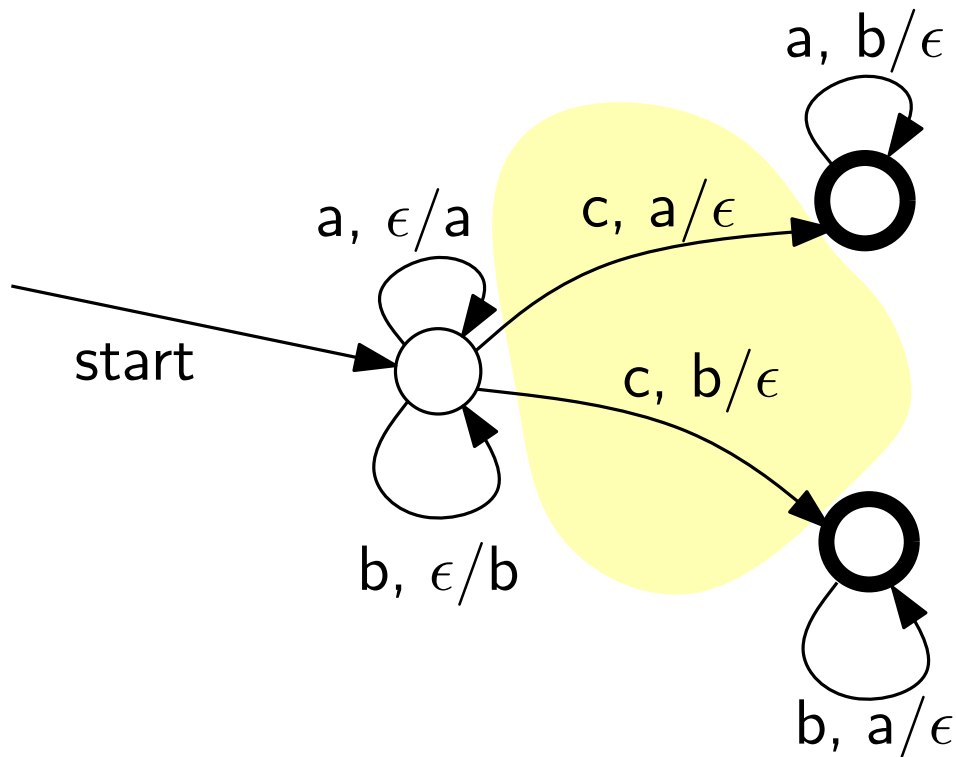


$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$

Läs a:n och b:n och  
lägg på stacken

# Stackautomater

Stackautomat (PDA, Push-Down Automaton):  
som DFA, men har ett **obegränsat** minne i form av en stack

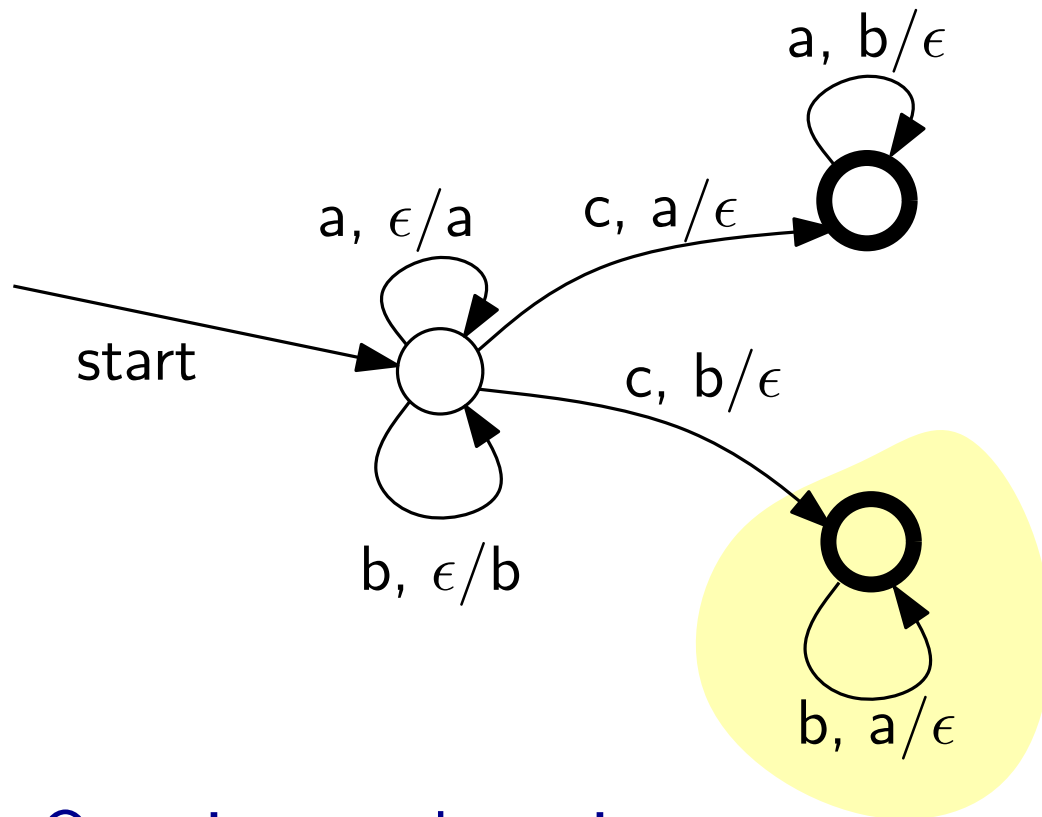


$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$

När det kommer ett  $c$ , hoppa  
till något av de accepterande  
tillstånden beroende på om det  
översta tecknet på stacken är  $a$   
eller  $b$

# Stackautomater

Stackautomat (PDA, Push-Down Automaton):  
som DFA, men har ett **obegränsat** minne i form av en stack

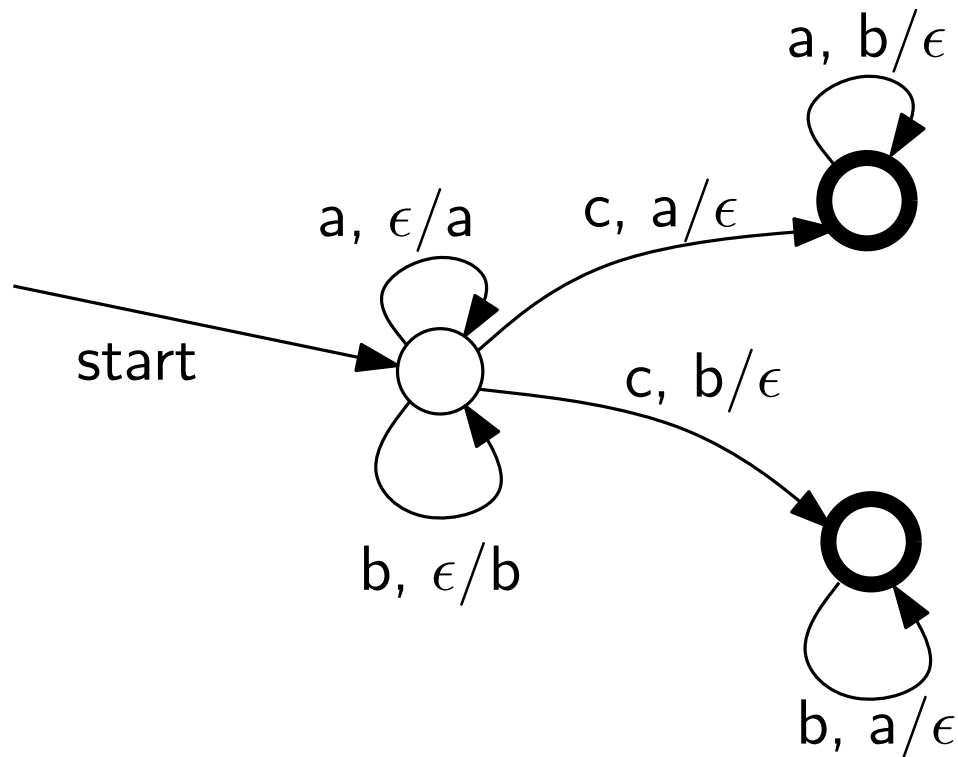


$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$

Om sista tecknet innan  $c$  var ett  $b$ : fortsätt läsa  $b$ :n från indata och kolla att det ligger minst lika många  $a$ :n på stacken

# Stackautomater

Stackautomat (PDA, Push-Down Automaton):  
som DFA, men har ett **obegränsat** minne i form av en stack

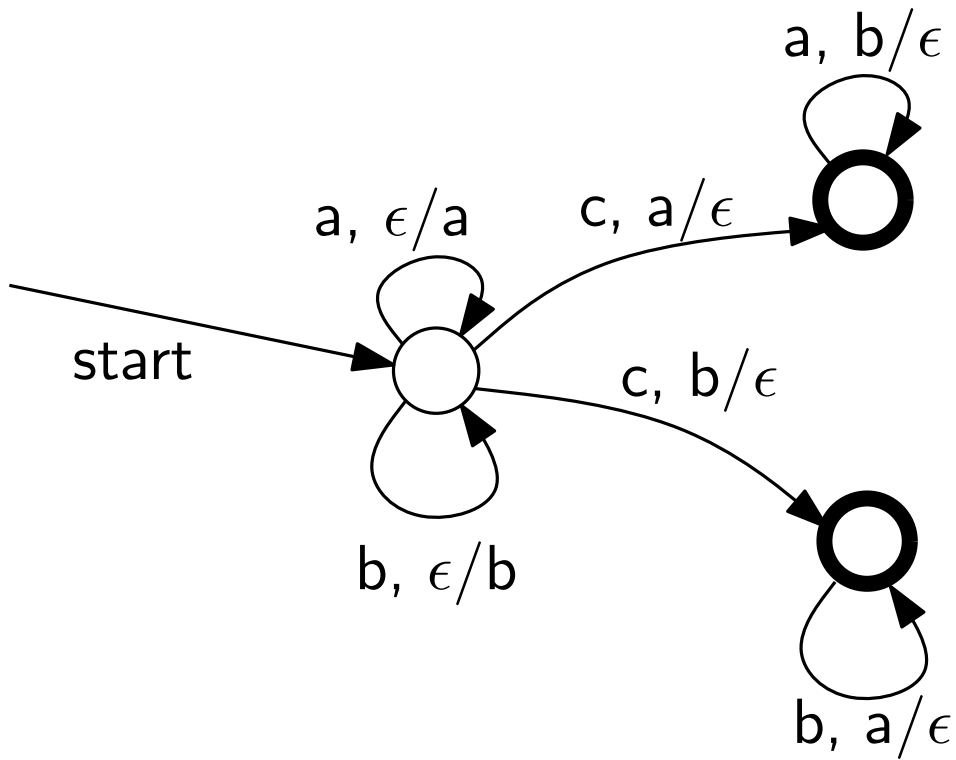


$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$

Automaten accepterar om den är i ett accepterande tillstånd  
**OCH** stacken är tom när indata är slut

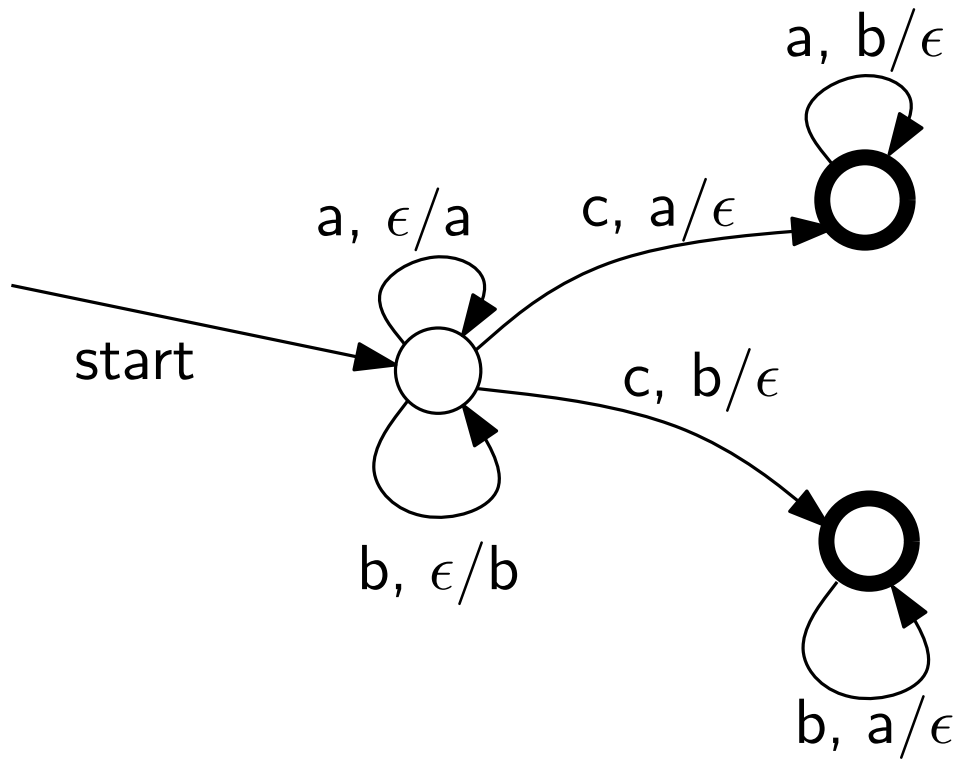
# Exempelkörning, stackautomat

$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$



# Exempelkörning, stackautomat

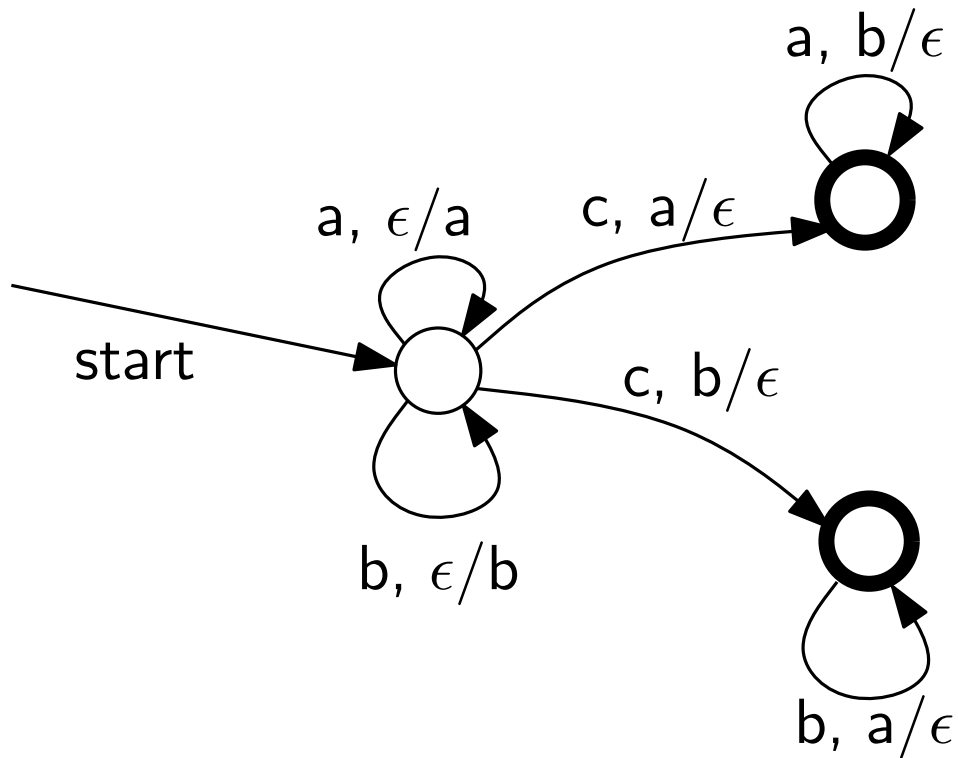
$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$



Indata:  
baabcbbb

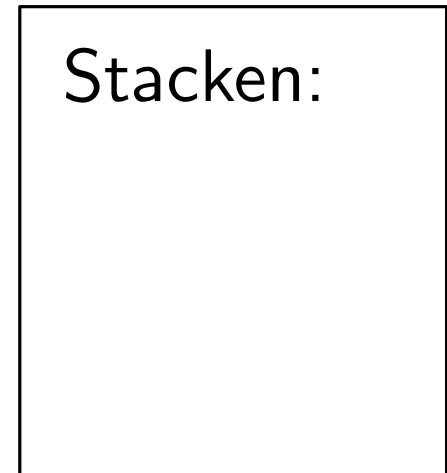
# Exempelkörning, stackautomat

$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$



Indata:  
baabcbbb

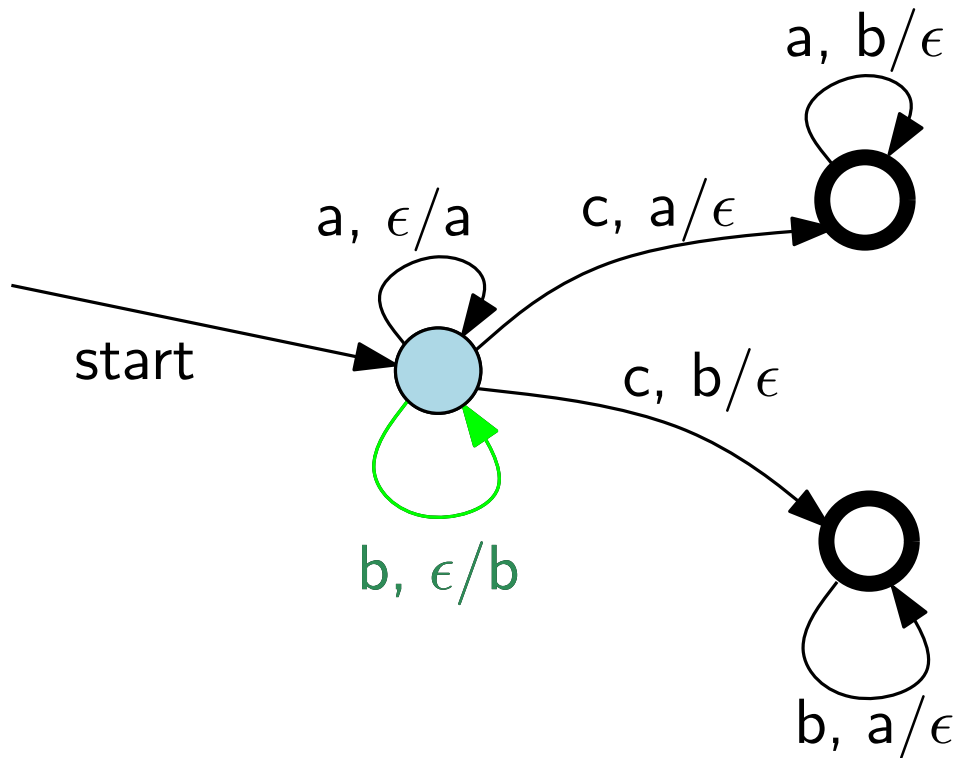
Stacken:





# Exempelkörning, stackautomat

$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$



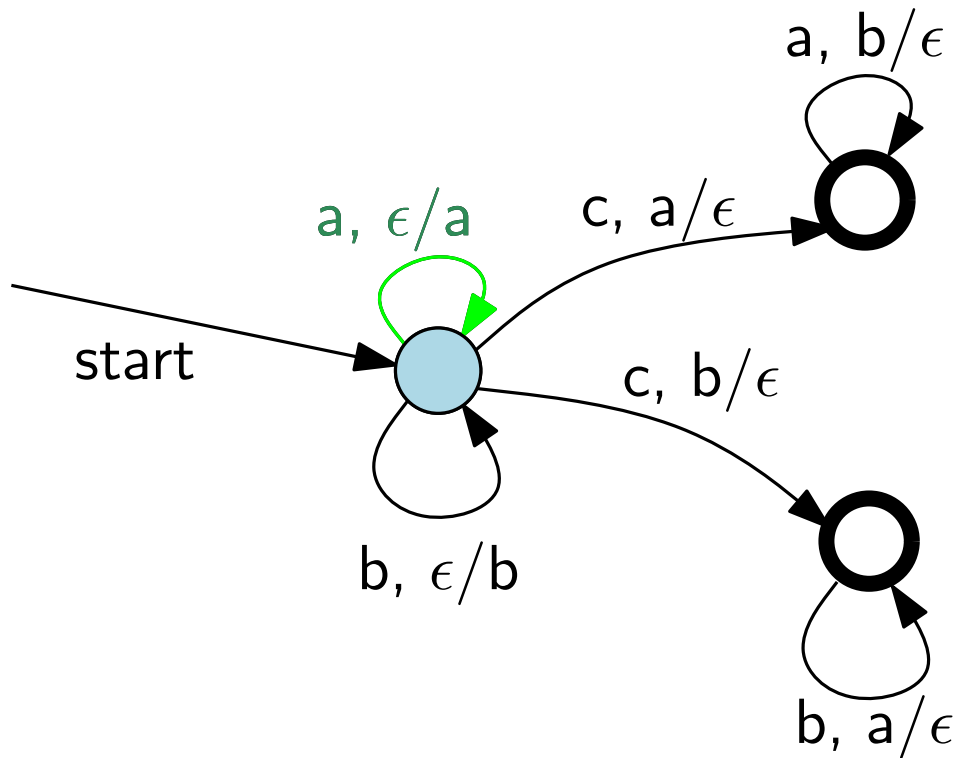
Indata:  
baabcbbb

Stacken:

b pushas

# Exempelkörning, stackautomat

$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$



Indata:  
b**a**abcbbb

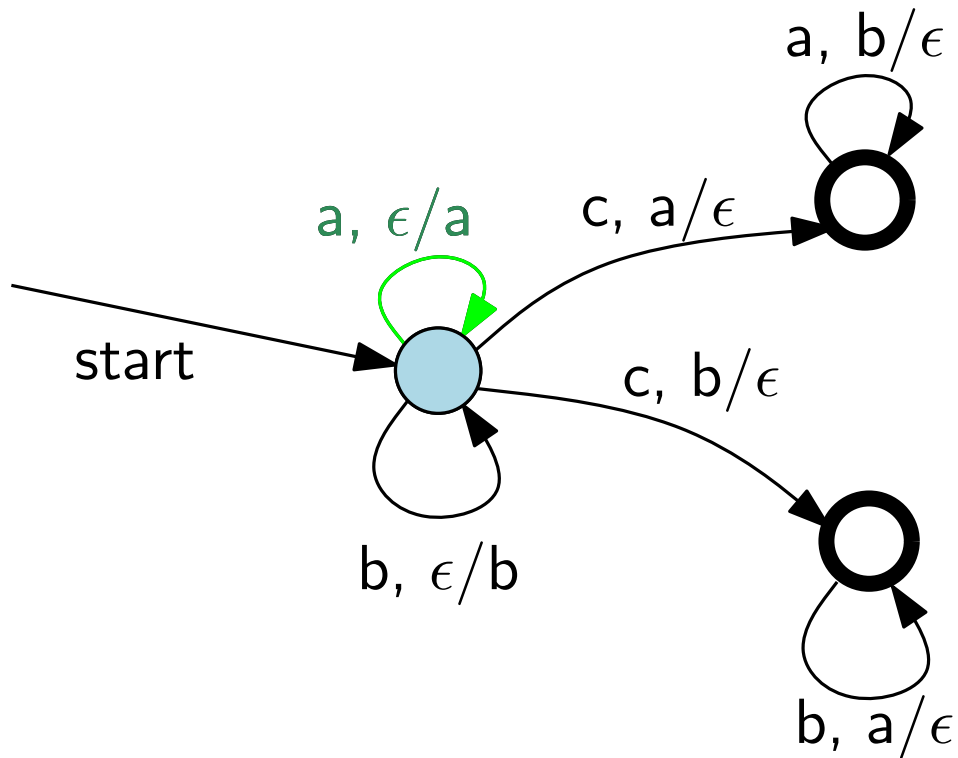
Stacken:

a pushas

b

# Exempelkörning, stackautomat

$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$



Indata:  
ba**a**bcbbb

Stacken:

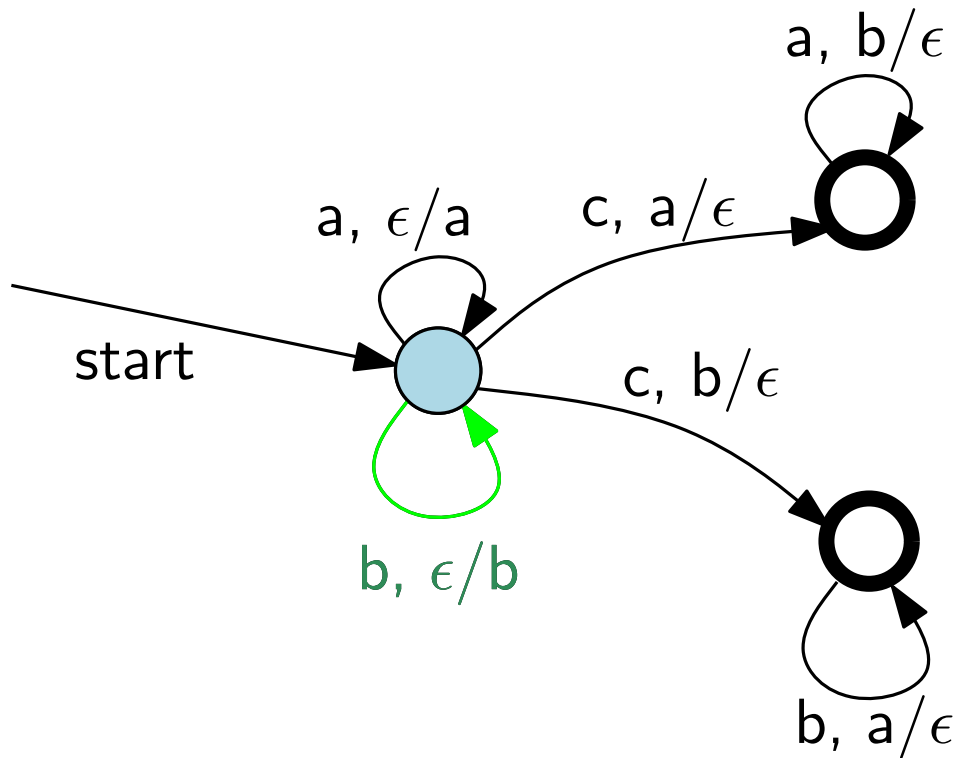
a pushas

a

b

# Exempelkörning, stackautomat

$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$

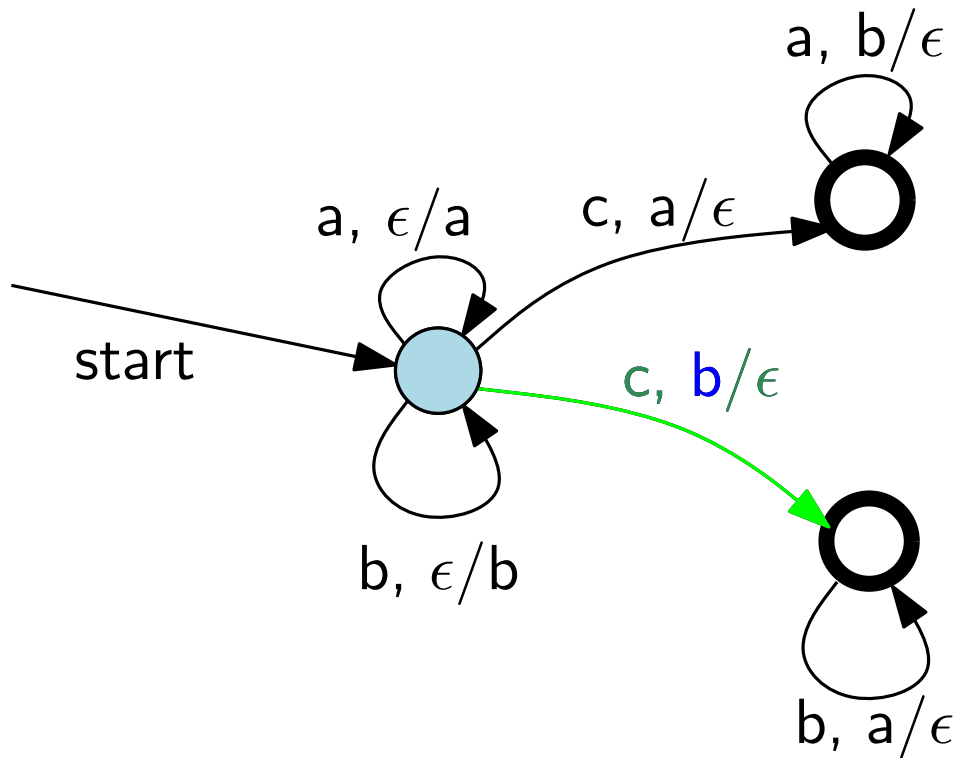


Indata:  
baab**b**cbbb

Stacken:  
b pushas  
a  
a  
b

# Exempelkörning, stackautomat

$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$

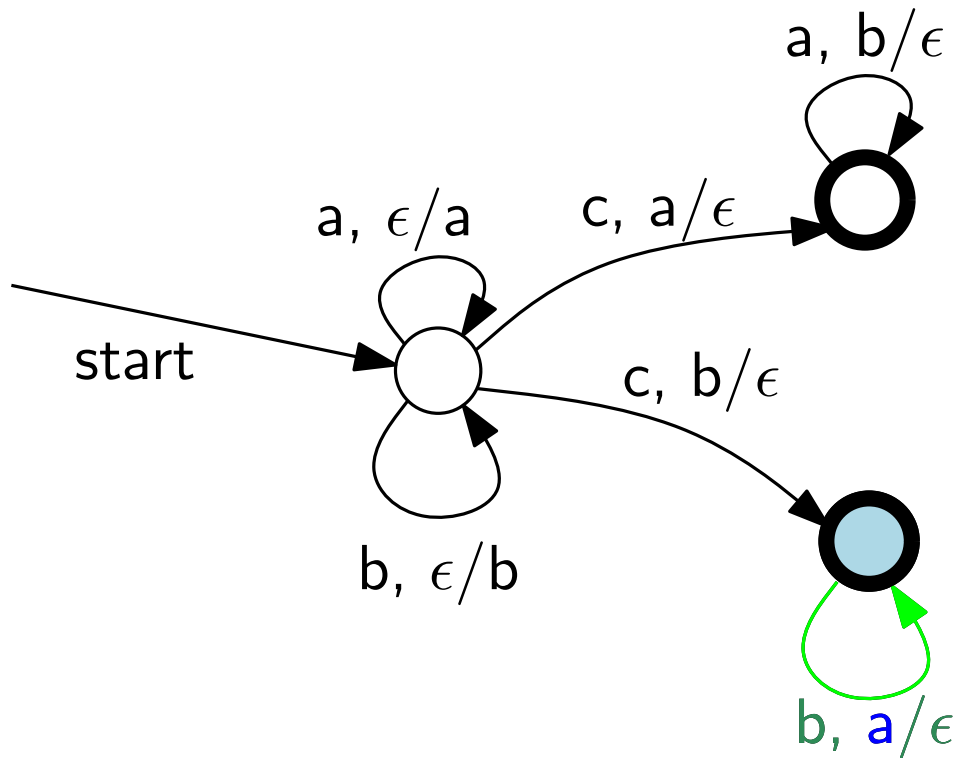


Indata:  
baab**c**bbb

Stacken:  
**b** poppas  
a  
a  
b

# Exempelkörning, stackautomat

$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$



Indata:  
baabc**b**bb

Stacken:

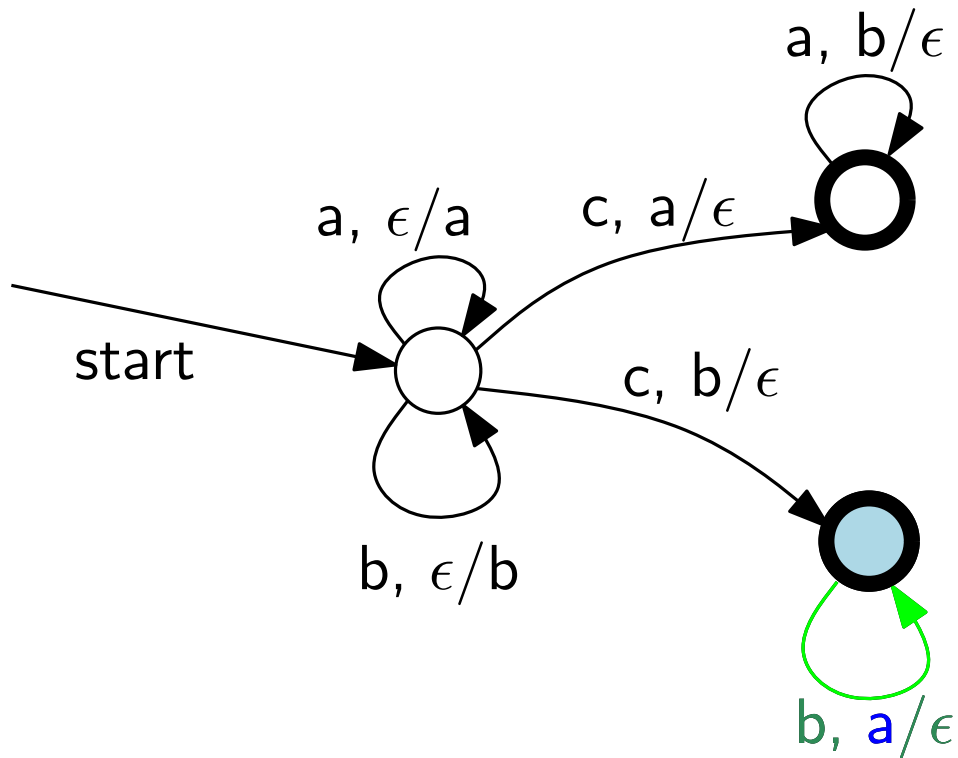
a poppas

a

b

# Exempelkörning, stackautomat

$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$



Indata:  
baabcb**b**

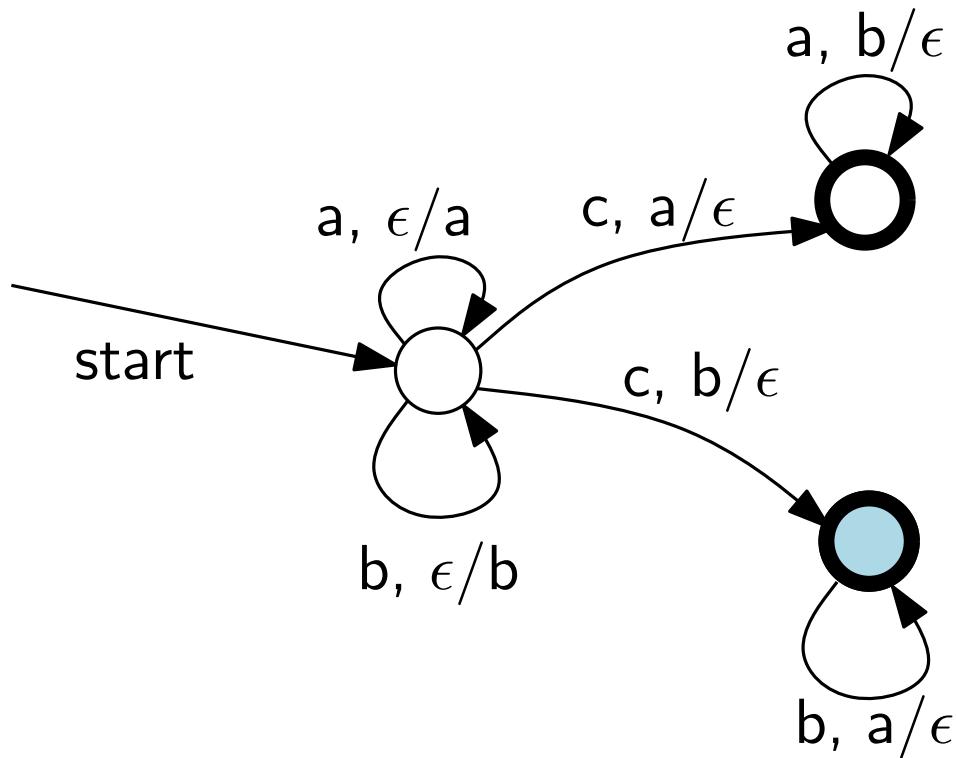
Stacken:

a poppas

b

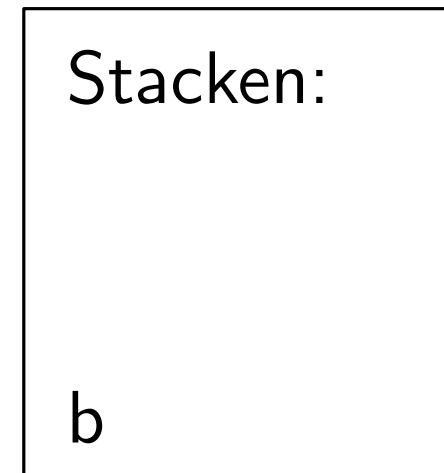
# Exempelkörning, stackautomat

$x, y/z$ : läs  $x$ , poppa  $y$   
från stacken, pusha  $z$



Indata:  
baabcbb**b**

Övergång saknas, för tecknet  $b$  måste  
vi ha ett  $a$  högst upp på stacken  
→ automaten accepterar inte





# Stackautomat för balanserade parentesuttryck

En grammatik för balanserade parentesuttryck:

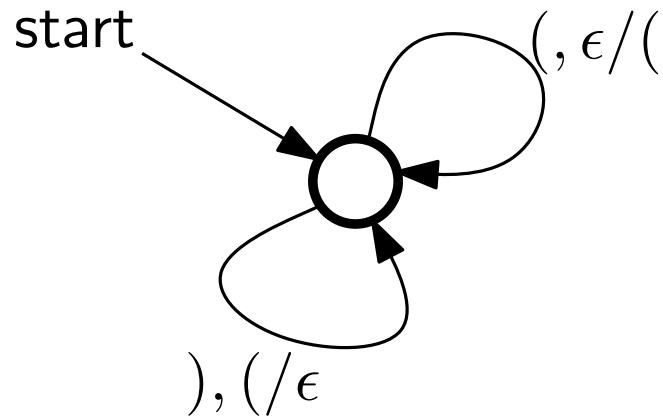
$$\text{Expr} \rightarrow \epsilon \mid (\text{Expr})\text{Expr}$$

# Stackautomat för balanserade parentesuttryck

En grammatik för balanserade parentesuttryck:

$$\text{Expr} \rightarrow \epsilon | (\text{Expr})\text{Expr}$$

Vi kan också lätt konstruera en PDA för språket, vi behöver faktiskt bara ett enda tillstånd! (Plus det implicita fail-tillståndet.)

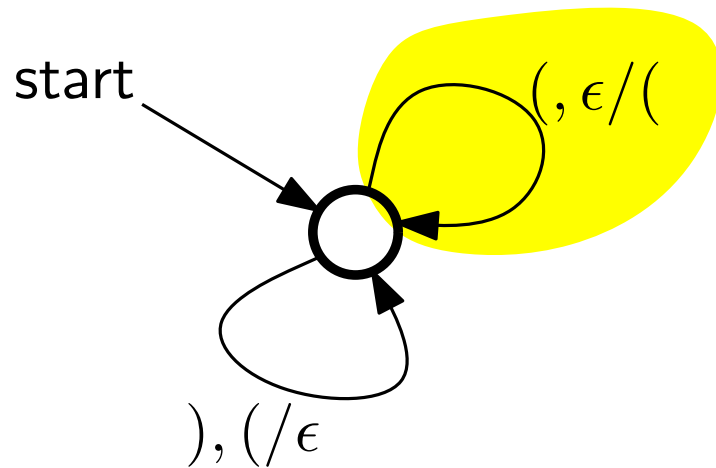


# Stackautomat för balanserade parentesuttryck

En grammatik för balanserade parentesuttryck:

$$\text{Expr} \rightarrow \epsilon | (\text{Expr})\text{Expr}$$

Vi kan också lätt konstruera en PDA för språket, vi behöver faktiskt bara ett enda tillstånd! (Plus det implicita fail-tillståndet.)



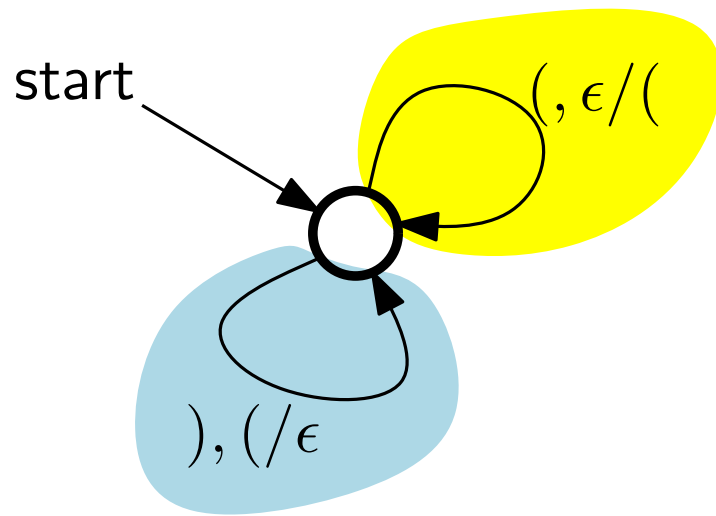
När vi ser en vänsterparentes, pusha på vänsterparentes på stacken

# Stackautomat för balanserade parentesuttryck

En grammatik för balanserade parentesuttryck:

$$\text{Expr} \rightarrow \epsilon | (\text{Expr})\text{Expr}$$

Vi kan också lätt konstruera en PDA för språket, vi behöver faktiskt bara ett enda tillstånd! (Plus det implicita fail-tillståndet.)



När vi ser en vänsterparentes, pusha på vänsterparentes på stacken

När vi ser högerparentes, poppa en vänsterparentes från stacken

# Från grammatiker till stackautomater

Parentesuttryck var enkelt att bygga stackautomat för.

# Från grammatiker till stackautomater

Parentesuttryck var enkelt att bygga stackautomat för.

Kan man alltid konvertera en grammatik till en stackautomat?

# Från grammatiker till stackautomater

Parentesuttryck var enkelt att bygga stackautomat för.

Kan man alltid konvertera en grammatik till en stackautomat?

**Nej, det kan man inte!**

# Från grammatiker till stackautomater

Parentesuttryck var enkelt att bygga stackautomat för.

Kan man alltid konvertera en grammatik till en stackautomat?

**Nej, det kan man inte!**

**Exempel:** Palindrom över  $\{a, b\}$

$\text{Palin} \rightarrow \epsilon \mid a \mid b \mid a \text{ Palin} \mid b \text{ Palin}$

Det existerar inte någon PDA som känner igen detta språk.



# Från grammatiker till stackautomater

Parentesuttryck var enkelt att bygga stackautomat för.

Kan man alltid konvertera en grammatik till en stackautomat?

**Nej, det kan man inte!**

**Exempel:** Palindrom över  $\{a, b\}$

Palin  $\rightarrow \epsilon \mid a \mid b \mid a$  Palin  $a \mid b$  Palin  $b$

Det existerar inte någon PDA som känner igen detta språk.

**Intuition:** PDA:n borde, när den kommit exakt halvvägs in i strängen, börja matcha av tecken mot de som den redan sett, men det finns inget sätt för den att veta när den är halvvägs.

# Från grammatiker till stackautomater

Parentesuttryck var enkelt att bygga stackautomat för.

Kan man alltid konvertera en grammatik till en stackautomat?

**Nej, det kan man inte!**

**Exempel:** Palindrom över  $\{a, b\}$

$\text{Palin} \rightarrow \epsilon \mid a \mid b \mid a \text{ Palin} \mid b \text{ Palin}$

Det existerar inte någon PDA som känner igen detta språk.

**Intuition:** PDA:n borde, när den kommit exakt halvvägs in i strängen, börja matcha av tecken mot de som den redan sett, men det finns inget sätt för den att veta när den är halvvägs.

Man kan bevisa detta med ett pumping-lemma för PDA:er, liknande beviset vi såg för DFA:er (beviset ingår ej i kursen).

# Grammatiker vs. stackautomater

Om stackautomater inte är tillräckligt kraftfulla för att kunna användas till alla grammatiker, vad är poängen?

# Grammatiker vs. stackautomater

Om stackautomater inte är tillräckligt kraftfulla för att kunna användas till alla grammatiker, vad är poängen?

Poängen är att de kan användas för “de flesta” grammatiker, t.ex. (nästan?) alltid de man får när man konstruerar programspråk.

Mer om detta nästa gång...

# Nästa föreläsning

## Grammatiker i praktiken

- Lexikal analys
- Härledning och syntaxträd
- Rekursiv medåkning

## Lite mer teori:

- Tvetydighet och omskrivning
- LL-grammatiker

