Föreläsning 22 i ADK

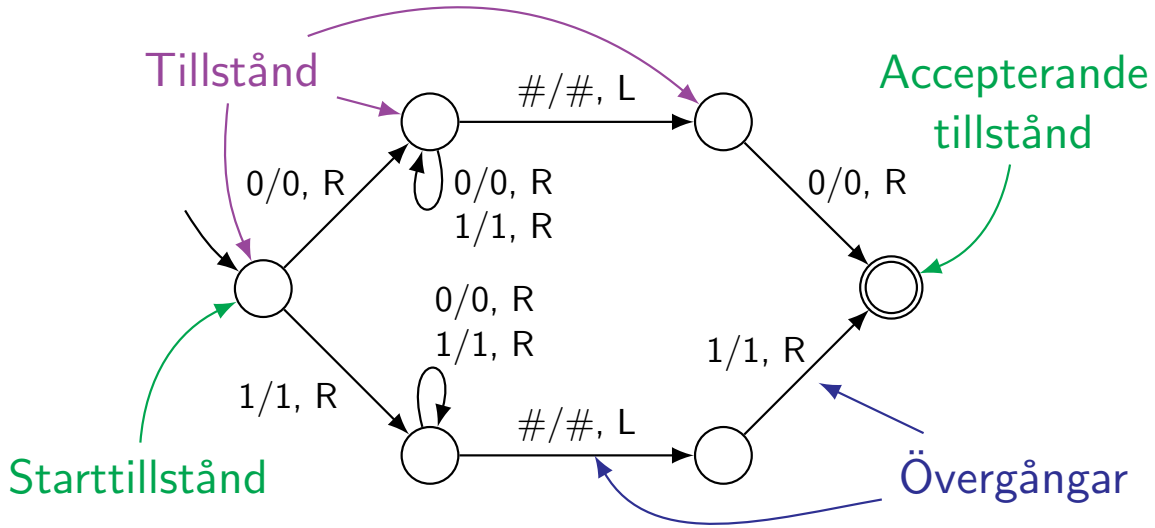
Formella definitioner, turingmaskiner

Viggo Kann

KTH

Turingmaskin

Exempel: kolla om den binära strängen på bandet (inmatningen) börjar och slutar med samma siffra.



$\xrightarrow{a/b,\ \mathrm{L}}$ Betyder: Om läshuvudet läser a, följ övergången, skriv b och flytta huvudet ett steg till vänster

$\xrightarrow{a/b,\ \mathrm{R}}$ 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:

Betyder: Om läshuvudet läser a, följ övergången, skriv b och flytta huvudet ett steg till vänster

- **L** - ett steg åt vänster
- **R** - ett steg åt höger
- **S** - Flytta inte på 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

Språk och beslutsproblem

Ett **formellt språk** är en mängd strängar.

Exempel:

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

Olika sätt att beskriva språk:

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

Varje beslutsproblem motsvarar ett språk!
Nämligen språket som består av alla ja-instanser

Definitioner av $\mathcal{P}$ och $\mathcal{NP}$

$\mathcal{P} = \{Q : \exists$ en turingmaskin som känner igen Q i polynomisk tid$\}$

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

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

Alternativ definition av $\mathcal{NP}$ med NDTM

En NDTM (ickedeterministisk turingmaskin) känner igen språket Q i polynomisk tid om:

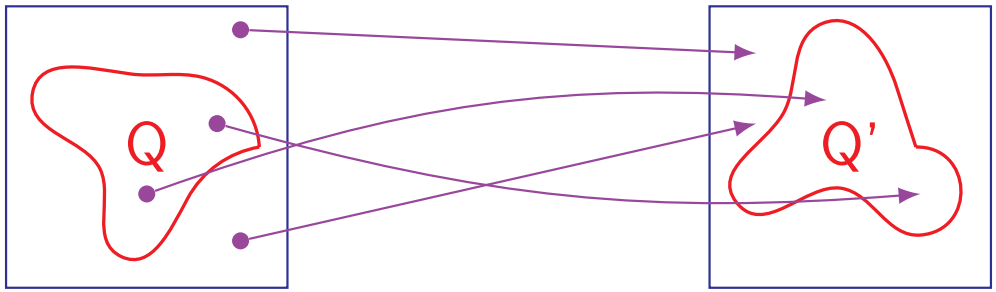
$$\begin{cases} x \in Q \Rightarrow \exists \text{ kedja av övergångar av pol. längd som accepterar } x \\ x \notin Q \Rightarrow \nexists \text{ kedja av övergångar som accepterar } x \end{cases}$$

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

Definitionerna av $\mathcal{NP}$ är ekvivalenta!

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 $Q \leqslant_p Q'$ eftersom Q inte kan vara svårare att lösa än Q'

Om $Q \leqslant_p Q'$ 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 **Karp-reduktion**