

# MAT1030 – Diskret Matematikk

## Forelesning 14: Mer om funksjoner

Dag Normann

Matematisk Institutt, Universitetet i Oslo

3. mars 2010

(Sist oppdatert: 2010-03-03 14:59)



# Kapittel 6: Funksjoner

# Injektive funksjoner

- I går begynte vi på kapitlet om funksjoner

$$f : X \rightarrow Y,$$

og vi brukte betegnelsene

- **definisjonsområdet** for  $X$
  - **verdiområdet** for  $Y$
  - **bildemengden**  $f[X]$  for  $\{f(x) : x \in X\}$ .
- Det som kjennetegner en funksjon er at det for alle  $x \in X$  finnes **en og bare en**  $y \in Y$  slik at  $y = f(x)$ .
  - Vi gikk deretter over til å se på **injektive** funksjoner:

$$x \neq y \Rightarrow f(x) \neq f(y).$$

# Injektive funksjoner

- Vi vil normalt bruke **Injektive** eller **enentydige** funksjoner når vi skal registrere fysiske objekter digitalt.
- Det er da ofte et poeng at verdiområdet er vesentlig større enn bildemengden.
- Et slående eksempel er registreringen av personer som et 11-sifret tall i form av **fødselsnummer**.
- Det er viktig at forskjellige personer har forskjellige fødselsnummer.
- Det er også viktig at det finnes langt flere 11-sifrede tall enn personer.
- På denne måten kan de virkelige fødselsnumrene fordeles slik at hvis man skriver feil, vil resultatet normalt ikke bli et gyldig fødselsnummer.

# Injektive funksjoner

- Hver bankkonto vil ha et kontonummer, også med 11 siffer.
- Forskjellige konti har forskjellige kontonummer.
- Mange konti tillater bruk av betalingskort, og mange kort er utstyrt med PIN-koder.
- Funksjonen som gir PIN-koden til et kort er ikke enentydig.
- Foreleser hadde en gang to kort med samme PIN-kode, et bankkort og et adgangskort til universitetets bygninger.

# Injektive funksjoner

## Eksempel

- Enkelte ganger kan det være aktuelt å bake en lokal relasjonsdatabase inn i en større.
- La oss anta at endel lokale bibliotek har digitalisert sine bokregistre, og at man nå ønsker å lage en nasjonal base for alle landets biblioteker.
- Da må vi konstruere en injektiv funksjon for hvert lokale bibliotek, som sender representasjonen av hver enkelt bok i den lokale basen på en representasjon av samme bok i den nasjonale basen.

# Injektive funksjoner

## Eksempel (fortsatt)

- Det vil finnes noen funksjoner, som forfatter, utgivelsesår, forlag m.m. som skal bevares.
- I det hele skal all informasjon lagret i den lokale basen kunne gjenfinnes i den nasjonale, mens den nasjonale basen gjerne kan inneholde mer informasjon om hver enkelt bok.
- En injektiv funksjon som bevarer informasjon på denne måten, kalles ofte for en **embedding**.
- Vi skal ikke gi en presis matematisk definisjon av hva vi mener med en *embedding* i disse forelesningene, men injektive funksjoner, i form av *embeddings*, spiller en stor rolle i forståelsen av sammenhenger mellom forskjellige relasjonsdatabaser.

# Injektive funksjoner

## Eksempel

- Et eksempel på embeddinger er **strengt voksende funksjoner**.
- Hvis  $A$  og  $B$  er to mengder,  $R$  er en relasjon på  $A$  og  $S$  er en relasjon på  $B$ , er en injektiv funksjon  $f : A \rightarrow B$  en **embedding** hvis vi for alle  $x$  og  $y$  i  $A$  har

$$xRy \Leftrightarrow f(x)Sf(y).$$

- Hvis  $R$  og  $S$  er **ordninger** svarer dette til at  $f$  er strengt voksende.



# Surjektive funksjoner

Den neste gruppen av funksjoner vi skal se på er de **surjektive** funksjonene:

## Definisjon

La  $f : X \rightarrow Y$  være en funksjon.

$f$  kalles **surjektiv** hvis bildemengden til  $f$  er hele  $Y$ .

På engelsk brukes ofte betegnelsen **onto** og på norsk kan vi si at  $f$  går fra  $X$  **på**  $Y$ .

# Surjektive funksjoner

## Eksempel (Surjektive funksjoner)

- La  $f : \mathbb{R} \rightarrow \mathbb{R}_{\geq 0}$  være gitt ved

$$f(x) = x^2.$$

Da er  $f$  surjektiv.

- $f_A(B) = A \cap B$  er surjektiv som en funksjon fra potensmengden til  $\mathcal{E}$  til potensmengden til  $A$ .
- La  $\text{PRIM} : \mathbb{N} \rightarrow \mathbb{N}$  være definert ved at  $\text{PRIM}(n)$  er primtall nr.  $n$ .

Da er ikke  $\text{PRIM}$  en surjektiv funksjon, fordi verdimengden er hele  $\mathbb{N}$ , mens bildemengden er mengden av primtall.

# Surjektive funksjoner

## Eksempel (Fortsatt)

- Enhver funksjon vil være surjektiv hvis vi setter verdiområdet lik bildemengden.
- Det er ofte i de tilfellene hvor det er uklart hva bildemengden er at det kan være relevant å spørre seg om en funksjon er surjektiv eller ikke.
- Selv om vi finner en algoritme for å beregne en

$$f : \mathbb{N} \rightarrow \mathbb{N}$$

trenger det ikke finnes noen algoritme for å avgjøre om  $n \in f[\mathbb{N}]$ .  
En funksjon  $f$  og bildemengden til  $f$  kan ha forskjellig **kompleksitet**.

# Surjektive funksjoner

## Merk

- I dette kurset vil injektive funksjoner spille en større rolle enn surjektive funksjoner.
- Hvis vi konstruerer digitale representasjoner for elementene i en mengde  $A$ , konstruerer vi samtidig en funksjon  $F$  fra representantene for elementene i  $A$  til  $A$  selv.
- Vi vil normalt ønske å vite om vi får representert alle elementene i  $A$  (håpløst hvis ikke  $A$  er endelig).
- Dette er det samme som å spørre om  $F$  er surjektiv.

# Sammensetning av funksjoner

## Eksempel (Sammensatte funksjoner)

- Anta at vi har en maskin som benytter 16-bit representasjoner av hele tall.
- La  $a$  være et helt tall slik at

$$-32767 \leq a \leq 32767.$$

- Hvis vi skal beregne  $f(a) = -a$  på maskinen trenger vi å
  1. Finne den digitale representasjonen av  $a$ .
  2. Beregne den digitale representasjonen av  $-a$  fra den digitale representasjonen til  $a$
  3. Finne  $-a$  ut fra den digitale representasjonen til  $-a$ .
- Vi ser at det er tre funksjoner involvert her, og vi har bruk for **sammensetningen** av dem.

# Sammensetning av funksjoner

## Eksempel (Sammensatte funksjoner)

- I skolematematikken, og i noen kurs i matematisk analyse, ser vi på sammensetninger av funksjoner definert på  $\mathbb{R}$  eller delmengder av  $\mathbb{R}$ .
- Kjernerregelen for derivasjon forteller oss hvordan vi kan derivere slike sammensetninger.

# Sammensetning av funksjoner

## Eksempel (Sammensatte funksjoner)

- La oss se på følgende pseudokode.
1. *Input*  $x$  [ $x$  naturlig tall]
  2.  $y \leftarrow 1$
  3. **While**  $x > 0$  **do**
    - 3.1  $y \leftarrow 2y$
    - 3.2  $x \leftarrow x - 1$
  4.  $z \leftarrow 1$
  5. **While**  $y > 0$  **do**
    - 5.1  $z \leftarrow 3z + 1$
    - 5.2  $y \leftarrow y - 1$
  6. *Output*  $z$

# Sammensetning av funksjoner

## Eksempel (Fortsatt)

- Denne pseudokoden deler seg naturlig i to deler.
- Instruksjonene 1. - 3. beregner  $y = f(x) = 2^x$ .
- Instruksjonene 4. - 6. beregner  $z$  som en funksjon  $z = g(y)$ , hvor vi ikke har noen opplagt formel for  $g(y)$ .
- Tilsammen vil pseudokoden definere en sammensatt funksjon  $z = g(f(x))$ .



# Sammensetning av funksjoner

## Definisjon

La  $f : X \rightarrow Y$  og  $g : Y \rightarrow Z$  være to funksjoner.

Vi definerer **sammensetningen**  $h = g \circ f$  som funksjonen

$$h : X \rightarrow Z$$

vi får ved først å bruke  $f$  på argumentet  $x$  og så  $g$  på **mellomverdien**  $f(x)$ .

Vi skriver også

$$h(x) = g(f(x)).$$

# Sammensetning av funksjoner

## Eksempel

- La  $f : \mathbb{N} \rightarrow \mathbb{N}$  være definert ved at  $f(n)$  er primtall nummer  $n$  og la  $g : \mathbb{N} \rightarrow \mathbb{N}$  være definert ved at  $g(m) = m^2$   
Da er  $(g \circ f)(4) = g(f(4)) = g(7) = 49$ .  
 $(f \circ g)(4) = f(g(4)) = f(16) = 53$ .
- Vi ser altså at selv om sammensetningen av funksjonene gir mening for begge rekkefølgene, kan det spille en rolle i hvilken rekkefølge vi setter dem sammen.

# Sammensetning av funksjoner

## Eksempel (Fortsatt)

- La  $f$  sende binærformen til et naturlig tall  $n$  over til desimalformen og la  $g$  gi oss tverrsummen av desimalformen til et tall.

Da gir det ingen mening å snakke om  $f \circ g$  fordi definisjonsområdet til  $f$  er mengden av binære representasjoner av naturlige tall og verdiområdet til  $g$  er en mengde av naturlige tall.

I MAT1030 er det viktig å skille mellom tall og de forskjellige representasjonene av tallene.

$g \circ f$  gir mening. Definisjonsområdet vil være mengden av binære representasjoner, “mellomområdet” vil være mengden av desimaltallsrepresentasjoner og verdiområdet vil være naturlige tall.

$$(g \circ f)(100110) = g(38) = 11.$$

# Sammensetning av funksjoner

## Teorem

La  $f : X \rightarrow Y$  og  $g : Y \rightarrow Z$  være to funksjoner.

La  $h = g \circ f$  være sammensetningen av  $f$  og  $g$ .

- a) Hvis både  $f$  og  $g$  er injektive, er  $h$  injektiv.
- b) Hvis både  $f$  og  $g$  er surjektive, er  $h$  surjektiv.

# Sammensetning av funksjoner

## Bevis

a) Anta at  $f$  og  $g$  er injektive.

$$x \neq y \Rightarrow f(x) \neq f(y) \Rightarrow g(f(x)) \neq g(f(y)).$$

Siden  $\Rightarrow$  er *transitiv*,  $h(x) = g(f(x))$  og  $h(y) = g(f(y))$  følger det at  $h$  er injektiv når  $f$  og  $g$  er det.

b) Anta at  $f$  og  $g$  er surjektive, og la  $z \in Z$  være vilkårlig.

Siden  $g$  er surjektiv, fins  $y \in Y$  slik at  $g(y) = z$ .

Siden  $f$  er surjektiv, fins  $x \in X$  slik at  $f(x) = y$ .

Da er  $z = g(f(x)) = h(x)$ .

## Sammensetning av funksjoner

- For mange programmeringsspråk kan gjennomkjøringen av et program oppfattes som en styrt sammensetning av mange funksjoner.
- Vi så på et eksempel hvor en pseudokode beskrev en algoritme for en sammensatt funksjon.
- Egentlig kan enhver instruksjon

$$x_i \leftarrow t(x_1, \dots, x_n)$$

i en pseudokode oppfattes som en ordre om at vi skal bruke funksjonen  $t$  der og da.

- Siden det å bruke disse funksjonene utgjør enkeltrinnene i beregningen, får vi resultatet etter å ha satt sammen slike instruksjoner.

- **Advarsel!**

Programmer, eller pseudokoder, skal egentlig oppfattes som en **generalisering** av sammensatte funksjoner, ettersom rekkefølgen vi bruker funksjonene i avhenger av startverdien på variablene.

## Eksempel (Inverse funksjoner)

- Med fare for å overfokusere på et tema skal vi nok en gang se på digital representasjon av tall.
- La  $X$  være mengden av reelle tall som har en digital representasjon med enkel presisjon.
- La  $Y$  være mengden av 32-bits representasjoner av reelle tall.
- La  $F : X \rightarrow Y$  være funksjonen som til et tall  $x$  gir oss den digitale representasjonen av  $x$ .
- La  $G : Y \rightarrow X$  være funksjonen som til en digital representasjon  $y$  av et tall gir oss tallet.
- Da er  $G(F(x)) = x$  for alle  $x \in X$  og  $F(G(y)) = y$  for alle  $y \in Y$ .
- Vi sier at  $G$  er den **inverse** av  $F$ .

# Inverse funksjoner

## Eksempel (Inverse funksjoner)

- Fra skolematematikken og begynneremnene i matematikk har vi flere eksempler på par av funksjoner som “opphever” hverandre:
  1.  $f : \mathbb{R} \rightarrow \mathbb{R}_{>0}$  definert ved  $f(x) = e^x$  og  $g : \mathbb{R}_{>0} \rightarrow \mathbb{R}$  definert ved  $g(y) = \ln(y)$ .
  2. Tangensfunksjonen  $f : \langle -\frac{\pi}{2}, \frac{\pi}{2} \rangle \rightarrow \mathbb{R}$  og Arcustangens-funksjonen  $g : \mathbb{R} \rightarrow \langle -\frac{\pi}{2}, \frac{\pi}{2} \rangle$ .
  3.  $f(x) = 2x$  og  $g(y) = \frac{y}{2}$
- Vi har utallige eksempler på par av funksjoner som representerer “omvendte regningsmåter av hverandre”.



# Inverse funksjoner

## Eksempel (Inverse funksjoner)

- La  $f : \mathbb{J} \rightarrow \mathbb{J}$  være definert ved at  $f(a)$  er heltallsverdien til  $\frac{a}{2}$ , det vil si det største hele tallet  $b$  slik at  $b \leq \frac{a}{2}$ .
- Kan vi finne en omvendning av  $f$ ?
- La oss regne på noen verdier, og la oss se hva som skjer:
- $\dots$ ,  $f(-2) = f(-1) = -1$ ,  $f(0) = f(1) = 0$ ,  $f(2) = f(3) = 1$ ,  $\dots$
- Hvis vi skal lage en omvendt funksjon  $g$  har vi to valg for  $g(1)$ , nemlig  $g(1) = 2$  og  $g(1) = 3$ .
- Velger vi  $g(1) = 2$ , vil  $g(f(3)) \neq 3$ , og velger vi  $g(1) = 3$  får vi  $g(f(2)) \neq 2$ .
- Vi ser at siden  $f$  ikke er injektiv, får vi et problem.

# Inverse funksjoner

## Eksempel (Inverse funksjoner)

- La  $P : \mathbb{N} \rightarrow \mathbb{N}$  være definert ved at  $P(n)$  er primtall nummer  $n$  i den voksende opplistingen

$$2, 3, 5, 7, 11, 13, 17, 19, \dots$$

- $P$  er injektiv, men  $P$  har ingen omvendt funksjon  $Q$ .
- Problemet er at vi ikke kan definere eksempelvis  $Q(15)$  siden 15 ikke er et primtall og derfor ikke har noe nummer.
- Hvis vi for eksempel prøver oss med  $Q(15) = 7$ , får vi

$$P(Q(15)) = 17 \neq 15.$$

- Vi ser at problemet ligger i at  $P$  ikke er surjektiv.

# Inverse funksjoner

## Definisjon

La  $f : X \rightarrow Y$  være en funksjon.

$g : Y \rightarrow X$  kalles en **invers** til  $f$ , eller en **omvendt funksjon** av  $f$ , hvis

- $g(f(x)) = x$  for alle  $x \in X$ .
- $f(g(y)) = y$  for alle  $y \in Y$ .

# Inverse funksjoner

## Teorem

a) La  $f : X \rightarrow Y$  være en funksjon.

Da har  $f$  en invers funksjon  $g : Y \rightarrow X$  hvis og bare hvis  $f$  er både injektiv og surjektiv.

b) En funksjon  $f$  kan ikke ha mer enn én invers.

## Definisjon

Hvis  $f : X \rightarrow Y$  har en invers, skriver vi  $f^{-1}$  for den inverse.

# Inverse funksjoner

## Bevis

- Anta først at  $f : X \rightarrow Y$  er både injektiv og surjektiv.

La  $y \in Y$ .

Siden  $f$  er surjektiv, fins det  $x \in X$  slik at  $f(x) = y$ , og siden  $f$  er injektiv fins det bare en slik  $x$ .

Da lar vi  $g(y) = x$ , og  $g$  vil være en invers.

# Inverse funksjoner

## Bevis (Fortsatt)

- Anta så at  $f$  har en invers  $g$ .
  - $x \neq y \Rightarrow g(f(x)) \neq g(f(y)) \Rightarrow f(x) \neq f(y)$ .  
Derfor er  $f$  injektiv.
  - La  $y \in Y$  og la  $x = g(y)$ .  
Da er  $y = f(x)$ .  
Det følger at  $f$  er surjektiv.
- Definisjonen under første punkt er den eneste måten å finne en invers på, så det kan ikke fins flere.

# Inverse funksjoner

## Oppgave

- a) Vis at hvis  $f : X \rightarrow Y$  og  $g : Y \rightarrow Z$  begge har inverse  $f^{-1}$  og  $g^{-1}$ , så vil sammensetningen

$$h = g \circ f$$

også ha en invers.

- b) Under antagelsene i a), vis at  $h^{-1} = f^{-1} \circ g^{-1}$ .

# Inverse funksjoner

- Vi kan bruke kvantorer til å gi presise definisjoner av *injektiv*, *surjektiv* og *sammensetning*:
- $f : X \rightarrow Y$  er **injektiv** hvis  $\forall x \in X \forall y \in X (f(x) = f(y) \rightarrow x = y)$ .
- $f : X \rightarrow Y$  er **surjektiv** hvis  $\forall y \in Y \exists x \in X (y = f(x))$ .
- Hvis  $f : X \rightarrow Y$ ,  $g : Y \rightarrow Z$  og  $h = g \circ f$  vil

$$\forall x \in X \forall z \in Z (h(x) = z \leftrightarrow \exists y \in Y (y = f(x) \wedge z = g(y))).$$

De som ønsker å forstå hvordan kvantorer brukes, bør overbevise seg selv om at dette er riktig.



# Funksjoner og programmering

- De fleste programmeringsspråk er utstyrt med predefinerte funksjoner.
- For noen av disse språkene kan man også definere sine egne funksjoner.
- En lommeregner har eksempelvis knapper for algebraiske operasjoner, trigonometriske funksjoner, logaritme- og eksponensialfunksjoner m.m.
- Noen lommeregnere tillater også at vi definerer våre egne funksjoner ved hjelp av sammensatte uttrykk, og tildels enkle programmer.
- Matematisk sett opererer disse funksjonene på representasjoner av tall og andre størrelser i lommeregneren eller på datamaskinen.
- Dette diskuteres i tilstrekkelig detalj i læreboka.

## Funksjoner og programmering

- Noen hjelpemidler kan gå under betegnelsen “funksjoner” i en programmeringssammenheng, men er ikke funksjoner i vanlig matematisk forstand.
- De viktigste er de som genererer et tilfeldig element i en mengde.
- Ber vi om en kabal, eller et minesveiperspill, får vi et nytt spill hver gang.
- Skal vi teste om et stort tall  $n$  er et primtall utfører vi en algebraisk test på tilfeldig valgte tall  $a_1, \dots, a_k$  mindre enn  $n$ .  
For hver test vil svaret *NEI* fortelle oss at  $n$  ikke er et primtall, mens svaret *JA* forteller oss at sannsynligheten er  $\frac{3}{4}$  for at  $n$  er et primtall.
- Denne primtallstesten kan gi forskjellige svar på om  $n$  er et primtall, og er derfor ikke en funksjon i matematisk forstand.
- Ved å la antall vilkårlige deltester være stort, eksempelvis 100, er usikkerheten omkring svaret i størrelsesorden  $2^{-200}$ , så for alle praktiske formål er primtallstesten en funksjon.

# Beregnbare funksjoner

- IT dreier seg mye om hvordan man løser oppgaver ved hjelp av elektroniske hjelpemidler, fortrinnsvis datamaskiner.
- All IT-aktivitet på maskin-nivå styres av **programmer**, uansett om vi ser dem eller ikke.
- Hvis man skal kunne forstå informasjonsteknologiens begrensninger, må vi derfor forstå grensene for hva det er mulig å skrive programmer for.
- Alle programmer beskriver egentlig funksjoner, selv om noen argumenter (som maskintid, maskinarkitektur o.a.) ikke er synlig.
- Det er derfor av interesse å studere de funksjonene som lar seg uttrykke ved hjelp av programmer.

# Beregnbare funksjoner

- Hvis vi begrenser oss til funksjoner fra  $\mathbb{N}_0$  til  $\mathbb{N}_0$  har vi gode matematiske karakteriseringer av de **beregnbare** funksjonene, det vil si de som kan programmeres i et eller annet programmeringsspråk. ( $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$ )
- Det viser seg at alle programmerbare funksjoner fra  $\mathbb{N}_0$  til  $\mathbb{N}_0$  kan formuleres som en av våre pseudokoder, hvor vi bare bruker navn på tallene 0 og 1, addisjon og multiplikasjon og Booleske tester uttrykt ved hjelp av  $=$  og  $<$ .

Det er ikke uvanlig for logikere eller folk som arbeider med teoretisk databehandling å la de naturlige tallene starte med 0.

Vi skal være snille og holde oss til måten boka gjør det på.

# Beregnbare funksjoner

Som en forberedelse til kapittel 7 om induksjon og rekursjon, skal vi se på to pseudokoder hvor vi har pålagt oss å begrense oss til addisjon, multiplikasjon og Boolske tester med  $=$  og  $<$  (men hvor vi dermed får lov til å bruke  $\leq$ ).

- I det første eksemplet skal vi beregne  $f(x, y) = \max\{0, x - y\}$ .
- I det andre eksemplet skal vi beregne  $g(x, y) = x^y$ .

# Beregnbare funksjoner

## Eksempel (Beregnbare funksjoner)

1. *Input*  $x$  [ $x \in \mathbb{N}_0$ ]
2. *Input*  $y$  [ $y \in \mathbb{N}_0$ ]
3.  $z \leftarrow 0$
4. **While**  $y < x$  **do**
  - 4.1  $y \leftarrow y + 1$
  - 4.2  $z \leftarrow z + 1$
5. *Output*  $z$

- Vi har ikke snakket om **induksjonsbevis** ennå. Det vil være den naturlige metoden for å vise korrekthet av et slikt program.
- I dette tilfellet ser vi at hvis  $x \leq y$  starter vi ikke løkka i det hele tatt, mens hvis  $y < x$  “teller” vi  $y$  opp til  $x$  samtidig som vi øker verdien av  $z$  tilsvarende mye.

# Beregnbare funksjoner

## Eksempel (Beregnbare funksjoner)

1. *Input*  $x$  [ $x \in \mathbb{N}_0$ ]
2. *Input*  $y$  [ $y \in \mathbb{N}_0$ ]
3.  $u \leftarrow 0$
4.  $z \leftarrow 1$
5. **While**  $u < y$  **do**
  - 5.1  $z \leftarrow z \cdot x$
  - 5.2  $u \leftarrow u + 1$
6. *Output*  $z$

Dette resulterer i at vi multipliserer  $x$  med seg selv  $y$  ganger, altså at vi beregner  $x^y$ .

# Beregnbare funksjoner

Dette sto vi igjen med da tiden var ute:

- I programmeringssammenheng er det ikke alltid så lett å vite når et gitt program med et gitt input faktisk gir oss et output i den mengden hvor vi vil ha det.
- I verste fall kan vi skrive programmer for funksjoner hvor det er umulig å bestemme hva definisjonsområdet er.
- Innenfor IT er det derfor naturlig også å studere **partielle funksjoner** fra en mengde  $X$  til en mengde  $Y$ .
- Dette vil være funksjoner hvor definisjonsområdet er en delmengde av  $X$  og hvor verdiområdet er  $Y$ .
- Tolkningen av et program som en funksjon fra et Cartesisk produkt av datatyper til en datatype vil vanligvis være som en partiel funksjon.