Exhaustive check and detecting useless clause for Algebraic Data Type

Maranget, Luc. “Warnings for pattern matching.” Journal of Functional Programming 17.3 (2007): 387-421.

Rikito Taniguchi (21M30241)

Introduction

Algebraic data types and pattern matching are importatnt features of functional programming languages such as Haskell, OCaml, and Scala.

For example, in Scala, we can defined `Tree` data type as follows.

```scala
sealed abstract class Tree
case class Branch(t1: Tree, t2: Tree) extends Tree
case class Leaf(v: Int) extends Tree
```

If the pattern match against `Tree` is not exhaustive, we have a risk of crashing at runtime if the incoming value is not-considered pattern.

```scala
// it crashes at runtime if t = Branch(Branch(...), Leaf(...))
def patternMatch(t: Tree) = t match {
  case Branch(Leaf(_), Leaf(_)) => ???
  case Leaf(_) => ???
}
```

On top of that, it we write a important logic into a pattern case which is never reached, we have risk of significant bug.

```scala
def patternMatch(t: Tree) = t match {
  case Branch(_, _) => ???
  // the following pattern never reaches
  case Branch(Leaf(_), Leaf(_)) => someImportantMethod()
  case Leaf(_) => ???
}
```

This report summarize the exhaustiveness check algorithm in OCaml. Maranget, Luc. "Warnings for pattern matching."Journal of Functional Programming 17.3 (2007): 387-421.

Therefore, it's significant feature for abstract data types and pattern matching to be able to statically check

- All patterns are considered?
- Is there a redundant pattern case?

Fortunately, most of modern functional programming languages has this feature. For example, in Scala, if the patterns are not exhaustive, it warns:

```scala
def patternMatch(t: Tree) = t match {
  case Branch(Leaf(_), Leaf(_)) => ???
  case Leaf(_) => ???
}
// abst.scala:6: warning: match may not be exhaustive.
```

```
// It would fail on the following inputs: Branch(Branch(_, _), Branch(_, _)), Branch(Branch(_, _), Leaf(_)), Branch(Leaf(_), Branch(_, _))
//   def patternMatch(t: Tree) = t match {
                                  ^
```

and warns to unreachable clause.

```
def patternMatch(t: Tree) = t match {
  case Branch(_, _) => ???
  case Branch(Leaf(_), Leaf(_)) => ???
  case Leaf(_) => ???
}
// abst.scala:8: warning: unreachable code
//     case Branch(Leaf(_), Leaf(_)) => ???
//                                      ^
```

In this report, I'll survey how OCaml checks exhaustiveness and detects useless clause by reading Warnings for pattern matching.

(That paper is a prior work of A generic algorithm for checking exhaustivity of pattern matching (short paper) which generalize the algorithm so it can cover other rich language features such as GADT, this algorithm is employed by Scala3 and Swift).

Preparation

Value

まずは論文中で使われる用語などを定義していく。

```
value ::=
          c(v_1, ..., v_a) (a >= 0)
```

$$
\begin{array}{rcl}
v & ::= & \\
 & | & c(v_1, \ldots, v_a)
\end{array}
$$

`c` means constructor.

// 型は少なくとも 1 つの値を持つことを仮定する

Patterns

In the real program, programmers sometimes write something like the following program:

```
x match {
  case Branch(...) => ???
  case other => ???
}
```

In this program, the second case `case other => ???` catches any values and bind it to `other`. However, in the context of checking exhaustiveness of pattern matches, we don't need to care the variable name, and we can see the variable pattern as a wildcard pattern.

Definitions

Instance Relation

pattern			
_	$\preceq$	v	
$(p_1 \| p_2)$	$\preceq$	v	(iff $p_1 \preceq v \vee p_2 \preceq v$)
$c(p_1, ...p_a)$	$\preceq$	$c(v_1, ...v_a)$	(iff $(p_1, ...p_a) \preceq (v_1, ...v_a)$)
$(p_1, ...p_a)$	$\preceq$	$(v_1, ...v_a)$	(iff $p_i \preceq v_i, i \in [1..n]$)

Examples

- `_` $\preceq$ `Leaf(1)`
- `_` $\preceq$ `Branch(Leaf(1), Leaf(1))`
- `(Leaf(_), Leaf(_))` $\preceq$ `(Leaf(1), Leaf(1))`
- `Branch(_, _)` $\preceq$ `Branch(Leaf(1), Leaf(1))`
- `Branch(_, _)| Leaf(_)` $\preceq$ `Branch(Leaf(1), Leaf(1))`

Pattern vector and pattern matrix

```
(x, y) match {
```

```
  case (Branch(_, _), Branch(_, _)) => ???
  case (Branch(_, _), Leaf(_)) => ???
  case (Leaf(_), Branch(_, _)) => ???
  case (Leaf(_), Leaf(_)) => ???
}
```

$$P = \begin{pmatrix} Branch(\_,\_) & Branch(\_,\_) \\ Branch(\_,\_) & Leaf(\_) \\ Leaf(\_) & Branch(\_,\_) \\ Leaf(\_) & Leaf(\_) \end{pmatrix} \tag{1}$$

We write the special pattern matrix of $m \times n$ as

- $\emptyset$: for matrix with $m = 0 \wedge n \geq 0$
- $()$: for matrix with $m \geq 0 \wedge n = 0$

for matrix $m = 0 \wedge n = 0$ we write $\emptyset$

ML Pattern matching (filter, match)

- P is pattern matrix
- $\vec{v}$ is a value vector $(v_1, ...v_n)$

when n equals to the width of P, **Row number i of P filters** $\vec{v}$ if the following two conditions are satisfied.

- $(p^i_1...p^i_n) \preceq (v_1...v_n)$
- $\forall j < i, (p^j_1...p^j_n) \preceq (v_1...v_n)$

(also, we say $\vec{v}$ **matches** row number i of P)

Instance Relations for Matrices

Vector $\vec{v}$ is an **instance of Matrix P** if and only if $\exists i \in [1..m] s.t. (p^i_1...p^i_n) \preceq (v_1...v_n)$ (say $\vec{v}$ matches P, or P filters $\vec{v}$)

We write $P^{[1..i)}$ as the 1 to i-1 rows of P (who is $(i-1) \times n$ matrix).

$\vec{v}$ matches P $\iff P^{[1..i)} \npreceq \vec{v} \wedge \vec{p^i} \preceq \vec{v}$

Exhaustiveness

P is **exhaustive** if and only if $\forall \vec{v}$ of the appropriate type P filters $\vec{v}$

Useless Clause

Row number i of P is **useless** if and only if $\nexists \vec{v}$ that matches row number i of P.

Useful Clause

We calculate `Exhaustiveness` and `Useless clause` by the following definition `Useful Clause`

- P is pattern matrix of $m \times n$
- $\vec{v}$ is a value vector $(v_1, ...v_n)$

$\vec{v}$ is **useful with respect to P** if and only if $\exists \vec{v} s.t. P \npreceq \vec{v} \vee \vec{q} \preceq \vec{v}$

intuitively, $\vec{v}$ doesn't match P and $\vec{v}$ match a pattern vector $\vec{q}$

- $U(P, \vec{q})$ = there exists $\vec{v}$ such that $P \npreceq \vec{v} \vee \vec{q} \preceq \vec{v}$
- $M(P, \vec{q}) = \vec{v} | P \npreceq \vec{v} \vee \vec{q} \preceq \vec{v}$

Propisition

- Matrix P is **exhaustive** iff $U(P, (\_, ...\_)) = false$
  - if we add `wildcard` patterns to `P` and it is NOT `useful`, P is `exhaustive`.
- Row number i of matrix P is **useless** iff $U(P^{[1..i)}, \vec{p^i}) = false$
  - Is there a value vector that doens't match 1 to (i-1)-th pattern, and match i-th case?

Now we broke down the `exhaustiveness` and `usefulness` problem into calculating a $U(P, \vec{q})$, so how can we calculate U. (recursion on `n`).

Calculate $U(P, \vec{q})$

We define recursive function U_{rec} and proove $U(P, \vec{q}) = U_{rec}(P, \vec{q})$

Base case

If P is $m \times 0$ matrix (remember we write such matrix as $()$), it depends on `m`.

- if $m > 0, U_{rec}(P, ()) = U_{rec}((), ()) = false$
- if $m = 0, U_{rec}(\emptyset, \vec{q}) = true$
  - $\emptyset$ never filter anything.
  - though $\vec{q} = ()$ we assume there exists at least one value vector for any pattern vector.

Induction

When $n > 0$ case analysis on q_1 (the first pattern of $\vec{q}$)

case1 q_1 is constructed pattern

$q_1 = c(r_1 ... r_a)$

We define **specialized matrix S(c, P)** and $U_{rec}(P, \vec{q}) = U_{rec}(S(c, P), S(c, \vec{q}))$

$S(c, P)$ has $(n + a - 1)$ columns, and it's i-th row is defined based on p_1^i (P's row i, first column).

p_1^i	row number i of $S(c, P)$
$c(r_1 ... r_a)$	$r_1 ... r_a, p_2^i ... p_n^i$
$c'(r_1 ... r_a)$ (c ≠c')	No row
_	$\_ \cdots \_, p_2^i ... p_n^i$
$(r_1 \| r_2)$	$S(c, \begin{pmatrix} r_1, p_2^i ... p_n^i \\ r_2, p_2^i ... p_n^i \end{pmatrix})$

Intuitively, we can interpret

- First case: “stripping”the constructor and check the patterns inside of a root construcor.
- Second case: It's obvious $\vec{q}$ is always useful with respect to $c'(...)$, therefore we can remove the row from specialized matrix to skip further checking.
- Final case: just deconstruct or pattern into two separate patterns

case2 q_1 is wildcard pattern

let $\Sigma = c_1 ... c_z$ the set of constructors that appears root constructor of P's first column.

For example,

$$P = \begin{pmatrix} Branch(Leaf(_), \_) & ... \\ \_ & ... \end{pmatrix} \Sigma = \{Branch\} \tag{2}$$

$$P = \begin{pmatrix} Branch(Leaf(_), \_) & ... \\ Leaf(\_) & ... \\ \_ & ... \end{pmatrix} \Sigma = \{Branch, Leaf\} \tag{3}$$

Branch based on Σ is **complete signature** or not. (Σ is complete signature iff it covers all constructors of the type).

(a) Σ consists a complete signature $U_{rec}(P, \vec{q}) = \vee_{k=1}^{z} U_{rec}(S(c_k, P), S(c_k, \vec{q}))$

Intuitively, if values are not nested P's first column is obviously exhaustive (since Σ consists of complete signature), to check the patterns inside of root constructors we deconstruct them by calculating specialized matrices.

(b) Σ doesn't consists of a complete signature Define **new default matrix** $D(P)$ of width $n-1$. Row number i of D(P) is defined based on p_1^i

p_1^i	row number i of $S(c, P)$
$c_k(t_1 ... t_{ak})$	No row
_	$p_2^i ... p_n^i$
$(r_1 \| r_2)$	$D(\begin{pmatrix} r_1, p_2^i ... p_n^i \\ r_2, p_2^i ... p_n^i \end{pmatrix})$

$U_{rec}(P, (_, q_2 ... q_n)) = U_{rec}(D(P), (q_2, ... q_n))$

case3 $q_1 = (r_1 \| r_2)$

$U_{rec}(P, ((r_1 \| r_2), q_2, ...q_n)) = U_{rec}(P, (r_1, q_2...q_n)) \vee U_{rec}(P, (r_2, q_2, ...q_n))$

Examples of U_{rec}

Example1

Consider the forllwing pattern match

```
(x: Tree) match {
  case Branch(Leaf(_), Leaf(_)) => ???
  case Branch(_, _) => ???
}
```

and checking if the second case is useful or not.

$$P = \Big(Branch(Leaf(\_), Leaf(\_))\Big), \vec{q} = (Branch(\_, \_)) \tag{4}$$

case1 since q_1 is constructed pattern

$$S(c, P) = \Big(Leaf(\_), Leaf(\_)\Big), S(c, \vec{q}) = (\_, \_) \tag{5}$$

$U_{rec}(P, \vec{q}) = U_{rec}(S(c, P), S(c, \vec{q}))$

case2 since $q_1 = \_$. $\Sigma = \{Leaf\}$ not complete signature

$D(S(c, P)) = \emptyset$

$U_{rec}(S(c, P), S(c, \vec{q})) = U_{rec}(\emptyset, \_) = true$

Therefore, $Branch(\_, \_)$ is useful crals with respect to P.

Example2

```
(x: Tree) match {
  case Branch(_, _) => ???
  case _            => ???
  case Leaf(_)      => ???
}
```

check if the third case is useful or not.

$$P = \begin{pmatrix} Branch(Leaf(\_), Leaf(\_)) \\ \_ \end{pmatrix}, \vec{q} = (Leaf(\_)) \tag{6}$$

case1 since $q_1 = Leaf(\_)$

$$S(c, P) = \begin{pmatrix} Norow \\ \_ \end{pmatrix} = \begin{pmatrix} \_ \end{pmatrix}, S(c, \vec{q}) = (\_) \tag{7}$$

$U_{rec}(\begin{pmatrix} \_ \end{pmatrix}, (\_))$

consider case2, $\Sigma = \{\}$ obviously it's not a complete signature.

$D(\begin{pmatrix} \_ \end{pmatrix}) = ()$ so

$U_{rec}(\begin{pmatrix} \_ \end{pmatrix}, (\_)) = U_{rec}((), ()) = false$

Example3

```
(x: Tree) match {
  case Branch(Leaf(_), Leaf(_)) => ???
  case Branch(_, _)             => ???
  case Leaf(_)                  => ???
}
```

and check if this pattern matching is exhaustive or not.

$$P = \begin{pmatrix} Branch(Leaf(\_), Leaf(\_)) \\ Branch(\_, \_) \\ Leaf(\_) \end{pmatrix}, \vec{q} = (\_) \tag{8}$$

case2, $\Sigma = \{Branch, Leaf\}$ complete signature.

$U_{rec}(P, \vec{q}) = U_{rec}(S(Branch, P), S(Branch, \vec{q})) \vee U_{rec}(S(Leaf, P), S(Leaf, \vec{q}))$

$$S(Branch, P) = \begin{pmatrix} Leaf(\_), Leaf(\_) \\ \_, \_ \end{pmatrix}, S(Branch, \vec{q}) = (\_, \_) \tag{9}$$

$$S(Leaf, P) = \left(\_,\right), S(Leaf, \vec{q}) = (\_) \tag{10}$$

$U_{rec}(S(Branch, P), S(Branch, \vec{q})) = U_{rec}((\_), (\_)) = false$

$U_{rec}(S(Leaf, P), S(Leaf, \vec{q})) = U_{rec}((\_), (\_)) = false$

$U_{rec}(P, \vec{q}) = U_{rec}(S(Branch, P), S(Branch, \vec{q})) \vee U_{rec}(S(Leaf, P), S(Leaf, \vec{q})) = false$

Therefore P is exhaustive.