For-comprehensions

Jacek Wasilewski

For-comprehensions (1)

- Higher-order functions provide powerful constructions for lists
  - it might get messy if multiple operations are performed
- For-comprehensions – syntactic sugar that makes composition of multiple operations simpler
- For-comprehensions are good for multiple operations with
  - foreach
  - map
  - flatMap
  - filter
- For-expressions are translated into calls to those methods.

For-comprehensions (2)

- General syntax:
  `for ( s ) yield e`
- Like for-loop in imperative languages
- Constructs a list of the results of all iterations
- Example:

```
for (p <- persons if p.age > 20) yield p.name

for {p <- persons
     if p.age > 20} yield p.name
```

For-comprehensions (3)

```
scala> val l = List.range(1, 10)
l: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9)

scala> l.filter(x => x % 2 == 0).map(x => x * x)
res46: List[Int] = List(4, 16, 36, 64)

scala> for (x <- l if x % 2 == 0) yield x * x
res47: List[Int] = List(4, 16, 36, 64)
```

For-comprehensions (4)

- General syntax
  `for ( s ) yield e`
- In the place of **`s`** we can put
  - generators (always start with a generator)
  - definitions
  - filters
- **`yield`** forms a new list
  - can be replaced with any command, e.g. **`println`**

For-comprehensions (4)

- A generator is of form
  `x <- list`
- Takes the list `list` and binds `x` to successive values in the list
  - introduces the variable `x` that can be used later
- Example:

```
scala> val l = List.range(1, 10)
l: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9)

scala> for (x <- l) yield x
res48: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

For-comprehensions (5)

- Filters
  `if some_condition_that_returns_boolean`
  - All elements are checked
  - condition is `false` – element is omitted
- Definitions
  `x = some_expression`

Example (1)

```
scala> for { i <- List.range(1, 6)
     |       j <- List.range(1, 6)
     |       s = i + j
     |       if s % 2 == 0 }
     | println (i, j)
```

```
(1,1)
(1,3)
(1,5)
(2,2)
(2,4)
(3,1)
(3,3)
(3,5)
(4,2)
(4,4)
(5,1)
(5,3)
(5,5)
```

Example (2)

```
case class Book(title: String, authors: List[String])

val books: List[Book] = List( Book("Structure and Interpretation of
Computer Programs", List("Abelson, Harold", "Sussman, Gerald J.")), ...)


for (b <- books; a <- b.authors if a.startsWith("Ullman")) yield b.title
List(Principles of Compiler Design)


for (b <- books if (b.title.indexOf("Program")) >= 0) yield b.title
List(Structure and Interpretation of Computer Programs, Programming in
Modula-2, Introduction to Functional Programming)
```