

Лекция 7

Изследване на свойствата на програми на Haskell

Едно от най-големите предимства на програмирането във функционален стил е възможността строго да се доказват свойства на функционалните програми.

Тук ще покажем един прост подход за доказване на това, че определено множество от функции е дефинирано така, че дадено свойство, във формулировката на което участват тези функции, е вярно за всички (крайни) списъци.

Принцип на структурната индукция при работата със списъци

В случай, че трябва да се докаже, че дадено свойство $P(xs)$ е вярно за всички крайни списъци xs , доказателството може да се извърши на две стъпки:

- **Базов случай.** Доказване, че е вярно $P([])$.
- **Индуктивна стъпка.** Доказване, че е вярно $P(x:xs)$ при предположение, че е вярно $P(xs)$.

Твърдението $P(xs)$ във втората стъпка се нарича **индуктивна хипотеза**, тъй като за него се предполага, че е вярно в процеса на доказателството на $P(x:xs)$.

Пример 1. Нека `sum` и `doubleAll` са функции, дефинирани както следва:

```
sum :: [Int] -> Int
sum []      = 0
sum (x:xs)  = x + sum xs
```

```
doubleAll :: [Int] -> [Int]
doubleAll []      = []
doubleAll (z:zs)  = 2*z : doubleAll zs
```

Ще покажем, че за всеки списък `xs` е в сила
 $\text{sum} (\text{doubleAll } xs) = 2 * \text{sum } xs$.

Доказателството ще извършим на две стъпки:

1. Базов случай: трябва да покажем, че $\text{sum} (\text{doubleAll} []) = 2 * \text{sum} []$.
2. Индуктивна стъпка: ще покажем, че е в сила $\text{sum} (\text{doubleAll} (x:xs)) = 2 * \text{sum} (x:xs)$, използвайки за целта индуктивната хипотеза $\text{sum} (\text{doubleAll} xs) = 2 * \text{sum} xs$.

Базов случай

$\text{sum} (\text{doubleAll} []) = \text{sum} [] = 0;$
 $2 * \text{sum} [] = 2 * 0 = 0.$

Следователно, $\text{sum} (\text{doubleAll} []) = 2 * \text{sum} []$.

Индуктивна стъпка

$$\begin{aligned} \text{sum} (\text{doubleAll} (x:xs)) &= \text{sum} (2*x : \text{doubleAll} xs) \\ &= 2*x + \text{sum} (\text{doubleAll} xs); \\ 2 * \text{sum} (x:xs) &= 2 * (x + \text{sum} xs) = 2*x + 2*\text{sum} xs. \end{aligned}$$

Според индуктивното предположение е в сила $\text{sum} (\text{doubleAll} xs) = 2*\text{sum} xs$, следователно

$$\text{sum} (\text{doubleAll} (x:xs)) = 2*\text{sum} (x:xs) .$$

Пример 2. Връзка между length и ++.

Нека length и ++ са функциите за намиране на дължината на списък и конкатенация на списъци, дефинирани както следва:

```
length :: [a] -> Int
length []      = 0
length (z:zs) = 1 + length zs
```

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

Ще покажем, че за всеки два списъка xs и ys е в сила $\text{length } (xs ++ ys) = \text{length } xs + \text{length } ys$.

Доказателство

Базов случай

Ще покажем, че $\text{length} ([] ++ ys) = \text{length} [] + \text{length} ys$.

$\text{length} ([] ++ ys) = \text{length} ys$;

$\text{length} [] + \text{length} ys = 0 + \text{length} ys = \text{length} ys$.

Следователно $\text{length} ([] ++ ys) = \text{length} [] + \text{length} ys$.

Индуктивна стъпка

Ще покажем, че е вярно

$\text{length} ((x:xs) ++ ys) = \text{length} (x:xs) + \text{length} ys$

при условие, че съгласно индуктивното предположение е вярно

$\text{length} (xs ++ ys) = \text{length} xs + \text{length} ys$.

$$\begin{aligned} \text{length } ((x:xs) ++ ys) &= \text{length } (x:(xs ++ ys)) \\ &= 1 + \text{length } (xs ++ ys) = 1 + \text{length } xs + \text{length } ys; \end{aligned}$$

$$\text{length } (x:xs) + \text{length } ys = 1 + \text{length } xs + \text{length } ys.$$

Следовательно

$$\text{length } ((x:xs) ++ ys) = \text{length } (x:xs) + \text{length } ys .$$

Пример 3. Връзка между reverse и ++.

Нека reverse е функцията, която обръща реда на елементите на даден списък, дефинирана по следния начин:

```
reverse :: [a] -> [a]
reverse []      = []
reverse (z:zs) = reverse zs ++ [z]
```

Ще покажем, че

$\text{reverse } (xs ++ ys) = \text{reverse } ys ++ \text{reverse } xs$.

Доказателство

Базов случай

$\text{reverse} ([] ++ ys) = \text{reverse } ys;$

$\text{reverse } ys ++ \text{reverse } [] = \text{reverse } ys ++ [] = \text{reverse } ys.$

Следователно

$\text{reverse} ([] ++ ys) = \text{reverse } ys ++ \text{reverse } [].$

Забележка. Горното доказателство ще е коректно, ако докажем, че добавянето чрез конкатенация ($++$) на празен списък към даден друг списък е операция – идентитет, т.е.

$xs ++ [] = xs$ за всеки списък xs .

Доказателството на последното твърдение може лесно да бъде извършено по индукция (използвайки принципа на структурната индукция).

Индуктивна стъпка

$\text{reverse} ((x:xs) ++ ys) = \text{reverse} (x:(xs ++ ys))$
 $= \text{reverse} (xs ++ ys) ++ [x].$

Според индуктивното предположение е вярно
 $\text{reverse} (xs ++ ys) = \text{reverse} ys ++ \text{reverse} xs.$

Следователно

$\text{reverse} ((x:xs) ++ ys) = (\text{reverse} ys ++ \text{reverse} xs) ++ [x].$

От друга страна

$\text{reverse} ys ++ \text{reverse} (x:xs) = \text{reverse} ys ++ (\text{reverse} xs ++ [x]).$

Тъй като операцията конкатенация на списъци ($++$) е асоциативна, т.е. за всеки три списъка xs , ys и zs е изпълнено $xs ++ (ys ++ zs) = (xs ++ ys) ++ zs$, то следователно
 $\text{reverse} ((x:xs) ++ ys) = \text{reverse} ys ++ \text{reverse} (x:xs).$

Забележки

1. Асоциативността на операцията '++' подлежи на строго доказателство (такова може лесно да се направи отново с използване на принципа на структурната индукция).
2. Принципът на структурната индукция е подходящ за доказване на свойства на функции, дефинирани с помощта на примитивна рекурсия.