On Duality in Functional Programming

Härmel Nestra

Institute of Computer Science

University of Tartu

e-mail: harmel.nestra@ut.ee

Note

THIS PRESENTATION IS PROVIDED “AS IS”. THE AUTHOR IS IN NO WAY LIABLE FOR ANY KIND OF INCONVENIENCES CAUSED BY USING THE IDEAS OF THIS PRESENTATION IN PRACTICE.

Outline

Sorry for the too abstract abstract.

1. Case-expression and record expression.
2. Pattern guard construction.
3. Pattern guards categorically.
4. Categorical dualization of pattern guards.

Case-expression and record expression

Case-expression syntax

< *case-expression* > → **`case`** < *expression* > **`of`**
< *pattern* >
`->` < *expression* >
. . . . . .
< *pattern* >
`->` < *expression* >

Record expression syntax

< *record expression* > → < *constructor* >
{
< *selector* >
= < *expression* >**,**
. . . . . .
< *selector* >
= < *expression* >**,**
}

Case-expression example

```
dropWhile p xs
  = case xs of
      z : zs
        -> if p z then dropWhile p zs else xs
      _
        -> []
```

Record expression example

```
toComplex x
  = (:+)
      {
        realPart
          = x,
        imagPart
          = 0,
      }
```

Categorical explanation of the duality

Let $(A_i \mid i \in I)$ a family of objects and C an object.

- A one-to-one correspondence:

$$\begin{array}{lcl} \sum_{i\in I} A_i \to C & \longleftrightarrow & \prod_{i\in I}(A_i \to C), \\ f & \longmapsto & (\mathrm{in}_i\,;f \mid i \in I)\,, \\ \bigtriangledown g & \longmapsfrom & g = (g_i \mid i \in I)\,. \end{array}$$

  Case-construction corresponds to operator $\bigtriangledown$.

- Dual one-to-one correspondence:

$$\begin{array}{lcl} C \to \prod_{i\in I} A_i & \longleftrightarrow & \prod_{i\in I}(C \to A_i), \\ f & \longmapsto & (f\,;\mathrm{ex}_i \mid i \in I)\,, \\ \bigtriangleup g & \longmapsfrom & g = (g_i \mid i \in I)\,. \end{array}$$

  Record construction corresponds to operator $\bigtriangleup$.

Pattern guard construction

Syntax

$$
\begin{array}{lcll}
< \textit{pattern guard} > & \rightarrow & < \textit{qualifier} >\texttt{,} \quad \ldots \texttt{,} & < \textit{qualifier} > \\
< \textit{qualifier} > & \rightarrow & < \textit{generator} > & \\
 & \mid & < \textit{guard} > &
\end{array}
$$

- Usage of pattern guards is similar to the Standard Haskell usage of guards.
- The traditional guard is obtained if the list of qualifiers consists of one qualifier which is a guard.
- Implemented in GHC (not in Hugs).

Semantics

The general idea is pattern matching together with backtracking.

- Guard is semantically a special case of generator.
  - Guard 𝔤 is equivalent to generator `True` **<-** 𝔤.
  - Thus one may assume every qualifier is a generator.
- The generators in one pattern guard are read from left to right.
- For generator 𝔭 **<-** 𝔢, pattern 𝔭 is matched against expression 𝔢.

  If this succeeds then the next generator of the same pattern guard is studied.

  Otherwise the whole pattern guard is abandoned, the bindings introduced by them are forgotten.
- The rhs corresponding to the first pattern guard whose all pattern matches succeed is selected.

  If no pattern guard satisfies this condition then a backtracking is performed.

Example

The most natural definition of `dropWhile` is obtained with help of pattern guards:

```
dropWhile p xs
  | z : zs <- xs, p z
    = dropWhile p zs
  | otherwise
    = xs
```

Special cases

Pattern guard construction generalizes both guard construction and case-expression.

In both cases, it suffices to have one qualifier in each pattern guard.

- Case-expression is obtained when all pattern guards consist of one generator and all rhss of the generators are (syntactically) equal.
  - This correspondence is not total since case-expression does not enable backtracking. However, this is not an important difference in our theoretical study.
- Guard construction is obtained when all pattern guards consist of one generator where the patterns of the generators are equal to `True`.

Pattern guards is what the programmer actually needs!

Pattern guards categorically

Assumption

Assume for simplicity that each pattern guard consists of one generator.

Elements of pattern guard construciton

The programmer determines explicitly:

- a set I whose elements correspond to cases;
- for every $i \in I$, a sum type $\sum_{j \in J_i} X_{ij}$ to which the rhs of the ith generator belongs;
- for every $i \in I$, a function $f_i : A \to \sum_{j \in J_i} X_{ij}$ which is the rhs of the ith generator;
- for every $i \in I$, index $j_i \in J_i$, corresponding to the addend of the sum type being selected by the lhs of the ith generator;
- for every $i \in I$, a function $g_i : X_{ij_i} \to B$ which is the rhs corresponding to the ith case.

Implicit categorical elements

In addition, we have

- injections $\iota_{j_i} : X_{ij_i} \to \sum_{j \in J_i} X_{ij}$;
- injections $\iota_i : X_{ij_i} \to \sum_{i \in I} X_{ij_i}$;
- a function $\bigtriangledown g : \sum_{i \in I} X_{ij_i} \to B$.

Function defined by pattern guard construction

A pattern guard construction determines:

- a function $\mu : A \to \sum_{i \in I} X_{ij_i}$ — making a decision;
- the final result $\mu \mathbin{;} \bigtriangledown g : A \to B$.

Choice function

The choice function $\mu : A \to \sum_{i \in I} X_{ij_i}$ categorically:

- for every $i \in I$, pair (f_i, ι_{j_i}) determines pullback square (α_i, ξ_i) with $\alpha_i : K_i \to A$ and $\xi_i : K_i \to X_{ij_i}$;
- μ satisfies

$$\forall i \in I \; (\alpha_i \; ; \mu = \xi_i \; ; \iota_i).$$

Pullback square

A pullback square corresponding to pair of functions $f : A \to C$ and $g : B \to C$ is a set $K = \{a \in A, b \in B \mid f(a) = g(b) \bullet (a, b)\}$ together with projection functions to A and B.

If g is injective then, for each a, there exists at most one b such that $f(a) = g(b)$:

- K degenerates to a subset of A,
- projection to A forms a natural injection,
- projection to B is a function which, for every a, returns a b such that $f(a) = g(b)$.

Our case

In our case:

- K_i is a part of type A, consisting of values in case of which the ith guard is chosen;
- α_i is the natural injection;
- ξ_i is a function which, for every value from K_i, returns a corresponding value from X_{ij_i}, i.e., the result of pattern matching.

Categorical dualization of pattern guards

Construction elements

The programmer must determine:

- a set I whose elements correspond to construction elements;
- for every $i \in I$, a product type $\prod_{j \in J_i} X_{ij}$ which is the domain of the rhs of the ith element (this rhs could be called coguard);
- for every $i \in I$, a function $f_i : \prod_{j \in J_i} X_{ij} \to A$ which is the rhs, i.e., the coguard;
- for every $i \in I$, an index $j_i \in J_i$ corresponding to the factor of the product type to which the lhs of the ith element of the construction belongs;
- for every $i \in I$, a function $g_i : B \to X_{ij_i}$ — the lhs.

Note that the lhss and rhss are interchanged w.r.t. pattern guard construction.

Implicit categorical elements

Additionally, we have

- projections $\pi_{j_i} : \prod_{j \in J_i} X_{ij} \to X_{ij_i}$;
- projections $\pi_i : \prod_{i \in I} X_{ij_i} \to X_{ij_i}$;
- function $\bigwedge g : B \to \prod_{i \in I} X_{ij_i}$.

Function under definition

The new construction should determine:

- function $\nu : \prod_{i \in I} X_{ij_i} \to A$ which could be called join function;
- the final result $\bigwedge g \mathbin{;} \nu : B \to A$.

Join function

Join function $\nu : \prod_{i \in I} X_{ij_i} \to A$ categorically:

- for every $i \in I$, pair (f_i, π_{j_i}) determines pushout square (β_i, η_i) with $\beta_i : A \to R_i$ and $\eta_i : X_{ij_i} \to R_i$;
- ν satisfies

$$\forall i \in I \, (\nu \, ; \beta_i = \pi_i \, ; \eta_i).$$

Pushout square

The pushout square corresponding to the pair of functions $f : C \to A$ and $g : C \to B$ is the factor set R of set $A + B$ w.r.t. the equivalence induced by relation $\{c \in C \mid \bullet(f(c), g(c))\}$, together with injections from A and B.

If g is surjective then, for each b, there exists an a such that

$$(a, b) \in \{c \in C \mid \bullet(f(c), g(c))\} \subseteq (A + B)/R\text{:}$$

- R degenerates to a factor set of A,
- the injection from A changes to natural surjection;
- the injection from B turns to the function which, for every b, returns the equivalence class containing that a in the case of which

$$(a, b) \in \{c \in C \mid \bullet(f(c), g(c))\}\,.$$

Our case

Call the factor set classification.

- R_i is a classification on type A which identifies at least for all $x \in X_{ij_i}$ all values $f_i(p)$ such that $\pi_{j_i}(p) = x$;
- β_i is the natural projection;
- η_i is the function which, for every $x \in X_{ij_i}$, returns the class containing values $f_i(p)$ such that $\pi_{j_i}(p) = x$;
- $\nu(q)$, where $q \in \prod_{i \in I} X_{ij_i}$, is such an $a \in A$ which, for every $i \in I$, belongs to the class of the ith classification that contains $f_i(p)$ for all p such that $\pi_{j_i}(p) = \pi_i(q)$.

Possible syntax

< *dual construction* >
→ **&** < *cogenerator* >
= < *expression* >
. . . . . .
& < *cogenerator* >
= < *expression* >

< *cogenerator* >
→ < *position* > < *simple pattern* > **<–** < *expression* >

Types

For every $i \in I$:

- the rhs of the ith construction element is of type A provided the variable defined in the simple pattern is of type $\prod_{j \in J_i} X_{ij}$;
- position in the ith cogenerator determines j_i;
- the rhs of the ith cogenerator is of type X_{ij_i}.

Semantics

All rhss are evaluated in order. The value of the construction is such a value which matches all rhss.

- The last operation is unification.
- If a contradiction arises between some rhss then evaluation of the whole construction fails.

Philosophical interpretation

Each cogenerator presents a fragment of some complex structure.

Each rhs presents partial information about some one and the same value in terms of the complex structure.

The task is to determine this one and the same value.

Special cases

If each rhs coincides with the variable bound in the lhs of the corresponding cogenerator then we in principle obtain a record expression.