Domain Specific Languages (DSLs)

Part 2: Parser Combinators

Walter Cazzola

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
e-mail: cazzola@dico.unimi.it

Payroll DSL: A First Parser Combinator Version.

Paycheck for employee "Buck Trends" is salary for 2 weeks minus deductions for:
- federal income tax is 15% of gross,
- state income tax is 5% of gross,
- insurance premiums are $500 in gross currency,
- retirement fund contributions are 10% of gross.

Paycheck = employeeName ~ salary ~ for ~ duration

duration = decimalNumber ~ weeksDays

weeksDays = week | weeks | day | days

amount = decimalNumber ~ in ~ gross ~ currency

deductAmount = percentage ~ amount

percentage = toBe ~ decimalNumber ~ in ~ gross ~ currency

decimalNumber = ...

doubleNumber = ...

References

DSLs

Walter Cazzola

Domain Specific Languages (DSLs)

Case Study: the DSL Grammar:

Paycheck = employeeName ~ salary ~ for ~ duration

deductions = for ~ { ~ deductItems ~ }

deductItems = deductItem ~ (repsep(deductItem, " "))

deductItem = tax | insurance | retirement

tax = state ~ tax

state = federal | state

insurance = insurance ~ premiums

retirement = retirement ~ fund ~ contributions

fund = percent ~ amount

amount = toBe ~ decimalNumber ~ in ~ gross ~ currency

decimalNumber = ...

doubleNumber = ...

termination | terminals | alternatives | reductions | repetitions

DSLs

Walter Cazzola

Domain Specific Languages (DSLs)

Payroll DSL: A First Parser Combinator Version.

package payroll.pcdsl
import scala.util.parsing.combinator._
import payroll.
import payroll.Type2Money._

class PayrollParserCombinatorsV1 extends JavaTokenParsers {

def paycheck ~= empl ~ gross ~ deduct

def empl ~> "employee" ~> employeeName

def gross ~> "salary" ~> duration

def deduct ~> "deductions" ~> (repsep(deductItem, ","))

def deductItem ~> tax | insurance | retirement

def deductKind =~ "fund" =~ fundContributions

def fundContributions =~ "percent" =~ amount

def amount =~ toBe =~ decimalNumber ~ in ~ gross ~ currency

def toBe =~ "is" | "are"

def doubleNumber =~ floatingPointNumber // floatingPointNumber from JavaTokenParsers
}
Sequential Composition
- ~ is used when the results produced by the productions on the left and right of the ~ should be retained for further processing
  
  ```scala
def paycheck = empl ~ gross ~ deduct
```
- --> is used when the result for the productions to the left is no longer needed
  
  ```scala
def empl = "paycheck" ~> "for" ~> "employee" ~> employeeName
```
- <> is used when the result for the productions to the right is no longer needed
  
  ```scala
def tax = fedState <= "income" <= "tax"
```

Alternative Composition
- | expresses when two parsers are in alternative
  
  ```scala
def weekdays = "weeks" | "week" | "days" | "day"
```

Repetitive Composition
- rep/repsep match zero or more repetitions
  
  ```scala
def deduct = "minus" ~> "deductions" ~> "for" ~> "{" ~> repsep(deductItem,"\)") <= \"
```

There is an opt method for optional terms not used.

To use the defined parser

```scala
val p = new PayrollParserCombinatorsV1
p.parseAll(p.paycheck, input)
```

parseAll is defined in a parent class it receives a parser (an invocation to paycheck in our case) and the input string to parse,

- if the parsing process is successful the result is an instance of p.Success[+T] a case class declared in the Parsers trait,
- the prefixed indicates that p.Success is a path-dependent type and permits to distinguish the result from two different parsers,
- the Success instance has two fields, the first is the result of the parse (of type T), the second is the remaining input to parse (normally empty);
- if the parse fails, the return instance is either a p.Failure or p.Error; both are derived from p.NoSuccess and contains fields for an error message and the unconsumed input at the point of failure.

As we parse the DSL
- we had to look up the employee by name
- fetch his gross salary for the specified period and calculate the deductions

Once the parser finishes
- we need to return a pair with the Employee instance and the completed Paycheck
Domain Specific Languages (DSLs)
Giving a Semantics to the DSL

Notes on the DSL
- The parser uses a map (Name) of known employees for simplicity.
- currentEmployee and grossAmount store the employee the parser is processing and his gross salary for the pay periods.
- This parser version is an evolution of the previous one which takes into consideration what should be the final result, e.g.,

```scala
payroll.pcdsl._
import payroll.Type2Money._
import payroll.pcdsl._
```

will return a Pair with the Employee and the computed Paycheck

```scala
def deduct = "minus" => "deductions" => "for" => "(" => deductItems <= ")"
```

- weeks and days ignore the parsed string, they just return a multiplication factor used to determine the total days in the duration production rule

Domain Specific Languages (DSLs)
Giving a Semantics to the DSL (Cont’d)

```scala
package payroll.pcdsl
import java.util.parsing.combinator...
import payroll.Type2Money...

class PayrollParserCombinators{val employees: Map[Name,Employee]} extends JavaTokenParsers{
  var currentEmployee: Employee = null
  var grossAmount: Money = Money(0)

  /** @return Parser[(Employee, Paycheck)] */
  def paycheck = emt = gross = deduct ^ { case e-g-d => e, Paycheck(g, d, d) }

  /** @return Parser[(Employee, Paycheck)] */
  def emt = "paycheck" => "for" => "employee" => employeeName ^^ { name =>
    val name = name.substring(1, name.length-1).split(" ")
    if (!employees.contains(n)) throw new UnknownEmployee(name)
    currentEmployee = employees(n); currentEmployee
  }

  /** @return Parser[String] */
  def duration = decimalNumber =~ weeksDays ^^ {
    case n = factor => n.toInt * factor
  }

  /** @return Parser[String] */
  def days = "days?" r ^^ { n =>
    case n = "days" => n
  }

  /** @return Parser[Money] */
  def gross = "is" => "salary" => "for" => duration ^^ {
    dur = grossAmount = salaryForDays(dur); grossAmount
  }

  /** @return Parser[String] */
  def deduct = "minus" => "deductions" => "for" => "(" => deductItems <= ")"

  /** @return Parser[String] */
  def salaryForDays(days: Int) = (currentEmployee.annualGrossSalary / 260.0) * days

  object PayrollBuilder {
    def main(args: Array[String]) = {
      val buck = Employee(Name("Buck", "Trends"), Money(80000))
      val jane = Employee(Name("Jane", "Buck"), Money(90000))
      val employees = Map(buck.name -> buck, jane.name -> jane)
      val p = new PayrollParserCombinators(employees)
      args.foreach { filename =>
        val src = scala.io.Source.fromFile(filename)
        val lines = src.nextLine
        p.parseAll(p, lines) match {
          case p.Success(Pair(employee, paycheck),_)
            print("%s %s: %s
", employee.name.first, employee.name.last, paycheck)
        }
        src.close()
    }
  }
```
Considering the following

- 2. correct programs in the new DSL

  paycheck for employee "Jane Doe"
  is salary for 2 weeks minus deductions for {}

  paycheck for employee "Buck Trends"
  is salary for 2 weeks minus deductions for {
    federal income tax is 25. percent of gross,
    state income tax is 5. percent of gross,
    insurance premiums are 500. in gross currency,
    retirement fund contributions are 10. percent of gross
  }

- and the wrong (inexistent employee) program

  paycheck for employee "John Doe"
  is salary for 2 weeks minus deductions for {}

They behave as follows

```scala
[16:28]cazzola@surfer:/lp/scala>scala PayRollBuilder test1.pr test2.pr test3.pr
Jane Doe: Paycheck($3461.54, $3461.54, $0.00)
Buck Trends: Paycheck($3076.92, $1346.15, $1730.77)
payroll.pcdsl.UnknownEmployee: Name(John, Doe)
at payroll.pcdsl.PayrollParserCombinators$$anonfun$empl$4.apply(payroll-pc.scala:24)
```

References

- Martin Odersky and Matthias Zenger.
  Scalable Component Abstractions.
- Nathanael Schäfl, Stéphane Ducasse, Oscar Nierstrasz, and Andrew P. Black.
  Traits: Composable Units of Behaviour.
- Venkat Subramaniam.
  Programming Scala.
  The Pragmatic Bookshelf, June 2009.
- Dean Wampler and Alex Payne.
  Programming Scala.
  O'Reilly, September 2009.