DD1350 Logik för dataloger

Fö 6 – Mängder

1

Mängder

En **mängd** (eng. *set*) är en samling objekt, t.ex.

{ *a, b, c* }
{ *nisse*, *KTH*, 17, *röd* }
$\mathbb{N}$ = de naturliga talen { 0, 1, 2, … }
$\mathbb{Z}$ = de hela talen { 0, 1,–1, 2, –2, … }
$\mathbb{Z}_+$ = de positiva hela talen { 1, 2, 3, … }
$\mathbb{Q}$ = de rationella talen
$\mathbb{R}$ = de reella talen

Tillhörighet

Om ett element tillhör mängden A skrivs detta

$x \in A$

t. ex.

$1 \in \{ 1, 2, 3 \}$

$x \notin A$ är ett annat sätt att skriva $\neg(x \in A)$, t.ex.

$4 \notin \{ 1, 2, 3 \}$

Mängder i mängder

Mängder kan vara element i mängder, t.ex. :

$A = \{ 1, 2, \{3, 4\}, 5 \}$

och

$B = \{ a, \mathbb{N} \}$

(Hur många element har A resp. B?)

Delmängd

Om $\forall x\, (x \in A \rightarrow x \in B)$ så är A en **delmängd** till B.
Detta skrivs: $A \subseteq B$

A och B är samma mängd ($A = B$) om $A \subseteq B$ och $B \subseteq A$.

Om $A \subseteq B$ men $A \neq B$, så är A en **äkta delmängd** till B.
Detta skrivs: $A \subset B$.

T.ex. om $A = \{ 1, 2, 3 \}$ och $B = \{ 1, 2, 3, 4 \}$ så är $A \subset B$ men $B \not\subset A$.

Skapa mängder

En ändlig mängd kan (i princip) skapas genom att lista alla dess element, t.ex. $\{ 1, 2, 3, 4 \}$.

Man kan också skapa en mängd genom att ange en egenskap som alla element i mängden ska ha, t.ex.
$\mathbb{N}_{\mathbf{Jämn}} = \{ x \mid \exists y(y \in \mathbb{N} \wedge x = 2y) \}$ (alla jämna tal)

läses: ”mängden av alla x sådana att …”

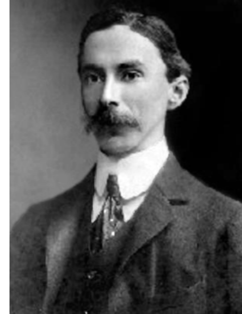
Detta kan dock ställa till problem…

Russells paradox

Definiera M som mängden av alla mängder som inte innehåller sig själva, alltså:

$M = \{ A \mid A \notin A \}$

Nu är frågan: Är det så att $M \in M$?



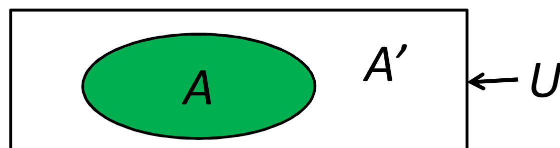
Bertrand Russell (1872-1970)

Universum och mängdkomplement

För att undvika Russells paradox krävs att alla mängder är delmängder av ett känt **universum** U.

Exempelvis: $A = \{ x \in U \mid P(x) \}$ ← Alla element i *U* som har egenskapen *P*.

Detta gör det meningsfullt att prata om **komplementet** A' till en mängd A: $A' = \{ x \in U \mid x \notin A \}$

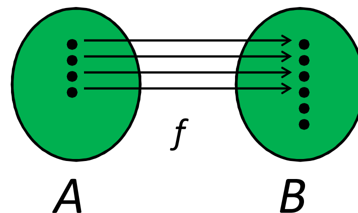


Kardinalitet

Kardinaliteten $|A|$ till en mängd A är ett mått på dess storlek.

$|A| \leq |B|$ om det för varje element i A finns ett unikt element i B.

dvs om det finns en funktion $f : A \rightarrow B$ sådan att om $a_1 \neq a_2$, $f(a_1)=b_1$ och $f(a_2)=b_2$ så är $b_1 \neq b_2$.



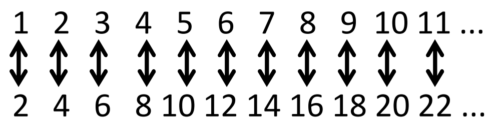
f säges då vara en **injektiv** funktion.

Kardinalitet

För ändliga mängder gäller att $|A| < |B|$ om $A \subset B$.
Detta gäller dock inte oändliga mängder!
T. ex. är $\mathbb{N}_{\text{Jämn}} \subset \mathbb{N}$, men $|\mathbb{N}_{\text{Jämn}}| = |\mathbb{N}|$ (dvs det finns lika många jämna tal som det finns tal).

Definiera $f : \mathbb{N} \rightarrow \mathbb{N}_{\text{jämn}}$ som $f(x) = 2x$ och
$g: \mathbb{N}_{\text{jämn}} \rightarrow \mathbb{N}$ som $g(x) = x/2$

Både *f* och *g* är **injektiva**.



Uppräkningsbara mängder

En mängd är **uppräkningsbar** om den har samma kardinalitet som $\mathbb{N}$.

I en uppräkningsbar mängd kan vi lista alla element $\{e_1, e_2, \ldots\}$ så att varje element förekommer på något index *k* i listan.

Föregående bild visade att de jämna talen är uppräkningsbara. Men det finns många fler exempel...

$\mathbb{Q}$ är uppräkningsbar

	1	2	3	4	...
1	1/1	1/2	1/3	1/4	...
2	2/1	2/2	2/3	2/4	...
3	3/1	3/2	3/3	3/4	...
4	4/1	...	...	...	...
...					

Mängden av strängar är uppräkningsbar

Mängden av alla strängar över ett ändligt alfabet (t.ex. a-z) är uppräkningsbar.

Lista först alla strängar av längd 1, i bokstavsordning:

a, b, c, ... , z

därefter längd 2:

aa, ab, ac, ..., ba, bb, ..., yz, zz

därefter längd 3, osv.

Detta innebär att {**alla Java-program**}, {**predikatlogiska formler**}, osv är **uppräkningsbara** mängder.

$2^{\mathbb{N}}$ är inte uppräkningsbar

Mängden av alla delmängder till $\mathbb{N}$ (skrivs $2^{\mathbb{N}}$) är **inte** uppräkningsbar. Betrakta en lista av delmängder till $\mathbb{N}$:

$$S_0, S_1, S_2, \ldots$$

och definiera mängden $D = \{ n \in \mathbb{N} \mid n \notin S_n \}$

Det följer från def att D är en delmängd till $\mathbb{N}$, men ändå skiljer sig från alla mängder S_0, S_1, S_2, ...

För varje tänkbar lista av delmängder till $\mathbb{N}$ kan man konstruera en sådan mängd D som inte är med i listan. Alltså är inte $2^{\mathbb{N}}$ uppräkningsbar.

Man kan däremot visa att $|2^{\mathbb{N}}| = |\mathbb{R}|$.

Diagonalisering

Bevistekniken på föregående bild kallas **diagonalisering** och upptäcktes av Georg Cantor (1845 – 1918).

Vi vet alltså att $|\mathbb{N}| < |2^{\mathbb{N}}|$. Frågan är om det finns någon mängd *A* sådan att: $|\mathbb{N}| < |A| < |2^{\mathbb{N}}|$?

Cantor gissade nej men kunde inte bevisa detta. Kallas för **kontinuumhypotesen.**

Rekursivt uppräkningsbara mängder

En mängd *M* är **rekursivt uppräkningsbar** om man kan skriva ett datorprogram *P* som givet ett objekt *x* ger resultatet ”ja” precis när $x \in M$.

Om $x \notin M$ så kan *P* svara ”nej” eller **inte terminera** alls.

Alla dessa mängder är rekursivt uppräkningsbara (och t.om. **rekursiva**; se nästa bild): {alla Java-program}, {alla predikatlogiska formler}, $\mathbb{N}, \mathbb{N}_{\textbf{Jämn}}, \mathbb{Q}$

Rekursiva mängder

En mängd M är **rekursiv** om om man kan skriva ett datorprogram P som givet ett objekt x ger resultatet ”ja” precis när $x \in M$, och ”nej” precis när $x \notin M$.

Dvs, P terminerar **alltid** och svarar alltid rätt.

Om M är rekursiv så säges problemet att bestämma huruvida $x \in M$ eller ej vara **avgörbart.**

M är rekursiv om och endast om både M och M' (=komplement-mängden till M) är rekursivt uppräkningsbara.

Ett oavgörbart problem

Betrakta alla möjliga Javaprogram J_1, J_2, ... som tar ett heltal som resultat och returnerar ett heltal som svar (eller aldrig terminerar).

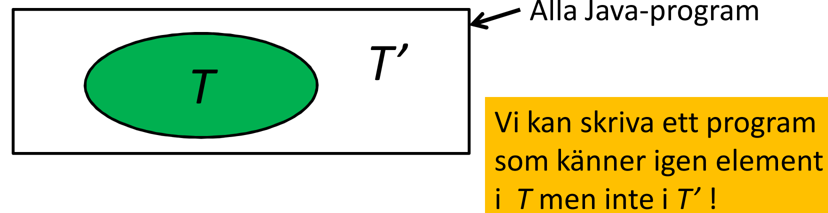
Definiera $D(n) = \begin{cases} 1 & \text{om } J_n(n) \text{ inte terminerar} \\ J_n(n)+1 & \text{annars} \end{cases}$

Antag att D kan implementeras (dvs $D = J_k$ för något k). Men då kommer $D(k)$ antingen både returnera 1 och inte terminera, eller returnera både x och x+1. **Motsägelse:** alltså finns inget program som beräknar D.

Termineringsproblemet

Diagonaliserings-resonemanget på föregående bild visar att **termineringsproblemet** (eng. *halting problem*) är ett oavgörbart problem.

Mängden *T* av terminerande Java-program är rekursivt uppräkningsbar men **inte rekursiv**.



Vad innebär termineringsproblemet?

Att termineringsproblemet är oavgörbart betyder **inte** att vi **aldrig** kan säga om ett program terminerar.

Program 1:

```
print( ”hej” );
```

Terminerar

Program 2:

```
while (true )
   { print(”hej”); }
```

Terminerar inte

Men det innebär att vi **inte kan hitta en systematisk metod** som **alltid** avgör, givet ett program P och indata x, ifall P terminerar på x eller ej.