Meeting 20: Mutation

Announcements

Lab 5
March-April 2014

<table>
<thead>
<tr>
<th>Su</th>
<th>Mo</th>
<th>Tu</th>
<th>We</th>
<th>Th</th>
<th>Fr</th>
<th>Sa</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>23</td>
<td>24</td>
<td>25: Spring Break</td>
<td>26</td>
<td>27: Spring Break</td>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>30</td>
<td>31</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Out, due April 6

The "Error Code" Problem

We, programmers, often need to write functions that return a result or an indication of error. In other words, we need to to express "missing results".

- Java: null is in any type
- Python: None is in any type
- Scala: Option[T] a type that is either None or Some(t) for t : T
- ML: 'a option a type that is either None or Some(a) for a : 'a

Have to check for null or None (or risk run-time error).

An Example
val parameter1: Option[String] = {
  getParameter() match {
    case None => None
    case Some(s) =>
      val sp = s.trim
      if (sp.length == 0) None
      else Some(sp.toUpperCase)
  }
}

val parameter2: Option[String] = {
  getParameter() map {
    s => s.trim
  } filter {
    s => s.length != 0
  } map {
    s => s.toUpperCase
  }
}

Scala Option

sealed abstract class Option[+A] {
  def isEmpty: Boolean
  def get: A

  def map[B](f: A => B): Option[B] =
    if (isEmpty) None else Some(f(this.get))

  def flatMap[B](f: A => Option[B]): Option[B] =
    if (isEmpty) None else f(this.get)
}

Some

final case class Some[+A](x: A) extends Option[A] {
  def isEmpty = false
  def get = x
}

None
case object None extends Option[Nothing] {
  def isEmpty = true
  def get = throw new NoSuchElementException("None.get")
}

A for expression

Example from OSV
Find all authors who have published at least two books

```scala
for (b1 <- books;
     b2 <- books if b1 != b2;
     a1 <- b1.authors;
     a2 <- b2.authors if a1 == a2)
  yield a1
```

Translated
A bit dense, but this is the actual computation on the underlying collection:

```scala
books flatMap
  { b1 => books withFilter { b2 => b1 != b2 } flatMap
    { b2 => b1.authors flatMap
      { a1 => b2.authors withFilter { a2 => a1 == a2 } map
        { a2 => a1 }
      }
    }
  }
```

Reminiscent of queries (SQL).

Application: Twitter Scalding

- Write queries in high-level Scala, translated to underlying Hadoop MapReduce
- Programming "big data" analytics
- Really at scale
- https://github.com/twitter/scalding
Questions
- pass-by-name
- judgement form for step (Fig 7)