Context

- **OCaml**:-
  - ML family programming language
  - combine functional, imperative and object-oriented aspects
- **Gospel** [1]:
  - formal specification language for OCaml
  - first-order logic
  - separation logic based semantics

Pattern matching

- idiomatic in functional languages, heavily used in OCaml
- pattern-based structural reasoning over algebraic data types

```haskell
type tree = E | N of tree * int * tree
let rec min = function
  | E -> None
  | N (E, x, _) -> Some x
  | N (l, _, _) -> min l
let compare = function
  | [], [] | E :: _, E :: _ -> 0
  | _, [] | N _ :: _, _ -> 1
  | [], _ | _, N _ :: _ -> -1
```

Problems

**Exhaustiveness**: are all cases considered?

A pattern matching **P** is exhaustive if and only if every possible (well typed) value is filtered by **P**. Thus, the function **h** is exhaustive, but **h'** is not.

```haskell
let rec h = function
  | E -> 0
  | N(E, _, _) -> 1
  | N(l, _, _) -> 1 + h l
let h' = function
  | E -> 0
  | N(E, _, _) -> 1
  | N(l, _, _) -> 1 + h l
```

**Redundancy**: is a pattern subsumed by the previous ones?

A pattern matching is redundant if and only if a line **i** is less general than a line **j** where **i < j**. Thus, the function **h** is redundant, but **h'** is not.

```haskell
let h = function
  | _ -> false
  | E -> true
let h' = function
  | _ -> false
  | E -> true
```

An algorithm

The usefulness algorithm, developed by Luc Maranget [2], solves both problems. A function

```haskell
is_useful : pat list → pat → bool
```

decides whether a pattern filters more values than a given list of patterns.

- **exhaustiveness**: is **x** useful to the whole pattern matching?
- **redundancy**: is every pattern useful to its predecessors?

Contributions

- termination and correctness proof
  - well-tested implementation into Gospel code base
  - with counter-example generation
  - with **when** clauses
- general purpose pattern matching generator

Proofs

- **Complexity**: proof that the execution time may be exponential in the number of lines of the pattern-matching.
- **Termination**: the hard part was finding the variant for **is_useful**, since or-patterns increase the size of the patterns in recursive calls.
- **Correctness**: by induction over the code of **is_useful**.

Implementation

- implementation of **is_useful** in Gospel
  - extensions:
    - handles **when** clauses
    - generates counter-examples
  - about 1,000 lines of code

Tests design

- Design and implementation of a highly customisable and randomised test generator [3].
- Consistency tests over 10,000 generated problems, which represent a total of ~ 500,000 lines of pattern-matchings.

References


Contact: paul.patault@universite-paris-saclay.fr