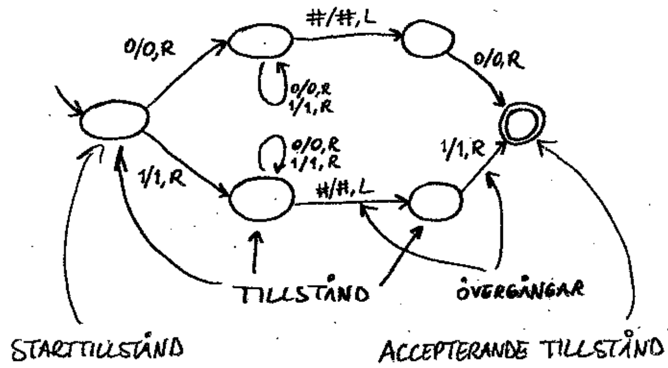
FÖRELÄSNING I ADK

- TURINGMASKINER
- CHURCHS TES
- FORMELLA DEFINITIONER
  - SPRÅK
  - P, NP
  - REDUKTION

TURINGMASKIN

EXEMPEL: KOLLA OM DEN BINÄRA STRÄNGEN PÅ BANDET (INMATNINGEN) BÖRJAR OCH SLUTAR MED SAMMA SIFFRA



$\xrightarrow{a/b,L}$ BETYDER: OM LÄSHUVUDET LÄSER a, FÖLJ ÖVERGÅNGEN, SKRIV b OCH FLYTTA HUVUDET ETT STEG TILL VÄNSTER

$\xrightarrow{a/b,R}$ SAMMA SAK, MEN FLYTTA HUVUDET ETT STEG TILL HÖGER ISTÄLLET

REGLER FÖR TURINGMASKIN

- AUTOMATEN BÖRJAR ALLTID I STARTTILLSTÅNDET
- DÅ STÅR LÄS/SKRIVHUVUDET PÅ FÖRSTA SYMBOLEN I INDATA. INDATA OMGES AV BLANKA (TECKNAS #)
- OM TURINGMASKINEN ÄR DETERMINISTISK FÅR DET INTE FINNAS FLERA ÖVERGÅNGAR MED SAMMA LÄSSYMBOL FRÅN SAMMA TILLSTÅND.
- OM TURINGMASKINEN HAMNAR I ETT ACCEPTERANDE TILLSTÅND AVBRYTS BERÄKNINGEN OCH JA RETURNERAS.
- OM TURINGMASKINEN HAMNAR I ETT LÄGE DÄR INGEN MATCHANDE ÖVERGÅNG FINNS SÅ AVBRYTS BERÄKNINGEN OCH NEJ RETURNERAS.

ÖVERGÅNGAR:

$\xrightarrow{a/b,L}$ BETYDER: OM LÄSHUVUDET LÄSER a, SKRIV b OCH FLYTTA HUVUDET ETT STEG ÅT VÄNSTER.

L – ETT STEG ÅT VÄNSTER
R – ETT STEG ÅT HÖGER
S – FLYTTA INTE HUVUDET

Churchs tes

> Varje algoritmiskt problem som kan lösas med något program skrivet i något språk kört på någon dator kan också lösas med en turingmaskin.

Följdsats:

> Beräkningsbarhet är robust

För bevis av övre gränser:
 Använd kraftfullt programspråk

För bevis av undre gränser:
 Använd Turingmaskinen

Definitioner av P och NP

$P = \{Q : \exists$ en TM som känner igen Q i pol. tid$\}$

En turingmaskin A <u>verifierar</u> instansen x till problemet Q om det finns en "lösning" y så att

$$A(x,y)=1 \Leftrightarrow x \in Q$$

Språket Q som verifieras av turingmaskinen A är

$Q = \{x \in \{0,1\}^* : \exists y \in \{0,1\}^* : A(x,y)=1\}$

$NP = \{Q : \exists$ en TM som verifierar Q i polynomisk tid$\}$

Alternativ definition av NP med NDTM

En NDTM (ickedeterministisk turingmaskin) <u>känner</u> igen språket Q i polynomisk tid om

$x \in Q \Rightarrow \exists$ kedja av övergångar av pol. längd som accepterar:
$x \notin Q \Rightarrow \nexists$ kedja av övergångar som accepterar x

$NP = \{Q : \exists$ en NDTM som känner igen Q i pol. tid$\}$

Definitionerna av NP är ekvivalenta!

Språk och beslutsproblem

Ett **<u>formellt språk</u>** är en mängd strängar.

Exempel:

$\{xy, yxx, xyzzy, zxy\}$
$\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \ldots\}$
$\{$syntaktiskt korrekta C-program$\}$
$\{$satisfierbara booleska formler$\}$

Olika sätt att beskriva språk:

- Räkna upp strängarna i språket
- En grammatik – regler som definierar språket
- En algoritm som <u>känner igen</u> strängarna i språket, dvs $A(x)=1$ omm x tillhör språket

Varje beslutsproblem motsvarar ett språk!

Nämligen språket som består av alla ja-instanser.

Turing- och Karpreduktioner

En Turingreduktion av problemet P till problemet Q är en algoritm för P som anropar en algoritm för Q en eller flera gånger.

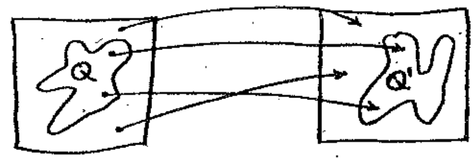
```
P(x) =
   :
  (här anropas algoritm för Q)
   :
  return y
```

En Karpreduktion av P till Q är det specialfall av Turingreduktion där P och Q är beslutsproblem, där algoritmen för Q bara anropas en gång, och där resultatet från den algoritmen direkt returneras

```
P(x) =
  konstruera en probleminstans y till Q
  return Q(y)
```

Polynomisk reduktion

Q kan reduceras till Q' om varje instans x av Q kan omformas till en instans x' av Q' så att $x \in Q \Leftrightarrow x' \in Q'$



Om reduktionen kan göras i polynomisk tid skriver vi $Q \leq_p Q'$ eftersom Q inte kan vara svårare att lösa än Q'.

Om $Q \leq_p Q'$ så gäller

$$Q' \in P \Rightarrow Q \in P$$

(om Q' är lätt så är Q också lätt)

$$Q \notin P \Rightarrow Q' \notin P$$

(om Q är svårt så är Q' också svårt)

Denna typ av reduktion kallas <u>Karp-reduktion</u>.