Inherent problems with OO

Magne Haveraaen
Universitet i Bergen

Inf329 – Selected topics in programming theory

2006-05-02

Programvaretenking

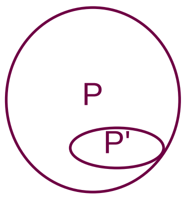
Uformell ressonnering om programvare er grunnleggende
- når vi skriver program
- når vi skal forstå andres program
- når vi skal lete etter feil
- når vi skal forbedre programmene

Det er mye innsikt om ressonnering å hente fra formelle metoder

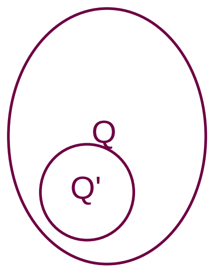
Litt logikk

{ P } m(…) { Q }

Kovarians



{ P' } m(…) { Q' }
der P' => P og Q' => Q



bevaring av invarianter

{ P & inv } m(…) { Q & inv }

Litt logikk: eksempler

real a

{ True } a := abs(a) { True }

{ True } a := abs(a) { $0 \leq a$ }

{ a er et heltall } a := abs(a) { a er et heltall }

{ a er et heltall } a := abs(a) { a er et naturlig tall }

{ a er et naturlig tall } a := abs(a) { a er et naturlig tall }

{ False } a := abs(a) { $a < 0$ }

Objektorientert programmering

- **dataabstraksjon**
  klasser
  klasser med klasser som parametre
  klasser med parametre (avhengige datatyper)
- **arv**
  spesifikasjonsarv
  delmengderelasjoner
  subtyping
  implementasjonsarv
- **objektidentitet**
  pekere

Dataabstraksjon / klasser

Sett utenfra:

- atomiske typer
- atomiske metoder som bearbeider verdier av tilhørende typer

Sett innenfra:

- Datastruktur (attributter)
  Datainvariant: skal alltid være oppfylt når objektet *er i ro*
  Abstraksjonsfunksjon / ekvivalensrelasjon:
  forteller når to konkrete verdier er samme abstrakte verdi
- Metoder
  - virker på datastrukturen
  - må bevare datainvariant og ekvivalensrelasjon

Dataabstraksjon: rasjonale tall 1

Klasse rasj1
// datadel
    Int t, n;
    DI(**obs** rasj a) { return r.n ≠ 0; }
    ==(**obs** rasj a, **obs** b) { return a.t * b.n == b.t * a.n; }
// metodedel, skriver m(a,b) fremfor a.m(b)
    rasj() : t(0), n(1) {}; // konstruktør av verdien 0
    += (**upd** rasj a, **obs** b){ a.t := a.t*b.n + b.t*a.n; a.n *= b.n; }
    *= (**upd** rasj a, **obs** b){ a.t *= b.t; a.n *= b.n; }
    abs (**upd** rasj a){ **if** a.t*a.n < 0 **then** a.t := -a.t **fi**; }
// algoritmene bevarer DI og ==

Dataabstraksjon: rasjonale tall 2

Klasse rasj2
// datadel
    Int t, n;
    DI(**obs** rasj a) { return r.n > 0; }
    ==(**obs** rasj a, **obs** b) { return a.t * b.n == b.t * a.n; }
// metodedel, skriver m(a,b) fremfor a.m(b)
    rasj() : t(0), n(1) {}; // konstruktør av verdien 0
    += (**upd** rasj a, **obs** b){ a.t := a.t*b.n + b.t*a.n; a.n *= b.n; }
    *= (**upd** rasj a, **obs** b){ a.t *= b.t; a.n *= b.n; }
    abs (**upd** rasj a){ **if** a.t < 0 **then** a.t := -a.t; **fi**; }
// rasj1-algoritmene bevarer her den strengere DI og ==

Dataabstraksjon: rasjonale tall 3

```
      red (upd int x, upd int y){ int d := gcd(x,y); x /= d; y /= d; }
Klasse rasj3
// datadel
      Int t, n;
      DI(obs rasj a) { int x:=a.t; int y:=a.t; red(x,y);
         return r.n > 0 && x==a.t && y==a.n; }
      ==(obs rasj a, obs b) { return a.t==b.t && a.n==b.n; }
// metodedel, skriver m(a,b) fremfor a.m(b)
      rasj() : t(0), n(1) {}; // konstruktør av verdien 0
      +=(upd rasj a, obs b){ a.t:=a.t*b.n+b.t*a.n; a.n*= b.n; red(a.t,a.n); }
      *=(upd rasj a, obs b){ a.t *= b.t; a.n *= b.n; red(a.t,a.n); }
      abs(upd rasj a){ if a.t < 0 then a.t := -a.t; fi; }
// kun abs bevarer her den strengere DI og ==
```

Dataabstraksjon: skjermpunkter / vektorer

```
Klasse pkt
// datadel
      Int x, y;
      DI(obs pkt a) { return true; }
      ==(obs pkt a, obs b) { return a.x==b.x && a.y==b.y; }
// metodedel
      pkt() : x(0), y(0) {}; // konstruktør for origo
      += (upd pkt a, obs b){ a.x += b.x; a.y += b.y; }
      *= (upd pkt a, obs b){ a.x *= b.x; a.y *= b.y; }
      real abs (obs pkt a){ return sqrt(a.x*a.x + a.y*a.y); }
```

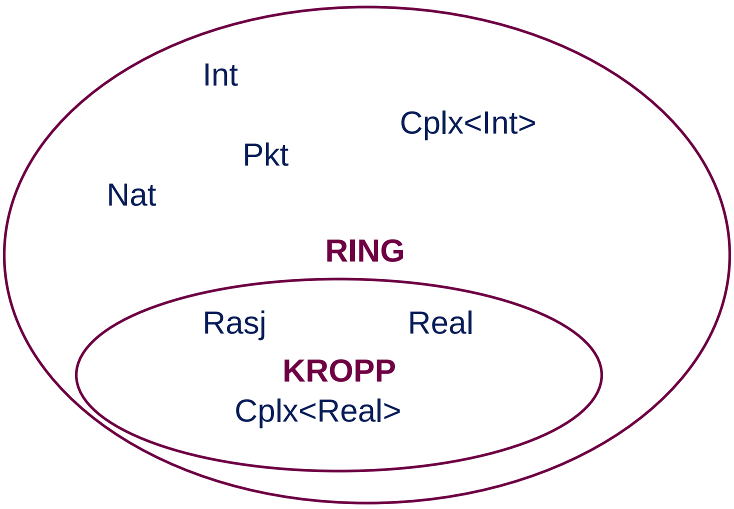
Dataabstraksjon: komplekse tall

```
Klasse cplx<num>
// datadel
    num r, i;
    DI(obs cplx<num> a) { return true; }
    ==(obs cplx<num> a, obs b) { return a.r==b.r && a.i==b.i; }
// metodedel
    cplx<num>() : r(num()), i(num()) {}; // konstruktør for origo
    += (upd cplx<num> a, obs b){ a.r += b.r; a.i += b.i; }
    *= (upd cplx<num> a, obs b)
        { num x := a.r*b.i+a.i*b.r; a.r := a.r*b.r-a.i*b.i; a.i := x; }
    real abs (obs cplx<num> a){ return sqrt(a.r*a.r + a.i*a.i); }
```

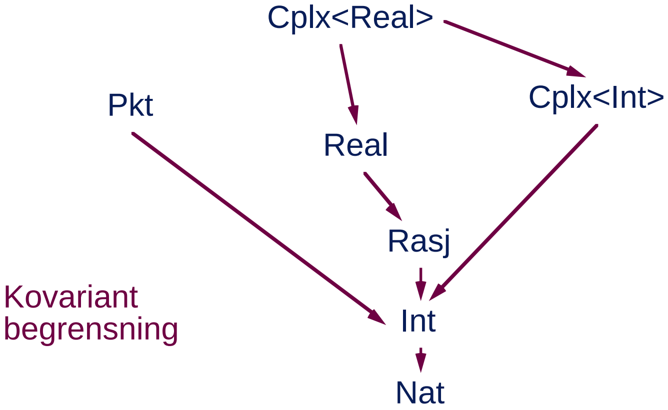
Dataabstraksjoner

	rasj1	rasj2	rasj3	pkt	cplx
DS	Int, Int	…	…	…	…
DI	$n \neq 0$	$n > 0$	n>0, forkortet	true	…
==	forholdstall	…	komponentvis	…	…
cons	0/1	…	…	0,0	…
+=	fellesnevner	…	f.n. forkortet	komp.vis	…
*=	komponentvis	…	k.vis forkortet	komp.vis	cplx
abs	abs(t*n)	abs(t)	…	diagonal	…

Arv: spesifikasjoner



Arv: delmengderelasjoner



Arv: subtyping

T' ≤ T ≤ T''
S' ≤ S ≤ S''

f : T -> S
f' : T'' -> S' // kontravariant subtype
g : S -> V

```
T x;
T' x';
    V v1 := g ( f ( x ) ) ;
    V v2 := g ( f ( x' ) ) ;      // smalere argument
    V v3 := g ( f' ( x ) ) ;      // kontravariant substitusjon
    V v4 := g ( f' ( x' ) ) ;
```

Arv: implementasjon

klasse C
- DSC c;
- DIC(**obs** C c);
- EQC(**obs** C x, **obs** C y);
- f(**upd** C x); // inn: DIC(x), ut: DIC(x)
- h(**upd** C x); // inn: DIC(x), ut: DIC(x)

klasse D subklasse av C
- DSD d; // datastruktur: DSC c, DSD d;
- DID(**obs** D x); // hva med DIC(x.c)?
- EQD(**obs** D x, **obs** D y); // hva med EQC(x.c,y.c)?
- f(**upd** D x); // inn: DID(x), ut: DID(x)
- g(**upd** D x); // inn: DID(x), ut: DID(x)

Arv: implementasjonskrav

C x;
    { DIC(x) & PF } f(**upd** C x); { DIC(x) & QF }
    { DIC(x) & PH } h(**upd** C x); { DIC(x) & QH }
D y;
    { DID(y) & PF } f(**upd** D y); { DID(y) & QF }
    { DID(y) & PG } g(**upd** D y); { DID(y) & QG }
    { DID(y) & PH } h(**upd** C y); { DID(y) & QH }

Konsistens for D krever at

- DID(y) => DIC(y.c)
- alle arvede metoder bevarer DID (kovariant)
- virtuelle metoder sikrer bruk av subklassens metoder

Arv: binære operasjoner

C: k(**upd** C x, **upd** C y) // bevarer DIC(x)&DIC(y)

D: k(**upd** D x, **upd** D y); // bevarer DID(x)&DID(y)

C x1, x2;
    { DIC(x1) & DIC(x2) } k(C x1, C x2); { DIC(x1) & DIC(x2) }
D y1, y2;
    { DID(y1) & DID(y2) } k(D y1, D y2); { DID(y1) & DID(y2) }
    { DIC(x1) & DID(y2) } k(C x1, D y2); { DIC(x1) & DID(y2) }

Arv: binære operasjoner

C: k(**upd** C x, **upd** C y) // bevarer DIC(x)&DIC(y)

{ DSC tmp := x.c; x.c := y.c; y.c := tmp; }

D: k(**upd** D x, **upd** D y); // bevarer DID(x)&DID(y)

{ DSC tmpc := x.c; DSD tmpd := x.d ;

x.c := y.c; x.d := y.d; y.c := tmpc; y.d := tmpd; }

C x1, x2;

{ DIC(x1) & DIC(x2) } k(C x1, C x2); { DIC(x1) & DIC(x2) }

D y1, y2;

{ DID(y1) & DID(y2) } k(D y1, D y2); { DID(y1) & DID(y2) }

{ DIC(x1) & DID(y2) } k(C x1, D y2); { DIC(x1) & DID(y2) }

Kravet for konsistens blir at DSC og DSD må være uavhengige

- DID(D x) = DIC(x.c) & DID'(x.d)
- EQD(D x, D y) = EQC(x.c,y.c) & EQD'(x.d,y.d)

Arvehierarkier

	rasj1	rasj2	rasj3	pkt	cplx
DS	Int, Int	…	…	…	…
DI	n ≠ 0	n > 0	n>0, forkortet	true	…
==	forholdstall	…	komponentvis	…	…
cons	0/1	…	…	0,0	…
+=	fellesnevner	…	f.n. forkortet	komp.vis	…
*=	komponentvis	…	k.vis forkortet	komp.vis	cplx
abs	abs(t*n)	abs(t)	…	diagonal	…

Spesifikasjon **Ring** pkt, Int **Kropp** rasj1≈rasj2≈rasj3, cplx<Real>

Delmengde Int ≤ rasj1≈rasj2≈rasj3≤ Real ≤ cplx<Real>

Subtyping cplx<Real> ≤ Real ≤ rasj1≈rasj2≈rasj3 ≤ Int

Implementasjon ???

Objektidentitet: ombytting

```
bytt ( upd Ring x, upd Ring Y)
{   // x = x0, y = y0
      x += y;     // x = x0 + y0,                y = y0
      y = x - y;  // x = x0 + y0,                y = (x0 + y0) - y0 = x0
      x -= y;     // x = (x0 + y0) - x0 = y0,    y = x0
    // x = y0, y = x0
}
Int a=4, b=5;
      { a=4 & b=5 } bytt(a,b); { a=5, b=4 }
a=4;
      { a=4 & b=4 } bytt(a,b); { a=4, b=4 }
      { a=4 & b=4 } bytt(a,a); { a=0, b=4 }
```

Objektidentitet: lagerklasse

```
Klasse Elt
      N n;  // nøkkel
      D d;  // data
      nyn(upd Elt, obs N x);
      nyd(upd Elt, obs D x);

Klasse L
      Liste<Elt> el;
      DI(obs L x); // nøklene i x.el er unike
      legginn(upd L x, obs Elt e);
      sorter(upd L x);
      Elt hent(obs L x, obs N n);
```

Objektidentitet: lagerklasseproblem

```
N n1, n2; D d1, d2; Elt e1, e2);
    nyn(e1,n1); nyd(e1,d1);
    nyn(e2,n2); nyd(e2,d2);
L x;
    { DI(x) } legginn(x,e1); { DI(x) }  // identiteten til e1 i x
    { DI(x) } legginn(x,e2); { DI(x) }  // identiteten til e2 i x
    { DI(x) } sorter(x); { DI(x) & sortert(x) }
    { DI(x) & sortert(x) } bytt(e1,e2); { DI(x) & ikke sortert(x) }

    { DI(x) } sorter(x) || bytt(e1,e2); { DI(x) & sortert(x)?? }

    { DI(x) } nyn(e1,n2); { ikke DI(x) }
```

Når fungerer OO

Unngå problemområdene

- kun unære metoder
  - ekvivalensrelasjon kan ikke implementeres
- arv
  - datainvariant er uavhengig for arvet del og utvidet del

  og / eller
  - utelukkende virtuelle metoder
  - arvet metode må bevare subtypeinvariant

Eksempler

- vindusystem
- geometriske figurer som tilføres fargeatributt ved arv

Konklusjon

Objektorientering

- dataabstraksjon: den gode
  - datainvariant DI
  - ekvivalens ==

- arv: den forvirrende
  - spesifikasjonsarv: velfungerende
  - delmengde: kovariant
  - subtyping: kontravariant
  - implementasjonsarv: uklar gjenbruksmekanisme

- objektidentitet: den grusomme