Praktische Informatik 3

Einführung in die Funktionale Programmierung

Vorlesung vom 17.12.08: Verifikation und Beweis

Christoph Lüth

WS 08/09

Fahrplan

- Teil I: Grundlagen
- Teil II: Abstraktion
  - Abstrakte Datentypen
  - Signaturen & Axiome
  - Korrektheit von Programmen
  - Zustand und Aktionen
- Teil III: Beispiele, Anwendungen, Ausblicke

Inhalt

- Verifikation: Wann ist ein Programm korrekt?
- Beweis: Wie beweisen wir Korrektheit und andere Eigenschaften?
- Techniken:
  - Vollständige Induktion
  - Strukturelle Induktion
  - Fixpunktinduktion
- Beispiele

Warum beweisen?

- Test findet Fehler
- Beweis zeigt Korrektheit
- Formaler Beweis
  - Beweis nur durch Regeln der Logik
  - Maschinell überprüfbar (Theorembeweiser)
  - Hier: Aussagenlogik, Prädikatenlogik

Was beweisen?

- Prädikate:
  - Haskell-Ausdrücke vom Typ `Bool`
  - Allquantifizierte Aussagen:
    wenn $P(x)$ Prädikat, dann ist $\forall x.P(x)$ auch ein Prädikat
  - Sonderfall Gleichungen `s == t`

Wie beweisen?

- Gleichungsumformung (equational reasoning)
- Fallunterscheidungen
- Induktion
- Wichtig: formale Notation

Ein ganz einfaches Beispiel

```
addTwice :: Int-> Int-> Int
addTwice x y = 2*(x+ y)
```

zz: `addTwice x (y+z) == addTwice (x+y) z`

	`addTwice x (y+ z)`	
=	`2*(x+(y+z))`	Def. `addTwice`
=	`2*((x+y)+z)`	Assoziativiät von `+`
=	`addTwice (x+y) z`	Def. `addTwice`

Fallunterscheidung

```
max, min :: Int-> Int-> Int
max x y = if x < y then y else x
min x y = if x < y then x else y
```

zz: `max x y - min x y` $= |x - y|$

`max x y - min x y`

Cases:

1. $x < y$

$=$	`y - min x y`	Def. `max`
$=$	`y - x`	Def. `min`
$=$	$\lvert x - y \rvert$	Wenn $x < y$, dann $y - x = \lvert x - y \rvert$

2. $x \geq y$

$=$	`x - min x y`	Def. `max`
$=$	`x - y`	Def. `min`
$=$	$\lvert y - x \rvert$	Wenn $x \geq y$, dann $x - y = \lvert x - y \rvert$

Rekursive Definition, induktiver Beweis

- Definition ist rekursiv
  - Basisfall (leere Liste)
  - Rekursion (`x:xs`)

```
rev :: [a]-> [a]
rev []     = []
rev (x:xs) = rev xs++ [x]
```

  - Reduktion der Eingabe (vom größeren aufs kleinere)
- Beweis durch Induktion
  - Schluß vom kleineren aufs größere

Beweis durch vollständige Induktion

Zu zeigen:

Für alle natürlichen Zahlen x gilt $P(x)$.

Beweis:

- Induktionsbasis: $P(0)$

- Induktionsschritt:
  Induktionsvoraussetzung $P(x)$, zu zeigen $P(x+1)$.

Beweis durch strukturelle Induktion

Zu zeigen:

Für alle (endlichen) Listen xs gilt $P(xs)$

Beweis:

- Induktionsbasis: $P([])$

- Induktionsschritt:
  Induktionsvoraussetzung $P(xs)$, zu zeigen $P(x : xs)$

Induktion: ein einfaches Beispiel

zz: `map f (map g xs) == map (f. g) xs`

Ind: 1. Induktionssbasis

```
    map f (map g [])
=   map f []                          Def. map für []
=   []                                Def. map für []
=   map (f.g) []                      Def. map für []
```

2. Induktionsschritt

```
    map f (map g (x:xs))
=   map f (g x: map g xs)              Def. map für x:xs
=   f (g x): map f (map g xs)          Def. map für x:xs
=   f (g x): map (f. g) xs             Induktionsvoraussetzung
=   (f. g) x: map (f. ) xs             Def. .
=   map (f. g) (x:xs)                  Def. map für x:xs
```

Weitere Beispiele

```
length (filter p xs)  <=  length xs                    (1)

length (xs++ ys)  ==  length xs + length ys            (2)

map f (xs++ ys)  ==  map f xs ++ map f ys              (3)

sum (map length xs)  ==  length (concat xs)            (4)
```

Strukturelle Induktion über anderen Datentypen

Gegeben binäre Bäume:

```
data Tree a  = Null | Node (Tree a) a (Tree a)
```

Zu zeigen:

Für alle (endlichen) Bäume t gilt $P(t)$

Beweis:

- Induktionsbasis: $P(\texttt{Null})$
- Induktionsschritt:
  Voraussetzung $P(s)$, $P(t)$, zu zeigen $P(\texttt{Node}\ s\ a\ t)$.

Ein einfaches Beispiel

- Gegeben: `map` für Bäume:

```
fmap :: (a-> b)-> Tree a-> Tree b
fmap f Null = Null
fmap f (Node s a t) = Node (fmap f s) (f a) (fmap f t)
```

- Sowie Aufzählung der Knoten:

```
inorder :: Tree a-> [a]
inorder Null = []
inorder (Node s a t) = inorder s ++ [a] ++ inorder t
```

- Zu zeigen: `inorder (fmap f t) = map f (inorder t)`

Ein einfaches Beispiel

zz: `inorder (fmap f t) = map f (inorder t)`

Ind: 1. Induktionssbasis

```
    inorder (fmap f Null)
=   inorder Null
=   []
=   map f [])
=   map f (inorder Null)
```

2. Induktionsschritt

```
    inorder (fmap f (Node s a t))
=   inorder (Node (fmap f s) (f a) (fmap f t))
=   inorder (fmap f s) ++ [f a] ++ inorder (fmap f t)
=   map f (inorder s) ++ [f a] ++ map f (inorder t)
=   map f (inorder s) ++ map f [a] ++ map f (inorder t)
=   map f (inorder s ++ [a] ++ inorder t)
=   map f (inorder (T s a t))
```

Eine Einfache Beweistaktik

- Induktionssbasis: einfach ausrechnen
  - Ggf. für zweite freie Variable zweite Induktion nötig
- Induktionsschritt:
  1. Definition der angewendeten Funktionen links nach rechts anwenden (auffalten)
  2. Ausdruck so umformen, dass Induktionssvoraussetzung anwendbar
  3. Definition der angewendeten Funktionen rechts nach links anwenden (einfalten)
- Schematisch: $P(\texttt{x:xs}) \rightsquigarrow E\ \texttt{x}\ (P\ \texttt{xs}) \rightsquigarrow E\ \texttt{x}\ (Q\ \texttt{xs}) \rightsquigarrow Q(\texttt{x:xs})$

Fallbeispiel: Der Speicher

- Zur Erinnerung:

```
data Store a b = Store [(a, b)]

empty :: Store a b
empty = Store []
```

Der Speicher

- Hilfsfunktionen auf Dateiebene liften
- Lesen:

```
get :: Eq a=> Store a b-> a-> Maybe b
get (Store s) a = get' s a

get' :: Eq a=> [(a, b)]-> a-> Maybe b
get' [] a          = Nothing
get' ((b, v):s) a =
    if a == b then Just v else get' s a
```

- Schreiben:

```
upd :: Eq a=> Store a b-> a-> b-> Store a b
upd (Store s) a v = Store (upd' s a v) where

upd' :: Eq a=> [(a, b)]-> a-> b-> [(a, b)]
upd' [] a v     = [(a, v)]
upd' ((b, w):s) a v =
    if a == b then (a, v):s else (b,w): upd' s a v
```

Zusammenfassung

- Formaler Beweis vs. Testen:
  - Testen: einfach (automatisch), findet Fehler
  - Beweis: mühsam (nicht automatisierbar), zeigt Korrektheit
- Formaler Beweis hier:
  - Aussagenlogik, einfache Prädikate
- Beweismittel:
  - Gleichungen (Funktionsdefinitionen)
  - Fallunterscheidung
  - Strukturelle Induktion (für alle algebraischen Datentypen)