



# Programmeringsmetodik DV1 Programkonstruktion 1

## Formell verifiering eller hur man bevisar att program gör rätt utan att testa dem

### Formell verifiering: Idé

- Skriv för- och eftervillkoren som matematiska uttryck (den *formella specifikationen*).
- Ge en matematisk tolkning av programmet (programmets *formella semantik*).
- Bevisa matematiskt att under antagandet att förvillkoret gäller (och att argumenten har typer som ges av MLs typkontroll) så är eftervillkoret lika med den matematiska tolkningen av programmet.

Ett sådant bevis garanterar att programmet alltid uppfyller eftervillkoret – dvs det gör alltid "rätt".

Förutsättningar: Den formella specifikationen är riktigt uttryckt, den formella semantiken är riktig, våra bevis är riktiga.....

### Programspråksemantik

Vi antar att det finns en (matematisk) funktion från program till matematiska objekt (t.ex. tal, mängder etc.). Funktionen skrivs  $[[...]]$ .

T.ex. gäller  $[[\text{size "abcd"}]] = 4$ .

Definitionen av  $[[...]]$  beskriver innebörden av programspråket. Jag använder en begränsad del av ML (t.ex. utan matchning) för att hålla nere storleken på semantiken.

Allt som skrivs mellan  $[[$  och  $]]$  är programkod. Allt som skrivs utanför  $[[...]]$  är matematik.

Exempel: I uttrycket  $[[3+5]]+7$  är plustecknet mellan  $[[...]]$  en ML-operator, medan plustecknet utanför är den matematiska operationen addition.

### Semantik för en del av ML

- $[[\text{if } B \text{ then } E1 \text{ else } E2]] = [[E1]]$  om  $[[B]] = \text{true}$
- $[[\text{if } B \text{ then } E1 \text{ else } E2]] = [[E2]]$  om  $[[B]] = \text{false}$
- $[[E1+E2]] = [[E1]]+[[E2]]$ , etc. för andra räkneoperationer
- $[[E1=E2]] = \text{true}$  om  $[[E1]]=[[E2]]$
- $[[E1=E2]] = \text{false}$  om  $[[E1]]\neq[[E2]]$
- $[[E1>E2]] = \text{true}$  om  $[[E1]]>[[E2]]$
- $[[E1>E2]] = \text{false}$  om  $[[E1]]\leq[[E2]]$

$[[F \ A]] = [[F]]([[A]])$

En funktionsdefinition  $\text{fun } F \ A = E$  utvidgar definitionen av  $[[...]]$  med ekvationen  $[[F]](A) = [[E]]$

(Här antas  $B, E1$ , etc. vara godtyckliga deluttryck.)

### En konvention

Om samma variabelnamn förekommer både innanför och utanför  $[[...]]$  så förutsätter vi att de betecknar samma värde – innanför  $[[...]]$  som ett ML-uttryck, utanför  $[[...]]$  som ett matematiskt objekt.

Exempel: Det gäller alltid att  $[[x]]=x$ , om  $x$  är ett tal.

Antag att  $x=3$ . I så fall gäller  $[[x+1]]=4$ .

Detta kan synas självklart men variabler i programkoden och matematiska variabler är helt olika saker. Det hela är matematiskt rätt intrikat och man kan egentligen inte göra det så lättvindigt som det verkar här. Skall man göra det riktigt så krävs en hel del matematisk apparatur som inte tillför något i den här föreläsningen.

Jag kommer alltså att förbigå problemet i tysthet...

### Exempel på användning av semantik

```
fun sign x =
  if x=0 then 0 else if x>0 then 1 else ~1
```

Detta ger ekvationen

$[[\text{sign}]](x) = [[\text{if } x=0 \text{ then } 0 \text{ else if } x>0 \text{ then } 1 \text{ else } \sim 1]]$

Vad ger semantiken för värde till programmet  $\text{sign}(2)+3$ ?

$[[\text{sign}(2)+3]] = [[\text{sign}(2)]]+[[3]] = [[\text{sign}]]([[2]])+3$   
 $= [[\text{if } 2=0 \text{ then } 0 \text{ else if } 2>0 \text{ then } 1 \text{ else } \sim 1]]+3$   
 $= [[\text{if } 2>0 \text{ then } 1 \text{ else } \sim 1]]+3$  (ty  $[[2>0]] = \text{true}$  om  $[[2]]>[[0]]$ )  
 $= [[\sim 1]]+3 = -1+3 = 2$

## Räkna ut potenser av heltal

```
(* exp(n,e)
  TYPE: int*int->int
  PRE: e≥0, n>0
  POST: ne
  EXAMPLE: exp(2,3) = 8 *)
(* VARIANT: e *)
fun exp(n,e) = if e = 0 then
  1
  else
  n * exp(n,e-1)
```

Funktionen beräknar potenser genom upprepad multiplikation.

Under antagande att funktionsdefinitionen finns och att

$n, e \in \mathbf{Z}$  (typvillkor) samt  $e \geq 0$  och  $n > 0$  (förvillkor)

skall bevisas att  $[[\text{exp}(n, e)]] = n^e$  (eftervillkor)

## Början på beviset

Funktionsdefinitionen ger ekvationen:

$$[[\text{exp}]](n, e) = [[\text{if } e=0 \text{ then } 1 \text{ else } n * \text{exp}(n, e-1)]]$$

Ifrån typvillkoret  $e \in \mathbf{Z}$  och förvillkoret  $e \geq 0$ , får vi  $e \in \mathbf{N}$ .

Välj ett godtyckligt  $n$  och bevisa  $[[\text{exp}]](n, e) = n^e$  med induktion över  $e$ .

Basfall: Antag  $e = 0$ .

$$\begin{aligned} [[\text{exp}]](n, e) &= [[\text{exp}]](n, 0) \\ &= [[\text{if } 0=0 \text{ then } 1 \text{ else } n * \text{exp}(n, 0-1)]] \\ &= 1 \quad (\text{eftersom } [[0=0]] = \text{true}) \\ &= n^0 \quad (\text{egenskap hos potenser då } n > 0 - \text{förvillkor}) \\ &= n^e \quad (\text{eftersom } e = 0) \end{aligned}$$

## Induktionsfallet

Funktionsdefinitionen ger ekvationen:

$$[[\text{exp}]](n, e) = [[\text{if } e=0 \text{ then } 1 \text{ else } n * \text{exp}(n, e-1)]]$$

Induktionsfall: För godtyckligt  $e \in \mathbf{N}$  antar vi  $[[\text{exp}]](n, e) = n^e$  och skall visa  $[[\text{exp}]](n, e+1) = n^{e+1}$ .

$$\begin{aligned} [[\text{exp}]](n, e+1) &= [[\text{if } e+1=0 \text{ then } 1 \text{ else } n * \text{exp}(n, e+1-1)]] \\ &= [[n * \text{exp}(n, e+1-1)]] \quad (\text{eftersom } [[e+1=0]] = \text{false om } e \geq 0) \\ &= [[n * \text{exp}(n, e)]] = n \cdot [[\text{exp}(n, e+1-1)]] \\ &= n \cdot [[\text{exp}]](n, e) \quad (\text{induktionshypotes}) \\ &= n \cdot n^e \quad (\text{induktionshypotes}) \\ &= n^{e+1} \end{aligned}$$

Alltså gäller  $[[\text{exp}]](n, e) = n^e$  för alla  $e \in \mathbf{N}$ . QED!

## Ett bättre program

```
(* exp(n,e)
  TYPE: int*int->int
  PRE: e≥0, n>0
  POST: ne
  EXAMPLE: exp(2,3) = 8 *)
(* VARIANT: e *)
fun exp(n,e) = if e = 0 then
  1
  else if e mod 2 = 0 then
  exp(n*n, e div 2)
  else
  n * exp(n*n, e div 2)
```

Hur fungerar denna funktion...? På vilket sätt är den bättre...?

Samma antaganden från typer och förvillkor som med förra exp, men bevis genom fullständig induktion.

## Bevis

Bevisa  $[[\text{exp}]](n, e) = n^e$  med fullständig induktion över  $e$ .

Induktionshypotes: ( $n$  omdöpt för att undvika namnkonflikter senare)

$$[[\text{exp}]](n', e') = n'^{e'} \quad \text{för alla } e' < e, n' > 0 \text{ och } n', e' \in \mathbf{N}.$$

Eftersom  $e \in \mathbf{Z}$  (typvillkor) och  $e \geq 0$  (förvillkor) vet vi att  $e \in \mathbf{N}$ .

Vi delar upp beviset i tre fall motsvarande de tre alternativen i  $\text{if}$ -uttrycken ger.

- Fall 1:  $e = 0$ .
- Fall 2a:  $e > 0$  och  $e$  jämn.
- Fall 2b:  $e > 0$  och  $e$  udda.

Eftersom  $e \in \mathbf{N}$  täcker dessa fall alla värden på  $e$ !

## Fall 1

Funktionsdefinitionen ger ekvationen:

$$[[\text{exp}]](n, e) = [[\text{if } e = 0 \text{ then } 1 \text{ else if } e \text{ mod } 2 = 0 \text{ then } \text{exp}(n*n, e \text{ div } 2) \text{ else } n * \text{exp}(n*n, e \text{ div } 2)]]$$

Fall 1: Antag  $e = 0$ .

$$\begin{aligned} [[\text{exp}]](n, e) &= [[\text{exp}]](n, 0) \\ &= [[\text{if } 0=0 \text{ then } 1 \text{ else } \dots]] \\ &= 1 \quad (\text{eftersom } [[0=0]] = \text{true}) \\ &= n^0 \quad (\text{egenskap hos potenser då } n > 0 - \text{förvillkor}) \\ &= n^e \quad (\text{eftersom } e = 0) \end{aligned}$$

Fall 1 klart!

## Fall 2a

Funktionsdefinitionen ger ekvationen:

$$[[\text{exp}]](n, e) = [[\text{if } e = 0 \text{ then } 1 \text{ else if } e \bmod 2 = 0 \text{ then } \text{exp}(n * n, e \text{ div } 2) \text{ else } n * \text{exp}(n * n, e \text{ div } 2)]]$$

Fall 2a. Antag  $e > 0$  och  $e$  jämn. Eftersom  $e$  jämn är  $[[e \bmod 2]] = 0$

$$\begin{aligned} & [[\text{exp}]](n, e) \\ &= [[\text{if } e = 0 \text{ then } 1 \text{ else } \dots]] \\ &= [[\text{if } e \bmod 2 = 0 \text{ then } \dots]] \quad (\text{ty } e > 0) \\ &= [[\text{exp}(n * n, e \text{ div } 2)]] \quad (\text{ty } [[e \bmod 2]] = 0) \\ &= [[\text{exp}]]([n * n], [[e \text{ div } 2]]) \\ &= [[\text{exp}]](n \cdot n, e/2) \quad (\text{eftersom } [[e \text{ div } 2]] = e/2 \text{ om } e \text{ jämn}) \\ &= \dots \quad (\text{fortsätter}) \end{aligned}$$

## Fall 2a (induktionshypotes)

Induktionshypotes:

$$[[\text{exp}]](n', e') = n'^{e'} \text{ för alla } e' < e, n' > 0 \text{ och } n', e' \in \mathbf{N}.$$

Eftersom  $e > 0$  är  $e/2 < e$ , eftersom  $e$  jämn gäller  $e/2 \in \mathbf{N}$ .

Eftersom  $n > 0$  är  $n \cdot n > 0$ , eftersom  $n \in \mathbf{N}$  (visat förut) gäller  $n \cdot n \in \mathbf{N}$ .

Alltså är induktionshypotesen tillämplig med  $e' = e/2$  och  $n' = n \cdot n$ .

$$\begin{aligned} \dots &= [[\text{exp}]](n \cdot n, e/2) \\ &= (n \cdot n)^{e/2} \quad (\text{induktionshypotesen!}) \\ &= n^e. \end{aligned}$$

Fall 2a klart!

## Fall 2b

Funktionsdefinitionen ger ekvationen:

$$[[\text{exp}]](n, e) = [[\text{if } e = 0 \text{ then } 1 \text{ else if } e \bmod 2 = 0 \text{ then } \text{exp}(n * n, e \text{ div } 2) \text{ else } n * \text{exp}(n * n, e \text{ div } 2)]]$$

Fall 2b. Antag  $e > 0$  och  $e$  udda.

Eftersom  $e$  udda är  $[[e \bmod 2]] = 1$

$$\begin{aligned} & [[\text{exp}]](n, e) \\ &= [[\text{if } e = 0 \text{ then } 1 \text{ else } \dots]] \\ &= [[\text{if } e \bmod 2 = 0 \text{ then } \dots]] \quad (\text{ty } e > 0) \\ &= [[n * \text{exp}(n * n, e \text{ div } 2)]] \quad (\text{ty } [[e \bmod 2]] = 1) \\ &= n * [[\text{exp}(n * n, e \text{ div } 2)]] = n * [[\text{exp}]]([n * n], [[e \text{ div } 2]]) \\ &= n * ([[ \text{exp} ]](n \cdot n, (e-1)/2)) \quad (\text{eftersom } [[e \text{ div } 2]] = (e-1)/2 \\ & \quad \quad \quad \text{om } e \text{ udda}) \\ &= \dots \quad (\text{fortsätter}) \end{aligned}$$

## Fall 2b (induktionshypotes)

Induktionshypotes:

$$[[\text{exp}]](n', e') = n'^{e'} \text{ för alla } e' < e, n' > 0 \text{ och } n', e' \in \mathbf{N}.$$

Eftersom  $e > 0$  är  $(e-1)/2 < e$ , eftersom  $e$  udda gäller  $(e-1)/2 \in \mathbf{N}$ .

Eftersom  $n > 0$  är  $n \cdot n > 0$ , eftersom  $n \in \mathbf{N}$  (visat förut) gäller  $n \cdot n \in \mathbf{N}$ .

Alltså är induktionshypotesen tillämplig med  $e' = (e-1)/2$  och  $n' = n \cdot n$ .

$$\begin{aligned} \dots &= n * ([[ \text{exp} ]](n \cdot n, (e-1)/2)) \\ &= n * (n \cdot n)^{(e-1)/2} \quad (\text{induktionshypotesen!}) \\ &= n \cdot n^{e-1} \\ &= n^e. \end{aligned}$$

Fall 2b och hela beviset klart. QED!

(QED = quod erat demonstrandum – vilket skulle bevisas)

## Andra typer av data

Program som behandlar andra typer av data (t.ex. tupler, strängar, listor) kan också verifieras om man på lämpligt sätt kan beskriva dem matematiskt. En sträng  $s$  kan t.ex. beskrivas som en funktion från naturliga tal mindre än strängens längd till teckenkoder.

Exempel. "Hej" kan beskrivas matematiskt som en funktion  $f$ , definierad som  $f(0) = 72$ ,  $f(1) = 101$ ,  $f(2) = 106$ , i övrigt odefinierad.

Bevis med sådana data uppfattas ofta som svåra eftersom vi inte är vana vid *räknereglerna* för sådana matematiska objekt

För vanliga tal tillämpar vi regler som  $n \cdot n^{e-1} = n^e$ ,  $x \cdot y = y \cdot x$  eller

$x \cdot (y+z) = x \cdot y + x \cdot z$  utan att tänka på det, men man måste veta t.ex. att

för strängar gäller  $[[\text{size}(x^y)]] = [[\text{size}(y^x)]]$

och  $[[\text{size}(x^y)]] = [[\text{size } x]] + [[\text{size } y]]$