Ambiguous pattern variables

Gabriel Scherer, Luc Maranget, Thomas Réfis

Northeastern University

September 22, 2016

The problem

```
type α exp =
  | Const of α
  | Mul of α exp * α exp

let is_neutral n = (n = 1)

let mul a b = match a, b with
| (Const n, v) | (v, Const n)
  when is_neutral n -> v
| a, b -> Mul (a, b)
```

The problem

```
type α exp =
  | Const of α
  | Mul of α exp * α exp

let is_neutral n = (n = 1)

let mul a b = match a, b with
| (Const n, v) | (v, Const n)
  when is_neutral n -> v
| a, b -> Mul (a, b)
```

```
mul (Const 2) (Const 1)
```

The problem

```
type α exp =
  | Const of α
  | Mul of α exp * α exp

let is_neutral n = (n = 1)

let mul a b = match a, b with
| (Const n, v) | (v, Const n)
  when is_neutral n -> v
| a, b -> Mul (a, b)
```

```
mul (Const 2) (Const 1)
= Mul (Const 2, Const 1)
```

ML patterns (formally)

$$
\begin{array}{rclr}
p & ::= & & \text{pattern} \\
 & \mid & \_ & \text{wildcard} \\
 & \mid & p \text{ as } x & \text{variable binding} \\
 & \mid & K(p_1, ..., p_n) & \text{constructor pattern} \\
 & \mid & p \mid q & \text{or-pattern}
\end{array}
$$

Variable patterns x are sugar for $(\_ \text{ as } x)$.

Pair patterns (p, q) are a special case of constructor pattern.

A **clause** of the form

```
| p when g -> e
```

matches p first, then test if g holds, and only then takes the branch to e.

The Clash

`(`p `|` q`)` `when` $g \rightarrow e$
readers think the guard g will test **both** p and q – angelic choice.

The specification clearly says otherwise:
`(p | q)` is left-to-right, and only then g is tried.

The Clash

`(`p `|` q`) when` $g \rightarrow e$
readers think the guard g will test **both** p and q – angelic choice.

The specification clearly says otherwise:
`(p | q)` is left-to-right, and only then g is tried.

Note: specifying evaluation order is not always good, after all...

The Clash

$(p \mid q)$ `when` $g \rightarrow e$
readers think the guard g will test **both** p and q – angelic choice.

The specification clearly says otherwise:
`(p | q)` is left-to-right, and only then g is tried.

Note: specifying evaluation order is not always good, after all...

Note: automatically turning this into $(p$ `when` $g) \mid (q$ `when` $g)$ does not work:

- changing the semantics of existing code: nope

The Clash

`(`p `|` q`) when` g $\rightarrow$ e
readers think the guard g will test **both** p and q – angelic choice.

The specification clearly says otherwise:
`(p | q)` is left-to-right, and only then g is tried.

Note: specifying evaluation order is not always good, after all...

Note: automatically turning this into `(`p `when` g`) | (`q `when` g`)` does not work:

- changing the semantics of existing code: nope
- nested guards don't exist and would break exhaustivity checking, etc.

The Clash

$(p \ | \ q)$ `when` $g \rightarrow e$
readers think the guard g will test **both** p and q – angelic choice.

The specification clearly says otherwise:
`(p | q)` is left-to-right, and only then g is tried.

Note: specifying evaluation order is not always good, after all...

Note: automatically turning this into $(p$ `when` $g)$ | $(q$ `when` $g)$ does not work:

- changing the semantics of existing code: nope
- nested guards don't exist and would break exhaustivity checking, etc.
- side-effects in g would be duplicated

The Clash

`(`p `|` q`) when` g $\rightarrow$ e
readers think the guard g will test **both** p and q – angelic choice.

The specification clearly says otherwise:
`(p | q)` is left-to-right, and only then g is tried.

Note: specifying evaluation order is not always good, after all...

Note: automatically turning this into `(`p `when` g`) | (`q `when` g`)` does not work:

- changing the semantics of existing code: nope
- nested guards don't exist and would break exhaustivity checking, etc.
- side-effects in g would be duplicated
- what about nested or-patterns? `(p | q)` may be deep.

At least complain about it!

Warn when

$$p \ \texttt{when} \ g \ \rightarrow \ e$$

and

- a value may match p in several ways (or-patterns)
- the test g may depend on which choice is taken: it contains **ambiguous pattern variables**

At least complain about it!

Warn when

$$p \ \texttt{when} \ g \ \rightarrow \ e$$

and

- a value may match p in several ways (or-patterns)
- the test g may depend on which choice is taken: it contains **ambiguous pattern variables**

  `(a, (`p `|` q`)) when a < 10` $\rightarrow$ `...`

At least complain about it!

Warn when

$$p \texttt{ when } g \rightarrow e$$

and

- a value may match p in several ways (or-patterns)
- the test g may depend on which choice is taken: it contains **ambiguous pattern variables**

```
(a, (p | q)) when a < 10 → ...
(a, p) | (a, q) when a < 10 → ...
```

At least complain about it!

Warn when

$$p \ \texttt{when} \ g \ \rightarrow \ e$$

and

- a value may match p in several ways (or-patterns)
- the test g may depend on which choice is taken: it contains **ambiguous pattern variables**

```
(a, (p | q)) when a < 10 → ...
(a, p) | (a, q) when a < 10 → ...
(Some v, e) | (e, Some v) when v = 0 → ...
```

At least complain about it!

Warn when

$$p \ \texttt{when} \ g \rightarrow e$$

and

- a value may match p in several ways (or-patterns)
- the test g may depend on which choice is taken: it contains **ambiguous pattern variables**

```
(a, (p | q)) when a < 10 → ...
(a, p) | (a, q) when a < 10 → ...
(Some v, e) | (e, Some v) when v = 0 → ...
(Some v, None) | (None, Some v) when v = 0 → ...
```

Our contribution

- an algorithm to detect ambiguous pattern variables
- implemented in OCaml 4.03 (released last April)

Demo

How to implement this warning? (Attempts.)

As for all pattern matching questions (compilation, exhaustivity, usefulness...):

pattern matrices

(the take-away of this talk)

Pattern matrix

A matrix: a space of matcheable values that share a common prefix.

$$
\begin{array}{c}
C_1[\Box_1, \ldots, \Box_n] \\
C_2[\Box_1, \ldots, \Box_n] \\
\ldots \\
C_n[\Box_1, \ldots, \Box_n]
\end{array}
\begin{bmatrix}
p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\
p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\
\vdots & \vdots & \ddots & \vdots \\
p_{m,1} & p_{m,2} & \cdots & p_{m,n}
\end{bmatrix}
$$

- rows: disjunction, alternative
- columns: sub-patterns matched in parallel
- contexts: common prefix, possibly different bindings

$$
\begin{array}{c}
((\mathtt{S}\ \Box)\ \mathtt{as}\ v, \Box) \\
(\mathtt{S}\ \Box, (\Box\ \mathtt{as}\ v))
\end{array}
\begin{bmatrix}
\mathtt{true} & \mathtt{N} \\
\_ & \mathtt{S\ false}
\end{bmatrix}
\quad \text{represents} \quad
\begin{array}{l}
\mid ((\mathtt{S}\ \mathtt{true})\ \mathtt{as}\ v, \mathtt{N}) \\
\mid (\mathtt{S}\ \_, \mathtt{S\ false}\ \mathtt{as}\ v)
\end{array}
$$

Matrix operation: splitting a row (1)

All head constructors.

$$\begin{array}{c} C_1[\square, \square] \\ C_2[\square, \square] \\ C_3[\square, \square] \end{array} \begin{bmatrix} & \mathtt{N} & p_{1,2} \\ \mathtt{S} & p_{2,1} & p_{2,2} \\ \mathtt{S} & p_{3,1} & p_{3,2} \end{bmatrix}$$

Matrix operation: splitting a row (1)

All head constructors.

$$\begin{array}{l} C_1[\square, \square] \\ C_2[\square, \square] \\ C_3[\square, \square] \end{array} \begin{bmatrix} \mathtt{N} & & p_{1,2} \\ \mathtt{S} & p_{2,1} & p_{2,2} \\ \mathtt{S} & p_{3,1} & p_{3,2} \end{bmatrix}$$

$$\Longrightarrow$$

$$C_1[\mathtt{N}, \square] \begin{bmatrix} p_{1,2} \end{bmatrix} \qquad \begin{array}{l} C_2[\mathtt{S}\ \square, \square] \\ C_3[\mathtt{S}\ \square, \square] \end{array} \begin{bmatrix} p_{2,1} & p_{2,2} \\ p_{3,1} & p_{3,2} \end{bmatrix}$$

Matrix operation: splitting a row (2)

Some head constructors.

$$
\begin{array}{l}
C_1[\square, \square] \\
C_2[\square, \square] \\
C_3[\square, \square]
\end{array}
\begin{bmatrix}
\texttt{N} & p_{1,2} \\
\texttt{x} & p_{2,2} \\
\texttt{S}\ p_{3,1} & p_{3,2}
\end{bmatrix}
$$

$$
\Longrightarrow \quad
\begin{array}{l}
C_1[\square, \square] \\
C_2[\square, \square] \\
C_3[\square, \square]
\end{array}
\begin{bmatrix}
\texttt{N} & p_{1,2} \\
(\texttt{N} \mid \texttt{S}\ \_)\ \texttt{as}\ x & p_{2,2} \\
\texttt{S}\ p_{3,1} & p_{3,2}
\end{bmatrix}
$$

$$
\Longrightarrow \quad
\begin{array}{c}
C_1[\square, \square] \\
C_2[\square\ \texttt{as}\ x, \square] \\
C_2[\square\ \texttt{as}\ x, \square] \\
C_3[\square, \square]
\end{array}
\begin{bmatrix}
\texttt{N} & p_{1,2} \\
\texttt{N} & p_{2,2} \\
\texttt{S}\ \_ & p_{2,2} \\
\texttt{S}\ p_{3,1} & p_{3,2}
\end{bmatrix}
$$

Matrix operation: (3)

No head constructors.

$$\begin{matrix} C_1[\square, \square] \\ C_2[\square, \square] \\ C_3[\square, \square] \end{matrix} \begin{bmatrix} \_ & p_{1,2} \\ \mathrm{x} & p_{2,2} \\ \_ & p_{3,2} \end{bmatrix}$$

$$\Longrightarrow$$

Matrix operation: (3)

No head constructors.

$$\begin{matrix} C_1[\square, \square] \\ C_2[\square, \square] \\ C_3[\square, \square] \end{matrix} \begin{bmatrix} \_ & p_{1,2} \\ \mathrm{x} & p_{2,2} \\ \_ & p_{3,2} \end{bmatrix}$$

$$\Longrightarrow$$

$$\begin{matrix} C_1[\_, \square] \\ C_2[\mathrm{x}, \square] \\ C_3[\_, \square] \end{matrix} \begin{bmatrix} p_{1,2} \\ p_{2,2} \\ p_{3,2} \end{bmatrix}$$

Typical matrix-based algorithm, simplified

Manipulate sets of matrices.

Start from a single matrix (single-element set).
Split matrices, repeat.
Stop when all matrices are empty.

Compute your answer from that.

Our algorithm, on an example (1)

```
(S v, a) | (a, S v) when is_neutral v → ...
```

$$\square \left[(\mathtt{S}\ v, a) \mid (a, \mathtt{S}\ v) \right]$$

Our algorithm, on an example (1)

```
(S v, a) | (a, S v) when is_neutral v → ...
```

$$\square \left[(\mathtt{S}\ v, a) \mid (a, \mathtt{S}\ v) \right]$$

$$\begin{matrix} \square \\ \square \end{matrix} \begin{bmatrix} (\mathtt{S}\ v, a) \\ (a, \mathtt{S}\ v) \end{bmatrix}$$

Our algorithm, on an example (1)

```
(S v, a) | (a, S v) when is_neutral v → ...
```

$$\Box\ \left[(\mathtt{S}\ v, a) \mid (a, \mathtt{S}\ v)\right]$$

$$\begin{matrix}\Box \\ \Box\end{matrix} \begin{bmatrix}(\mathtt{S}\ v, a) \\ (a, \mathtt{S}\ v)\end{bmatrix}$$

$$\begin{matrix}(\Box, \Box) \\ (\Box, \Box)\end{matrix} \begin{bmatrix}\mathtt{S}\ v & a \\ a & \mathtt{S}\ v\end{bmatrix}$$

Our algorithm, on an example (1)

```
(S v, a) | (a, S v) when is_neutral v → ...
```

$$\square \left[(\mathtt{S}\ v, a) \mid (a, \mathtt{S}\ v) \right]$$

$$\begin{array}{c} \square \\ \square \end{array} \begin{bmatrix} (\mathtt{S}\ v, a) \\ (a, \mathtt{S}\ v) \end{bmatrix}$$

$$\begin{array}{c} (\square, \square) \\ (\square, \square) \end{array} \begin{bmatrix} \mathtt{S}\ v & a \\ a & \mathtt{S}\ v \end{bmatrix}$$

$$\begin{array}{c} (\square, \square) \\ (\square\ \mathtt{as}\ a, \square) \\ (\square\ \mathtt{as}\ a, \square) \end{array} \begin{bmatrix} \mathtt{S}\ v & a \\ \mathtt{S}\ \_ & \mathtt{S}\ v \\ \mathtt{N} & \mathtt{S}\ v \end{bmatrix}$$

Our algorithm, on an example (1)

```
(S v, a) | (a, S v) when is_neutral v → ...
```

$$\Box\ \big[(\mathrm{S}\ v, a) \mid (a, \mathrm{S}\ v)\big]$$

$$\begin{array}{c} \Box \\ \Box \end{array} \begin{bmatrix} (\mathrm{S}\ v, a) \\ (a, \mathrm{S}\ v) \end{bmatrix}$$

$$\begin{array}{c} (\Box, \Box) \\ (\Box, \Box) \end{array} \begin{bmatrix} \mathrm{S}\ v & a \\ a & \mathrm{S}\ v \end{bmatrix}$$

$$\begin{array}{c} (\Box, \Box) \\ (\Box \text{ as } a, \Box) \\ (\Box \text{ as } a, \Box) \end{array} \begin{bmatrix} \mathrm{S}\ v & a \\ \mathrm{S}\ \_ & \mathrm{S}\ v \\ \mathrm{N} & \mathrm{S}\ v \end{bmatrix}$$

$$\begin{array}{c} (\mathrm{S}\ \Box, \Box) \\ (\mathrm{S}\ \Box \text{ as } a, \Box) \end{array} \begin{bmatrix} v & a \\ \_ & \mathrm{S}\ v \end{bmatrix} \qquad (\mathrm{N} \text{ as } a, \Box)\ \big[\mathrm{S}\ v\big]$$

Our algorithm, on an example (2)

```
(S v, a) | (a, S v) when is_neutral v → ...
```

$$\begin{matrix} (\mathtt{S}\ v, \square) \\ (\mathtt{S}\ \_\ \mathtt{as}\ a, \square) \end{matrix} \begin{bmatrix} a \\ \mathtt{S}\ v \end{bmatrix} \qquad (\mathtt{N}\ \mathtt{as}\ a, \square) \begin{bmatrix} \mathtt{S}\ v \end{bmatrix}$$

Our algorithm, on an example (2)

```
(S v, a) | (a, S v) when is_neutral v → ...
```

$$\begin{matrix} (\mathtt{S}\ v, \square) \\ (\mathtt{S}\ \_\ \mathtt{as}\ a, \square) \end{matrix} \begin{bmatrix} a \\ \mathtt{S}\ v \end{bmatrix} \qquad (\mathtt{N}\ \mathtt{as}\ a, \square)\ \begin{bmatrix} \mathtt{S}\ v \end{bmatrix}$$

$$\begin{matrix} (\mathtt{S}\ v, (\square\ \mathtt{as}\ a)) \\ (\mathtt{S}\ v, (\square\ \mathtt{as}\ a)) \\ (\mathtt{S}\ \_\ \mathtt{as}\ a, \square) \end{matrix} \begin{bmatrix} \mathtt{S}\ \_ \\ \mathtt{N} \\ \mathtt{S}\ v \end{bmatrix} \qquad (\mathtt{N}\ \mathtt{as}\ a, \square)\ \begin{bmatrix} \mathtt{S}\ v \end{bmatrix}$$

Our algorithm, on an example (2)

```
(S v, a) | (a, S v) when is_neutral v → ...
```

$$\begin{matrix} (\mathtt{S}\ v, \square) \\ (\mathtt{S}\ \_\ \mathtt{as}\ a, \square) \end{matrix} \begin{bmatrix} a \\ \mathtt{S}\ v \end{bmatrix} \qquad (\mathtt{N}\ \mathtt{as}\ a, \square)\ \begin{bmatrix} \mathtt{S}\ v \end{bmatrix}$$

$$\begin{matrix} (\mathtt{S}\ v, (\square\ \mathtt{as}\ a)) \\ (\mathtt{S}\ v, (\square\ \mathtt{as}\ a)) \\ (\mathtt{S}\ \_\ \mathtt{as}\ a, \square) \end{matrix} \begin{bmatrix} \mathtt{S}\ \_ \\ \mathtt{N} \\ \mathtt{S}\ v \end{bmatrix} \qquad (\mathtt{N}\ \mathtt{as}\ a, \square)\ \begin{bmatrix} \mathtt{S}\ v \end{bmatrix}$$

$$\begin{matrix} (\mathtt{S}\ v, (\mathtt{S}\ \square\ \mathtt{as}\ a)) \\ (\mathtt{S}\ \_\ \mathtt{as}\ a, \square) \end{matrix} \begin{bmatrix} \_ \\ v \end{bmatrix} \qquad (\mathtt{S}\ v, \mathtt{N})\ \begin{bmatrix} \cdot \end{bmatrix} \qquad (\mathtt{N}\ \mathtt{as}\ a, \square)\ \begin{bmatrix} \mathtt{S}\ v \end{bmatrix}$$

Our algorithm, on an example (3)

```
(S v, a) | (a, S v) when is_neutral v → ...
```

$$\begin{matrix} (\mathtt{S}\ \ v\ , (\mathtt{S}\ \_\ \mathtt{as}\ a)) \\ (\mathtt{S}\ \_\ \mathtt{as}\ a,\ \ v\ ) \end{matrix} \begin{bmatrix} \cdot \\ \cdot \end{bmatrix} \qquad (\mathtt{S}\ v, \mathtt{N}) \begin{bmatrix} \cdot \end{bmatrix} \qquad (\mathtt{N}\ \mathtt{as}\ a, \mathtt{S}\ v) \begin{bmatrix} \cdot \end{bmatrix}$$

Our algorithm, on an example (3)

```
(S v, a) | (a, S v) when is_neutral v → ...
```

$$\begin{matrix} (\mathtt{S}\ \boxed{v}, (\mathtt{S}\ \_\ \mathtt{as}\ a)) \\ (\mathtt{S}\ \_\ \mathtt{as}\ a, \boxed{v}) \end{matrix} \begin{bmatrix} \cdot \\ \cdot \end{bmatrix} \qquad (\mathtt{S}\ v, \mathtt{N}) \begin{bmatrix} \cdot \end{bmatrix} \qquad (\mathtt{N}\ \mathtt{as}\ a, \mathtt{S}\ v) \begin{bmatrix} \cdot \end{bmatrix}$$

Actual implementation

No need for sets of matrices: we recursively traverse the set/tree of splits. Long time, but short space.

Most algorithms don't keep contexts, they retain only what they need. In our case, variable bindings positions.

See the extended abstract for a more algorithmic presentation.

Conclusion

- Arthur Charguéraud, Martin Clochard and Claude Marché: a problem
- us: a solution

Future work: negative information.