```
type binop = Add | Sub | Mul | Div
type var = int
type expr =
  | Const of int
  | Var of var
  | Binop of binop * expr * expr
  | Letin of var * expr * expr
  | Ifzero of expr * expr * expr
  | Call of string * expr
  | Raise of expr
  | TryWith of expr * var * expr
type def = { name : string; locals: int; body : expr; }
type program = { funs : def list; print : expr; }


let rec expr h = function
  | Const _ | Var _ -> false
  | Binop (_, e1, e2)
  | Letin (_, e1, e2) -> expr h e1 || expr h e2
  | Ifzero (e1, e2, e3) -> expr h e1 || expr h e2 || expr h e3
  | Call (f, e) -> h f || expr h e
  | Raise _ -> true
  | TryWith (e1, _, e2) -> expr h e1 && expr h e2

let fixpoint p =
  let r = Hashtbl.create 17 in
  let raise_exn f = Hashtbl.find r f in
  List.iter (fun f -> Hashtbl.add r f.name false) p.funs;
  let rec fix () =
    let fixpoint_reached = ref true in
    List.iter
      (fun f ->
        let b = expr raise_exn f.body in
        if not (raise_exn f.name) && b then begin
          fixpoint_reached := false;
          Hashtbl.add r f.name true
        end)
      p.funs;
    if not !fixpoint_reached then fix ()
  in
  fix ();
  raise_exn
```