Pattern Match Analysis

For Higher-Order Languages

Ruud Koot

28 August 2012

Well-typed programs cannot "go wrong".

—Robin Milner, 1978

Wrong!

```
*** Exception: Non-exhaustive patterns in function f
```

Partial Functions

$$head\ (x : xs) = x$$

Partial Functions

$$\mathit{head}\ (x : \mathit{xs}) = x$$

$$\begin{aligned}\mathit{main} = {} & \textbf{let}\ \mathit{xs} = \textbf{if}\ \mathit{length}\ \texttt{"foo"} > 5\ \textbf{then}\ [1,2,3]\ \textbf{else}\ [\,] \\ & \textbf{in}\ \mathit{head}\ \mathit{xs}\end{aligned}$$

Partial Functions

$$\mathit{head}\ (x : \mathit{xs}) = x$$

$$\begin{aligned}\mathit{main} = \ & \mathbf{let}\ \mathit{xs} = \mathbf{if}\ \mathit{length}\ \texttt{"foo"} > 5\ \mathbf{then}\ [1,2,3]\ \mathbf{else}\ [\,]\\ & \mathbf{in}\ \mathit{head}\ \mathit{xs}\end{aligned}$$

```
On line 2 you applied the function "head" to the empty
list "xs". The function "head" expects a non-empty list
as its first argument.
```

Compiler Construction

$$desugar :: ComplexAST \rightarrow SimpleAST$$

Compiler Construction

$$desugar :: ComplexAST \rightarrow SimpleAST$$

$$desugar :: AST \rightarrow AST$$

Invariants (1)

$$
\begin{array}{l}
\textbf{type } \mathit{Bitstring} = [\mathit{Int}] \\
\mathit{add} :: \mathit{Bitstring} \to \mathit{Bitstring} \to \mathit{Bitstring} \\
\mathit{add}\ [\,]\qquad y \qquad\quad = y \\
\mathit{add}\ x \qquad [\,] \qquad\quad = x \\
\mathit{add}\ (0 : x)\ (0 : y) = 0 : \mathit{add}\ x\ y \\
\mathit{add}\ (0 : x)\ (1 : y) = 1 : \mathit{add}\ x\ y \\
\mathit{add}\ (1 : x)\ (0 : y) = 1 : \mathit{add}\ x\ y \\
\mathit{add}\ (1 : x)\ (1 : y) = 0 : \mathit{add}\ (\mathit{add}\ [1]\ x)\ y
\end{array}
$$

Invariants (2)

$$
\begin{aligned}
&\mathit{risers} :: \mathit{Ord}\ a \Rightarrow [a] \to [[a]] \\
&\mathit{risers}\ [\,] = [\,] \\
&\mathit{risers}\ [x] = [[x]] \\
&\mathit{risers}\ (x_1 : x_2 : xs) = \textbf{let}\ (s : ss) = \mathit{risers}\ (x_2 : xs) \\
&\qquad \textbf{in if}\ x_1 \leqslant x_2\ \textbf{then}\ (x_1 : s) : ss \\
&\qquad\qquad \textbf{else}\ [x_1] : (s : ss)
\end{aligned}
$$

Computes monotonically increasing segments

$$\mathit{risers}\ [1,3,5,1,2] \ \leadsto\ [[1,3,5],[1,2]]$$

Related Work

Dependent ML (Xi)

Dependent-types over a decidable domain

Related Work

Dependent ML (Xi)
: Dependent-types over a decidable domain

Static Contract Checking (Xu)
: Contracts specified in Haskell

Related Work

Dependent ML (Xi)
: Dependent-types over a decidable domain

Static Contract Checking (Xu)
: Contracts specified in Haskell

Catch (Mitchell)
: Constraint-based, first-order language only

Examples

$$\textbf{True} \quad : \quad \textbf{Bool}^{\{\textbf{True}\}}$$

Examples

$$\begin{array}{rcl} \textbf{True} & : & \textbf{Bool}^{\{\textbf{True}\}} \\ 42 & : & \textbf{Int}^{\{42\}} \end{array}$$

Examples

$$\begin{array}{rcl}
\textbf{True} & : & \textbf{Bool}^{\{\textbf{True}\}} \\
42 & : & \textbf{Int}^{\{42\}} \\
(7, \textbf{False}) & : & (\textbf{Int}^{\{7\}}, \textbf{Bool}^{\{\textbf{False}\}})^{\top}
\end{array}$$

Examples

$$
\begin{array}{rcl}
\textbf{True} & : & \textbf{Bool}^{\{\textbf{True}\}} \\
42 & : & \textbf{Int}^{\{42\}} \\
(7, \textbf{False}) & : & (\textbf{Int}^{\{7\}}, \textbf{Bool}^{\{\textbf{False}\}})^{\top} \\
[3, 2, 1] & : & [\textbf{Int}^{\{1,2,3\}}]^{\{(\_ : \_ : \_ : [])\}}
\end{array}
$$

Examples

$$
\begin{array}{rcl}
\mathbf{True} & : & \mathbf{Bool}^{\{\mathbf{True}\}} \\
42 & : & \mathbf{Int}^{\{42\}} \\
(7, \mathbf{False}) & : & (\mathbf{Int}^{\{7\}}, \mathbf{Bool}^{\{\mathbf{False}\}})^{\top} \\
[3, 2, 1] & : & [\mathbf{Int}^{\{1,2,3\}}]^{\{(\_:\_:\_:[])\}} \\
\lambda x.\ x + 1 & : & \mathbf{Int}^{\top} \xrightarrow{\top} \mathbf{Int}^{\top}
\end{array}
$$

Higher-Order Functions

Program

$$
\begin{aligned}
\mathit{main}\ b\ f = &\ \textbf{if}\ b\ \textbf{then} \\
&\quad \textbf{if}\ f\ 42\ \textbf{then}\ 100\ \textbf{else}\ 200 \\
&\ \textbf{else} \\
&\quad \textbf{if}\ f\ 43\ \textbf{then}\ 300\ \textbf{else}\ 400
\end{aligned}
$$

Higher-Order Functions

Program

$$\begin{aligned} main\ b\ f = \ & \textbf{if}\ b\ \textbf{then} \\ & \quad \textbf{if}\ f\ 42\ \textbf{then}\ 100\ \textbf{else}\ 200 \\ & \textbf{else} \\ & \quad \textbf{if}\ f\ 43\ \textbf{then}\ 300\ \textbf{else}\ 400 \end{aligned}$$

Type

$$\textbf{Bool} \to (\textbf{Int} \to \textbf{Bool}) \to \textbf{Int}$$

Higher-Order Functions

Program

$$
\begin{aligned}
main\ b\ f = {} & \textbf{if } b \textbf{ then} \\
& \quad \textbf{if } f\ 42 \textbf{ then } 100 \textbf{ else } 200 \\
& \textbf{else} \\
& \quad \textbf{if } f\ 43 \textbf{ then } 300 \textbf{ else } 400
\end{aligned}
$$

Type

$$
\mathbf{Bool}^{\{\mathsf{T},\mathsf{F}\}} \to (\mathbf{Int}^{\{42,43\}} \to \mathbf{Bool}^{\{\mathsf{T},\mathsf{F}\}})_{-} \to \mathbf{Int}^{\{100,200,300,400\}}
$$

Higher-Order Functions

Program

$$
\begin{aligned}
&\mathit{main}\ b\ f = \textbf{if}\ b\ \textbf{then} \\
&\qquad \textbf{if}\ f\ 42\ \textbf{then}\ 100\ \textbf{else}\ 200 \\
&\quad \textbf{else} \\
&\qquad \textbf{if}\ f\ 43\ \textbf{then}\ 300\ \textbf{else}\ 400
\end{aligned}
$$

Type

$$\mathbf{Bool}_{-}^{\{\mathsf{T},\mathsf{F}\}} \to (\mathbf{Int}_{+}^{\{42,43\}} \to \mathbf{Bool}_{-}^{\{\mathsf{T},\mathsf{F}\}})_{-} \to \mathbf{Int}_{+}^{\{100,200,300,400\}}$$

Higher-Order Functions

Program

$$\begin{aligned} \mathit{main}\ b\ f = \ &\textbf{if}\ b\ \textbf{then} \\ &\quad \textbf{if}\ f\ 42\ \textbf{then}\ 100\ \textbf{else}\ 200 \\ &\textbf{else} \\ &\quad \textbf{if}\ f\ 43\ \textbf{then}\ 300\ \textbf{else}\ 400 \end{aligned}$$

Type

$$\mathbf{Bool}_{-}^{\{\mathbf{T}\}} \to (\mathbf{Int}_{+}^{\{41,42,43\}} \to \mathbf{Bool}_{-}^{\{\mathbf{F}\}})_{-} \to \mathbf{Int}_{+}^{\{100,200,300,400,500\}}$$

Integers

$$\text{Sign} ::= + \mid 0 \mid -$$

Integers

$$\text{Sign} ::= + \mid 0 \mid -$$

$$\text{Parity} ::= \text{Even} \mid \text{Odd}$$

$$\ldots$$

Lists

$$\text{Shape} ::= \mathbf{[]} \mid \mathbf{(}\ \_\ \mathbf{:}\text{Shape}\mathbf{)} \mid \star$$

Lists

$$\text{Shape} ::= \mathbf{[]} \mid \mathbf{(}\,\_\,\mathbf{:}\text{Shape}\mathbf{)} \mid \star$$

$$\top_{\text{Shape}} = \{\mathbf{[]}, \mathbf{(}\,\_\,\mathbf{:[])}, \mathbf{(}\,\_\,\mathbf{:(}\,\_\,\mathbf{:[]))}, \mathbf{(}\,\_\,\mathbf{:(}\,\_\,\mathbf{:}\star\mathbf{))}\}$$

Overview

1. Generate constraints
2. Solve constraints
3. ???
4. Profit!

Typing relation

Relation

$$\widehat{\Gamma} \vdash e : \widehat{\tau} \leadsto C \ \& \ R$$

Legend

$\widehat{\Gamma}$	Annotated type environment
e	Expression being typed
$\widehat{\tau}$	Type of the expression
C	Equality constraints (e.g. $\alpha = \mathbf{Bool}^{\varphi} \to \mathbf{Bool}^{\psi}$)
R	Subset constraints (e.g. $\{[\,], (\_ : \varphi)\} \subseteq \psi$)

Constructing Values

$$\frac{\beta_1, \beta_2 \text{ fresh} \quad \widehat{\Gamma} \vdash e_1 : \alpha_1 \rightsquigarrow C_1 \;\&\; R_1 \qquad \widehat{\Gamma} \vdash e_2 : \alpha_2 \rightsquigarrow C_2 \;\&\; R_2 \quad C = C_1 \cup C_2 \cup \{\alpha_2 = [\alpha_1]^{\beta_2}\} \quad R = R_1 \cup R_2 \cup \{(\_ : \beta_2) \subseteq \beta_1\}}{\widehat{\Gamma} \vdash (e_1 : e_2) : [\alpha_1]^{\beta_1} \rightsquigarrow C \;\&\; R} \text{[T-Cons]}$$

Pattern Matching

$$\dfrac{\begin{array}{c}\widehat{\Gamma} \vdash g : \widehat{\tau}_{\mathrm{g}} \leadsto C_{\mathrm{g}} \;\&\; R_{\mathrm{g}} \qquad\qquad \beta \text{ fresh} \\ \widehat{\Gamma} \vdash e_1 : \widehat{\tau}_1 \leadsto C_1 \;\&\; R_1 \qquad\qquad \widehat{\Gamma} \vdash e_2 : \widehat{\tau}_2 \leadsto C_2 \;\&\; R_2 \\ C = C_{\mathrm{g}} \cup C_1 \cup C_2 \cup \{\widehat{\tau}_{\mathrm{g}} = \mathbf{Bool}^{\beta}, \widehat{\tau}_1 = \widehat{\tau}_2\} \\ R = R_{\mathrm{g}} \cup R_1 \cup R_2 \cup \{\beta \subseteq \{\mathbf{True}, \mathbf{False}\}\}\end{array}}{\widehat{\Gamma} \vdash \mathbf{if}\ g\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 : \widehat{\tau}_1 \leadsto C \;\&\; R} \;[\text{T-If}]$$

Overview

- Solve C using unification
  - Includes unifying annotation variables
  - Apply resulting substitution θ to $\widehat{\tau}$ and R
- Solve R using worklist algorithm
  - Do dependency analysis
  - Determine input-independent $R' \subseteq R$
  - Solve R' using worklist algorithm
    - Determines lowerbound L and upperbound U for all β
  - Check for pattern match failures ($L \subseteq U$)
- Generalize over $\mathrm{ftv}(\widehat{\tau}) - \mathrm{ftv}(\widehat{\Gamma})$ and $R - R'$

Example

Intermediate result

$$
\begin{aligned}
\beta_1 &= (\{\mathbf{[]}, (\_\,\mathbf{:}\mathbf{[]}), (\_\,\mathbf{:}(\_\,\mathbf{:}\mathbf{[]}))\}, \top) \\
\beta_2 &= (\bot, \top)
\end{aligned}
$$

Example

Intermediate result

$$\begin{aligned}\beta_1 &= (\{[\,], (\_:[\,]), (\_:(\_:[\,]))\}, \top) \\ \beta_2 &= (\bot, \top)\end{aligned}$$

Constraint

$$\beta_1 \subseteq \{(\_:\beta_2)\}$$

Example

Intermediate result

$$\begin{aligned}\beta_1 &= (\{[], (\_:[]), (\_:(\_:[]))\}, \top) \\ \beta_2 &= (\bot, \top)\end{aligned}$$

Constraint

$$\beta_1 \subseteq \{(\_:\beta_2)\}$$

Substitute LHS

$$\{[], (\_:(\_:[])), (\_:(\_:(\_:[])))\} \subseteq \{(\_:\beta_2)\}$$

Example

Intermediate result

$$\begin{aligned} \beta_1 &= (\{[], (\_:[]), (\_:(\_:[]))\}, \top) \\ \beta_2 &= (\bot, \top) \end{aligned}$$

Constraint

$$\beta_1 \subseteq \{(\_:\beta_2)\}$$

Substitute LHS

$$\{[], (\_:(\_:[])), (\_:(\_:(\_:[])))\} \subseteq \{(\_:\beta_2)\}$$

Project out fields

$$\begin{aligned} \{[]\} &\subseteq \emptyset \\ \{(\_:[]), (\_:(\_:[]))\} &\subseteq \beta_2 \end{aligned}$$

Example

Intermediate result

$$\begin{aligned}\beta_1 &= (\{[], (\_:[]), (\_:(\_:[]))\}, \top) \\ \beta_2 &= (\bot, \top)\end{aligned}$$

Project out fields

$$\begin{aligned}\{[]\} &\subseteq \emptyset \\ \{(\_:[]), (\_:(\_:[]))\} &\subseteq \beta_2\end{aligned}$$

Update intermediate results

$$\begin{aligned}L(\beta_2) &:= L(\beta_2) \sqcup \{(\_:[]), (\_:(\_:[]))\} \\ &= \{(\_:[]), (\_:(\_:[]))\}\end{aligned}$$

Conclusions

So, does it work?

Further Research

Sometimes, but there's a lot of room for improvement.

- Unnatural type for *map*
- Principal types for operators