Ambiguous pattern variables

Gabriel Scherer, Luc Maranget, Thomas Réfis

Northeastern University

September 22, 2016
The problem

```plaintext
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

```ml
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)

\[ p ::= \quad \text{pattern} \]
\[ | \quad _ \quad \quad \quad \quad \text{wildcard} \]
\[ | \quad p \text{ as } x \quad \quad \quad \text{variable binding} \]
\[ | \quad K(p_1, \ldots, p_n) \quad \quad \text{constructor pattern} \]
\[ | \quad p \mid q \quad \quad \quad \text{or-pattern} \]

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

\[ | \quad p \text{ when } g \rightarrow e \]

matches \( p \) first, then test if \( g \) holds, and only then takes the branch to \( e \).
The Clash

\((p \mid q) \text{ when } g \rightarrow e\)

readers think the guard \(g\) will test \textbf{both} \(p\) and \(q\) – angelic choice.

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

\((p \mid q) \text{ when } g \rightarrow e\)

readers think the guard \(g\) will test \textbf{both} \(p\) and \(q\) – angelic choice.

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

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

\[(p \Vert q) \text{ when } g \rightarrow e\]

readers think the guard \(g\) will test both \(p\) and \(q\) – angelic choice.

The specification clearly says otherwise:
\[(p \Vert q)\] 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 \((p \text{ when } g) \Vert (q \text{ when } g)\) does not work:

- changing the semantics of existing code: nope
The Clash

\((p \mid q) \text{ when } g \rightarrow e\)

readers think the guard \(g\) will test \textbf{both} \(p\) and \(q\) – angelic choice.

The specification clearly says otherwise:
\((p \mid q)\) 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 \((p \text{ when } g) \mid (q \text{ when } g)\) does not work:

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

\((p \mid q) \text{ when } g \rightarrow e\)

readers think the guard \(g\) will test both \(p\) and \(q\) – angelic choice.

The specification clearly says otherwise:

\((p \mid q)\) 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 \((p \text{ when } g) \mid (q \text{ when } g)\) 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

\((p \mid q) \text{ when } g \rightarrow e\)

readers think the guard \(g\) will test \textbf{both} \(p\) and \(q\) – angelic choice.

The specification clearly says otherwise:
\((p \mid q)\) 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 \((p \text{ when } g) \mid (q \text{ when } g)\) 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? \((p \mid q)\) may be deep.
At least complain about it!

Warn when

\[ p \text{ when } g \rightarrow e \]

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

\[ p \text{ when } g \rightarrow e \]

and

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

\[(a, (p \mid q)) \text{ when } a < 10 \rightarrow \ldots\]
At least complain about it!

Warn when

\[ p \text{ when } g \rightarrow e \]

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

\((a, (p | q)) \text{ when } a < 10 \rightarrow \ldots\)
\((a, p) | (a, q) \text{ when } a < 10 \rightarrow \ldots\)
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

\[
(a, (p \mid q)) \texttt{ when } a < 10 \rightarrow \ldots
\]
\[
(a, p) \mid (a, q) \texttt{ when } a < 10 \rightarrow \ldots
\]
\[
(\text{Some } v, e) \mid (e, \text{Some } v) \texttt{ when } v = 0 \rightarrow \ldots
\]
At least complain about it!

Warn when

\[ p \text{ when } g \rightarrow e \]

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

\[
(a, (p \mid q)) \text{ when } a < 10 \rightarrow \ldots \\
(a, p) \mid (a, q) \text{ when } a < 10 \rightarrow \ldots \\
(\text{Some } v, e) \mid (e, \text{Some } v) \text{ when } v = 0 \rightarrow \ldots \\
(\text{Some } v, \text{None}) \mid (\text{None, Some } v) \text{ when } v = 0 \rightarrow \ldots
\]
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 matchable values that share a common prefix.

\[
\begin{align*}
C_1[\square_1, \ldots, \square_n] &= \left[ \begin{array}{cccc} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,1} & p_{m,2} & \cdots & p_{m,n} \end{array} \right] \\
C_2[\square_1, \ldots, \square_n] &= \left[ \begin{array}{cccc} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,1} & p_{m,2} & \cdots & p_{m,n} \end{array} \right] \\
\vdots & \\
C_n[\square_1, \ldots, \square_n] &= \left[ \begin{array}{cccc} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,1} & p_{m,2} & \cdots & p_{m,n} \end{array} \right]
\end{align*}
\]

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

\[
\begin{align*}
((S \, \square) \, as \, v, \square) \left[ \begin{array}{cc} true & N \end{array} \right] \quad \text{represents} \quad & \left| \quad ((S \, true) \, as \, v, N) \right. \\
(S \, \square, (\square \, as \, v)) \left[ \begin{array}{c} _{} \end{array} \right] \left| (S \, _, S \, false \, as \, v) \right. \\
\end{align*}
\]
Matrix operation: splitting a row (1)

All head constructors.

\[
\begin{align*}
C_1[\square, \square] & \begin{bmatrix} N & p_{1,2} \\
C_2[\square, \square] & S \begin{bmatrix} p_{2,1} & p_{2,2} \\
C_3[\square, \square] & S \begin{bmatrix} p_{3,1} & p_{3,2} \\
\end{bmatrix}
\end{bmatrix}
\end{align*}
\]
Matrix operation: splitting a row (1)

All head constructors.

\[
\begin{align*}
C_1[\square, \square] & \quad \left[ \begin{array}{cc} N & p_{1,2} \\ \end{array} \right] \\
C_2[\square, \square] & \quad \left[ \begin{array}{cc} S & p_{2,1} \\
S & p_{2,2} \end{array} \right] \\
C_3[\square, \square] & \quad \left[ \begin{array}{cc} S & p_{3,1} \\
S & p_{3,2} \end{array} \right]
\end{align*}
\]
Matrix operation: splitting a row (2)

Some head constructors.

\[
\begin{align*}
C_1[\Box, \Box] & \begin{bmatrix} N & p_{1,2} \\ \end{bmatrix} \\
C_2[\Box, \Box] & \begin{bmatrix} x & p_{2,2} \\ \end{bmatrix} \\
C_3[\Box, \Box] & \begin{bmatrix} S & p_{3,1} & p_{3,2} \\ \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\implies C_1[\Box, \Box] & \begin{bmatrix} N & p_{1,2} \\ \end{bmatrix} \\
C_2[\Box, \Box] & \begin{bmatrix} (N \mid S \_ ) \text{ as} & x & p_{2,2} \\ \end{bmatrix} \\
C_3[\Box, \Box] & \begin{bmatrix} S & p_{3,1} & p_{3,2} \\ \end{bmatrix}
\end{align*}
\]

\[
\begin{align*}
\implies C_1[\Box, \Box] & \begin{bmatrix} N & p_{1,2} \\ \end{bmatrix} \\
C_2[\Box \text{ as } x, \Box] & \begin{bmatrix} N & p_{2,2} \\ \end{bmatrix} \\
C_2[\Box \text{ as } x, \Box] & \begin{bmatrix} S & \_ & p_{2,2} \\ \end{bmatrix} \\
C_3[\Box, \Box] & \begin{bmatrix} S & p_{3,1} & p_{3,2} \\ \end{bmatrix}
\end{align*}
\]
No head constructors.

\[
\begin{array}{c}
C_1[\square, \square] \begin{bmatrix} - & p_{1,2} \\ \end{bmatrix} \\
C_2[\square, \square] \begin{bmatrix} x & p_{2,2} \\ \end{bmatrix} \\
C_3[\square, \square] \begin{bmatrix} - & p_{3,2} \\ \end{bmatrix}
\end{array}
\]
Matrix operation: (3)

No head constructors.

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

\[ \Rightarrow \]
\[ C_1[\_, \square] \begin{bmatrix} p_{1,2} \end{bmatrix} \]
\[ C_2[\_x, \square] \begin{bmatrix} p_{2,2} \end{bmatrix} \]
\[ C_3[\_, \square] \begin{bmatrix} p_{3,2} \end{bmatrix} \]
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)

\[(S \ v, a) \mid (a, S \ v) \texttt{ when is\_neutral } v \rightarrow \ldots\]

\[\square [(S \ v, a) \mid (a, S \ v)]\]
Our algorithm, on an example (1)

\((S \ v, a) \ | \ (a, S \ v)\) when is_neutral \(v\) \(\rightarrow\) ...

\[\square [(S \ v, a) \ | \ (a, S \ v)]\]

\[\square [(S \ v, a)]\]

\[\square [(a, S \ v)]\]
Our algorithm, on an example (1)

\((S \; v, \; a) \mid (a, \; S \; v)\) when \texttt{is\_neutral} \; v \rightarrow \ldots

\begin{align*}
\square & [(S \; v, \; a) \mid (a, \; S \; v)] \\
\square & [(S \; v, \; a)] \\
\square & [(a, \; S \; v)] \\
(\square, \; \square) & [S \; v \; a] \\
(\square, \; \square) & [a \; S \; v]
\end{align*}
Our algorithm, on an example (1)

\((S \, v, \, a) \mid (a, \, S \, v)\) \textit{when} is Neutral \(v \rightarrow \ldots\)

\[
\Box \left[(S \, v, \, a) \mid (a, \, S \, v)\right]
\]

\[
\Box \left[(S \, v, \, a)\right]
\]

\[
\Box \left[(a, \, S \, v)\right]
\]

\[
(\Box, \Box) \left[\begin{array}{c}
S \, v \\
ad
\end{array}\right]
\]

\[
(\Box, \Box) \left[\begin{array}{c}
a \\
S \, v
\end{array}\right]
\]

\[
(\Box, \Box) \left[\begin{array}{c}
S \, v \\
as \, a
\end{array}\right]
\]

\[
(\Box \text{ as } \Box, \Box) \left[\begin{array}{c}
S \, v \\
S \, v
\end{array}\right]
\]

\[
(\Box \text{ as } \Box, \Box) \left[\begin{array}{c}
N \\
S \, v
\end{array}\right]
\]
Our algorithm, on an example (1)

\[(S \ v, \ a) \ | \ (a, \ S \ v) \ \text{when is\_neutral} \ v \rightarrow \ldots\]

\[\square \ [(S \ v, \ a) \ | \ (a, \ S \ v)]\]

\[\square \ [(S \ v, \ a)]\]

\[\square \ [(a, \ S \ v)]\]

\[(\square, \square) \ [S \ v \ a]\]

\[(\square, \square) \ [a \ S \ v]\]

\[(\square, \square) \ [S \ v \ a]\]

\[(\square \ as \ a, \square) \ [S \ - \ S \ v]\]

\[(\square \ as \ a, \square) \ [N \ S \ v]\]

\[(S \ \square, \square) \ [v \ a]\]

\[(S \ \square \ as \ a, \square) \ [- \ S \ v]\]

\[(N \ as \ a, \square) \ [S \ v]\]
Our algorithm, on an example (2)

\[(S \ v, \ a) \mid (a, \ S \ v) \text{ when } \text{is\_neutral} \ v \rightarrow \ldots\]

\[(S \ v, \ □) \quad \begin{bmatrix} a \end{bmatrix} \quad (N \ as \ a, \ □) [S \ v]\]

\[(S - \ as \ a, \ □) \quad \begin{bmatrix} S \ v \end{bmatrix} \quad (N \ as \ a, \ □) [S \ v]\]
Our algorithm, on an example (2)

\[(S \, v, \, a) \mid (a, \, S \, v) \text{ when } \text{is\_neutral} \, v \rightarrow \ldots\]

\[
\begin{align*}
(S \, v, \Box) & \quad \left[ a \right] \\
(S - \text{as} \, a, \Box) & \quad [S \, v]
\end{align*}
\]

\[
\begin{align*}
(N \, \text{as} \, a, \Box) & \quad [S \, v] \\
(N \, \text{as} \, a, \Box) & \quad [S \, v]
\end{align*}
\]
Our algorithm, on an example (2)

\[(S \ v, a) | (a, S \ v) \text{ when is}\_\text{neutral } v \rightarrow \ldots\]

\[
\begin{align*}
(S \ v, \square) & \quad \left[\begin{array}{c} a \\
S \ v \end{array}\right] \\
(S \ as \ a, \square) & \quad \left[\begin{array}{c} S \ v \\
\square \end{array}\right]
\end{align*}
\]

\[
\begin{align*}
(S \ v, (\square \ as \ a)) & \quad \left[\begin{array}{c} S \ v \\
N \end{array}\right] \\
(S \ v, (\square \ as \ a)) & \quad \left[\begin{array}{c} S \ v \\
\square \end{array}\right]
\end{align*}
\]

\[
\begin{align*}
(S \ v, (S \square \ as \ a)) & \quad \left[\begin{array}{c} S \ v \\
\square \end{array}\right] \\
(S \ as \ a, \square) & \quad \left[\begin{array}{c} S \ v \\
\square \end{array}\right]
\end{align*}
\]

\[
\begin{align*}
(N \ as \ a, \square) & \quad \left[\begin{array}{c} S \ v \\
\square \end{array}\right]
\end{align*}
\]
Our algorithm, on an example (3)

\[(S \, v, \, a) \mid (a, \, S \, v) \text{ when is_neutral} \quad v \rightarrow \ldots\]

\[(S \, v, (S \_ \, \text{as} \, a)) [\cdot] \quad (S \, v, N) [\cdot] \quad (N \, \text{as} \, a, \, S \, v) [\cdot]\]
Our algorithm, on an example (3)

\[(S \ v, a) \mid (a, S \ v) \text{ when is neutral } \ v \rightarrow \ldots\]

\[
(S \ v, (S \ _\text{as} \ a)) \ \vdash \quad (S \ v, N) \ \vdash \quad (N \text{as} \ a, S \ v) \]

\[
(S \ _\text{as} \ a, \ v) \]

Gabriel Scherer, Luc Maranget, Thomas Réfis (Northeastern University)
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.