PATTERN MATCHING : EXHAUSTIVE TESTS FOR EXHAUSTIVENESS CHECK

Paul Patault

Univ. Paris-Saclay, Laboratoire Méthodes Formelles – *M1 internship supervised by Jean-Christophe Filliâtre*

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

  `pat ::= x | v | C (pat,...,pat) | (pat | pat) | ...`

```
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.

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

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

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.

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

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

An algorithm

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

`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 findind 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

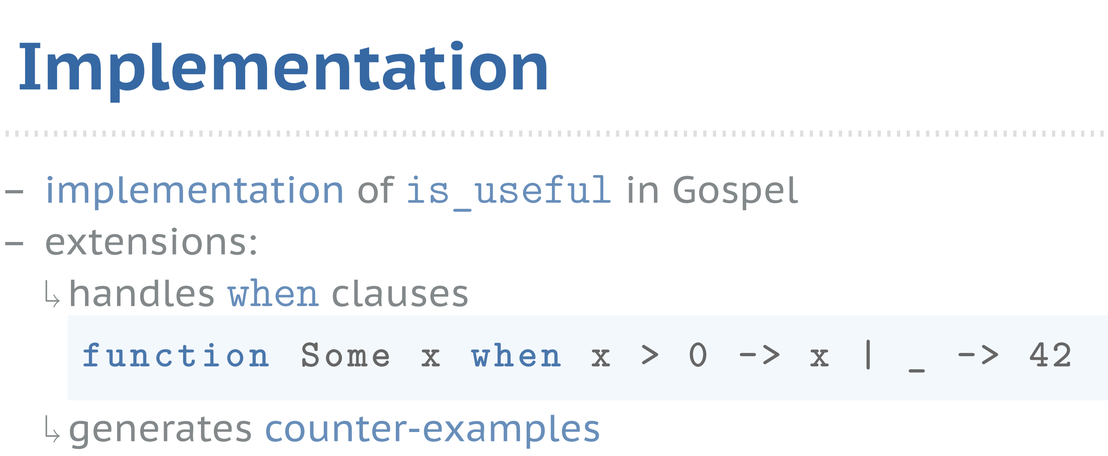
- implementation of `is_useful` in Gospel
- extensions:
  - handles `when` clauses

    ```
    function Some x when x > 0 -> x | _ -> 42
    ```

  - 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 $\sim$ 500,000 lines of pattern-matchings.



References

[1] Arthur Charguéraud et al. “GOSPEL—providing OCaml with a formal specification language”. In: *International Symposium on Formal Methods*. Springer. 2019, pp. 484–501.

[2] Luc Maranget. “Warnings for pattern matching”. In: *Journal of Functional Programming* 17.3 (2007), pp. 387–421.

[3] Paul Patault. *Minipat, a randomised pattern matching generator*. July 2022. url: `https://www.paulpatault.fr/minipat`.

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