Typage

Exercice 1 :

```
module AstType =
struct

(* Opérateurs binaires existants dans Rat - résolution de la surcharge *)
type binaire = Fraction | PlusInt | PlusRat | MultInt | MultRat | EquInt |
EquBool | Inf

(* Expressions existantes dans Rat *)
(* = expression de AstTds *)
type expression =
  | AppelFonction of Tds.info_ast * expression list
  | Ident of Tds.info_ast
  | Booleen of bool
  | Entier of int
  | Unaire of AstSyntax.unaire * expression
  | Binaire of binaire * expression * expression

(* instructions existantes Rat *)
(* = instruction de AstTds + informations associées aux identificateurs,
mises à jour *)
(* + résolution de la surcharge de l'affichage *)
type bloc = instruction list
 and instruction =
  | Declaration of Tds.info_ast * expression
  | Affectation of Tds.info_ast * expression
  | AffichageInt of expression
  | AffichageRat of expression
  | AffichageBool of expression
  | Conditionnelle of expression * bloc * bloc
  | TantQue of expression * bloc
  | Retour of expression
  | Empty (* les nœuds ayant disparus: Const *)

(* informations associées à l'identificateur (dont son nom), liste des
paramètres, corps *)
type fonction = Fonction of Tds.info_ast * Tds.info_ast list * bloc

(* Structure d'un programme dans notre langage *)
type programme = Programme of fonction list * bloc

let taille_variables_declarees i =
  match i with
  | Declaration (info,_) ->
    begin
    match Tds.info_ast_to_info info with
    | InfoVar (_,t,_,_) -> getTaille t
```

```
    | _ -> failwith "internal error"
    end
  | _ -> 0 ;;

end
```

Exercice 2 :

```
let rec analyse_type_expression e =
  match e with
  | AstTds.AppelFonction(ia, listExp) ->
  | AstTds.Ident(ia) ->
  | AstTds.Booleen(bool) ->
  | AstTds.Entier(int) ->
  | AstTds.Unaire(un, expr) ->
  | AstTds.Binaire(bin, expression1, expression2) ->
    let (ne1, te1) = analyse_type_expression expression1 in
    let (ne2, te2) = analyse_type_expression expression2 in
      if (te1 != te2) then
        raise (TypeBinaireInattendu (bin, te1, te2))
      else
        match bin with
        | Fraction ->
        | Plus ->
        | Mult ->
        | Equ ->
        | Inf ->

let rec analyse_type_instruction tf i =
  match i with
  | AstTds.Declaration (t, ia, e) ->
  | AstTds.Affectation (ia, e) ->
  | AstTds.Affichage e ->
  | AstTds.Conditionnelle (c,t,e) ->
  | AstTds.TantQue (c,b) ->
  | AstTds.Retour (e) ->
  | AstTds.Empty ->
    AstType.Empty

and analyse_type_bloc tf li =

let analyse_type_fonction (AstTds.Fonction(t,ia,lp,li))  =

let analyser (AstTds.Programme (fonctions,prog)) =
  let nf = List.map analyse_type_fonction fonctions in
  let nb = analyse_type_bloc None prog in
  Programme (nf,nb)
```