SIP Proposal

Boolean pattern matching expressions

Johannes Rudolph (johannes.rudolph@gmail.com)
Mathias Doenitz (mathias@spray.io)

31 October 2012 Initial version

Introduction

Pattern matching is a versatile syntactic element in Scala used to match and deconstruct a compound value and (optionally) extract its elements.
One common use case is to check if some value matches a given pattern and to provide the result of the check as a boolean value. This can be expressed straightforwardly by a pattern matching expression consisting of one case clause containing the “subject” pattern evaluating to `true` and a fall-back case clause matching everything else evaluating to `false`. For example:

```
val x: List[Option[String]] = // ...

val matches =
  x match {
    case List(Some("test")) => true
    case _ => false
  }
```

Even though completely correct this construct is syntactically inefficient because the main information, the pattern to test against, is buried in a lot of syntactic overhead.

We think that the relative commonness of such a “match/doesn’t match” check (e.g. as a filter predicate) warrants a small syntax extension offering an alternative for expressing the above in a more succinct way.

Proposed syntax

Allow

```
x match pattern [if guard]
```

as a short form for

```
x match {
  case pattern [if guard] => true
  case _ => false
}
```

One especially beneficial effect of this extension would be that it allowed one to define a `T => Boolean` (predicate) function literal via a `_ match pattern` expression, which is similarly efficient on the syntax level as "Pattern Matching Anonymous Functions" (SLS 8.5).
Creating even shorter forms of anonymous “match/doesn’t match” function literals may be difficult to achieve because of the similarity between expressions and patterns and the resulting possible ambiguity in the parser, which is currently solved by allowing patterns only after certain keywords in certain positions.

Specification changes

The section about pattern matching expression syntax (8.4) needs to be amended and the proposed transformation be explained.

Add the new rule for the boolean pattern match:

Expr ::= PostfixExpr ‘match’ ‘{’ CaseClauses ‘}’ | PostfixExpr ‘match’ Pattern [Guard]

Add explanation:

A pattern matching expression of the second form

PostfixExpr ‘match’ Pattern [Guard]

is transformed to a pattern matching expression of the first form by creating two regular case clauses. The first case clause matches the given pattern and has the constant value `true`; the second case clause is a wildcard pattern matching everything else and has the constant value `false`.

Implementation

A regular pattern matching expression is introduced by the keyword `match` followed by several case clauses embraced in curly brackets. A boolean pattern matching expression is introduced by `match` followed by the single pattern directly. The parser can transform a boolean pattern matching expression directly into the regular form so that no new syntax trees are necessary to represent the new form.

See the implementation proposal for a concrete implementation.

Interactions with other language features

- Infix expressions: A boolean pattern matching expression must be enclosed in parentheses to be used in infix expressions.

Possible disadvantages

- People might start using if/else chains with boolean matches instead of using regular pattern matching with several case clauses

Alternatives

- Long form as given in the introduction
- Library functions

You can cut down a bit of the noise today with `PartialFunction.cond`:

```
scala> if (PartialFunction.cond(1) { case 2 => true }) {}
```

or with a new method `PartialFunction.matches`:

```
class Matches[T](x: T) {
  def matches(pf: PartialFunction[T, Any]) = pf isDefinedAt x
}

implicit def anyToMatches[T](x: T) = new Matches[T](x)

scala> val x: List[Option[String]] = List(None)

scala> x matches { case List(Some("Test")) =>  }
res0: Boolean = false

scala> x matches { case List(None) =>  }
res1: Boolean = true
```

Examples

```
val x: List[Option[String]] = //...
x match List(Some("Test"))
```

or

```
x.count(_ match Some("Test"))
```

or even

```
x.count(_ match Some("Test" | "test2"))
```

Boolean pattern matching expressions would also enable match-based assertions, as discussed on the mailing list.

References

Mailing list discussion: https://groups.google.com/d/topic/scala-debate/JdJx03POwyg/discussion
Stackoverflow discussion: http://stackoverflow.com/questions/4437061/scala-short-form-of-pattern-matching-that-returns-boolean
Implementation proposal:
Branch: https://github.com/jrudolph/scala/tree/boolean-match
Commit: https://github.com/jrudolph/scala/commit/b56717f18b494d983c74e52f9224b61f0b596069