ADVANCED PATTERNS (60/60 points)

Let's rewrite some pattern matching with advanced constructs.

1. Factorize the pattern matching of function `simplify` using or-patterns. It should boil down to three cases.
2. The `only_small_lists` function takes a list as input and returns this list only if it contains two or less elements, otherwise the empty list is returned. Rewrite this function using or-patterns and an `as` -pattern. It should boil down to two cases.
3. Turn the third case of `no_consecutive_repetition` into two distinct cases, dropping the `if` construct in favor of a `when` clause.

THE GIVEN PRELUDE

```
type e = EInt of int | EMul of e * e | EAdd of e * e
```

YOUR OCAML ENVIRONMENT

```
let simplify = function
  | EMul (EInt 0, e) | EMul (e, EInt 0) -> EInt 0
  | EMul (EInt 1, e) | EMul (e, EInt 1) | EAdd (EInt 0, e) | EAdd (e, EInt 0) | e-> e
;;

let only_small_lists = function
  | ([_;_]|[_]) as l -> l
  | ([] | _) -> []
;;

let rec no_consecutive_repetition = function
  | ([]|[_]) as l -> l
  | x :: y :: ys when x = y -> no_consecutive_repetition (y :: ys)
  | x :: y :: ys -> x :: (no_consecutive_repetition (y :: ys))
;;

```

Evaluate >
Switch >>
Typecheck
Reset Templ
Full-screen
Check & Sa

Exercise complete (click for details) — 60 pts

v Exercise 1: `simplify` — Completed, 20 pts

Found a toplevel definition for `simplify` .

You broke it down to three cases, bravo! — 5 pts

You used an or pattern, bravo! — 5 pts

Found `simplify` with compatible type.

Computing
```
simplify
  (EMul (EMul (EMul (EInt (-3), EInt (-4)), EInt (-4)),
    EAdd (EAdd (EInt 2, EInt 0), EAdd (EInt (-4), EInt 0))))
```
Correct value — 1 pt
```
(EMul (EMul (EMul (EInt (-3), EInt (-4)), EInt (-4)),
  EAdd (EAdd (EInt 2, EInt 0), EAdd (EInt (-4), EInt 0))))
```
Computing
```
simplify
  (EAdd (EAdd (EInt (-2), EMul (EInt 3, EInt (-1))),
    EAdd (EAdd (EInt 2, EInt (-1)), EMul (EInt 2, EInt 1))))
```
Correct value — 1 pt
```
(EAdd (EAdd (EInt (-2), EMul (EInt 3, EInt (-1))),
  EAdd (EAdd (EInt 2, EInt (-1)), EMul (EInt 2, EInt 1))))
```
Computing
```
simplify
  (EAdd (EAdd (EMul (EInt 0, EInt 1), EMul (EInt (-4), EInt 3)),
    EMul (EMul (EInt 4, EInt (-3)), EMul (EInt (-5), EInt 3))))
```
Correct value — 1 pt
```
(EAdd (EAdd (EMul (EInt 0, EInt 1), EMul (EInt (-4), EInt 3)),
  EMul (EMul (EInt 4, EInt (-3)), EMul (EInt (-5), EInt 3))))
```

Correct value `(EInt 0)` 1 pt

Computing `simplify (EAdd (EMul (EInt 3, EAdd (EInt (-3), EInt (-4))), EInt 4))`

Correct value `(EAdd (EMul (EInt 3, EAdd (EInt (-3), EInt (-4))), EInt 4))` 1 pt

Computing
```
simplify
  (EMul (EAdd (EInt (-2), EInt (-1)), EMul (EInt 0, EMul (EInt 1, EInt 3))))
```
Correct value 1 pt
```
(EMul (EAdd (EInt (-2), EInt (-1)), EMul (EInt 0, EMul (EInt 1, EInt 3))))
```
Computing
```
simplify
  (EAdd (EAdd (EMul (EInt (-4), EInt 1), EInt (-2)),
    EMul (EInt 3, EInt (-4))))
```
Correct value 1 pt
```
(EAdd (EAdd (EMul (EInt (-4), EInt 1), EInt (-2)), EMul (EInt 3, EInt (-4))))
```
Computing `simplify (EInt 1)`

Correct value `(EInt 1)` 1 pt

Computing
```
simplify
  (EMul (EMul (EInt 4, EAdd (EInt (-3), EInt (-5))),
    EMul (EInt (-4), EMul (EInt (-5), EInt (-5)))))
```
Correct value 1 pt
```
  (EMul (EMul (EInt 4, EAdd (EInt (-3), EInt (-5))),
    EMul (EInt (-4), EMul (EInt (-5), EInt (-5)))))
```

v Exercise 2: `only_small_lists` Completed, 25 pts

Found a toplevel definition for `only_small_lists` .

You broke it down to two cases, bravo! 5 pts

You used an as pattern, bravo! 5 pts

You used an or pattern, bravo! 5 pts

Now I will check that it behaves correctly

Found `only_small_lists` with compatible type.

Computing `only_small_lists ['y'; 'l'; 'u'; 'e']`

Correct value `[]` 1 pt

Computing `only_small_lists ['z'; 'o'; 'h'; 'u'; 'm'; 'k'; 'a']`

Correct value `[]` 1 pt

Computing `only_small_lists []`

Correct value `[]` 1 pt

Computing `only_small_lists ['u'; 'q']`

Correct value `['u'; 'q']` 1 pt

Computing `only_small_lists ['v'; 'h'; 'h'; 'h'; 'g'; 'i'; 'w'; 'a'; 'y']`

Correct value `[]` 1 pt

Found `only_small_lists` with compatible type.

Computing `only_small_lists [4; -5; 0]`

Correct value `[]` 1 pt

Computing `only_small_lists [-2; 0; -3]`

Correct value `[]` 1 pt

Computing `only_small_lists []`

Correct value `[]` 1 pt

Computing `only_small_lists [1; -5; 0; 4; -4; 0]`

Correct value `[]` 1 pt

Computing `only_small_lists [-5; -1; -3]`

Correct value `[]` 1 pt

v Exercise 3: `no_consecutive_repetition` Completed, 15 pts

Found a toplevel definition for `no_consecutive_repetition` .

You used when clause, bravo! 5 pts

Now I will check that it behaves correctly

Found `no_consecutive_repetition` with compatible type.

Computing `no_consecutive_repetition ['a'; 'c'; 'b'; 'b'; 'c'; 'a'; 'b']`

Correct value `['a'; 'c'; 'b'; 'c'; 'a'; 'b']` 1 pt

Computing `no_consecutive_repetition ['c'; 'b'; 'c'; 'a'; 'a'; 'b'; 'a'; 'b'; 'c']`

Correct value `['c'; 'b'; 'c'; 'a'; 'b'; 'a'; 'b'; 'c']` 1 pt

Computing `no_consecutive_repetition ['c'; 'b'; 'b'; 'b']`

Correct value `['c'; 'b']` 1 pt

Computing `no_consecutive_repetition ['a'; 'b'; 'b'; 'b'; 'a'; 'c'; 'c'; 'a']`

Correct value `['a'; 'b'; 'a'; 'c'; 'a']` 1 pt

Computing `no_consecutive_repetition []`

Correct value `[]` 1 pt

Found `no_consecutive_repetition` with compatible type.

Correct value [1; 2; 1; 2; 0] 1 pt
Computing no_consecutive_repetition [0; 1; 1; 0; 2; 2; 2]
Correct value [0; 1; 0; 2] 1 pt
Computing no_consecutive_repetition [0; 2; 0]
Correct value [0; 2; 0] 1 pt
Computing no_consecutive_repetition [2; 2; 0]
Correct value [2; 0] 1 pt