```
(* ***************** *)
(*   Test Chapitre 2  *)
(* Graignic Guillaume *)
(*  Cadoret Olivier   *)
(* ***************** *)

(* Pour tester on lance analyse caractere avec la première régle de la grammaire S et une liste
possible d'unité lexicale sans oublié l'unité lexicale UL_EOF à la fin
   On récupère l'arbre concret construit et une liste vide *)

(*************************************)
(* Test 1 : Ici on test : t < y et x = y *)
let (arbConc,listeVide) =  analyse_caractere(Non_term S,[UL_IDENT "t";UL_INF;UL_IDENT
"y";UL_ET;UL_IDENT "x";UL_EGAL;UL_IDENT "y";UL_EOF]);;
(*
val arbConc : arbre_concret =
  ACNT (S,
   [ACNT (Expr,
     [ACNT (Termb,
       [ACNT (Facteurb,
         [ACNT (Relation,
           [ACT (UL_IDENT "t"); ACNT (Op, [ACT UL_INF]); ACT (UL_IDENT "y")])]);
        ACNT (SuiteTermb,
         [ACT UL_ET;
          ACNT (Termb,
           [ACNT (Facteurb,
             [ACNT (Relation,
               [ACT (UL_IDENT "x"); ACNT (Op, [ACT UL_EGAL]);
                ACT (UL_IDENT "y")])]);
            ACNT (SuiteTermb, [])])])]);
      ACNT (SuiteExpr, [])]);
    ACT UL_EOF])
val listeVide : unite_lexicale list = []
*)

(*************************************)
(* Test 2 : Ici on test : si a = b alors c = d sinon a = d fsi *)
let (arbConc,listeVide) =  analyse_caractere(Non_term S,[UL_SI;UL_IDENT "a";UL_EGAL;UL_IDENT
"b";UL_ALORS;UL_IDENT "c";UL_EGAL;UL_IDENT "d";UL_SINON;UL_IDENT "a";UL_EGAL;UL_IDENT
"d";UL_FSI;UL_EOF]);;
(*
val construit_arbre_abstrait : arbre_concret -> arbre_abstrait = <fun>
#   *         val arbConc : arbre_concret =
  ACNT (S,
   [ACNT (Expr,
     [ACNT (Termb,
       [ACNT (Facteurb,
         [ACT UL_SI;
          ACNT (Expr,
           [ACNT (Termb,
             [ACNT (Facteurb,
               [ACNT (Relation,
                 [ACT (UL_IDENT "a"); ACNT (Op, [ACT UL_EGAL]);
                  ACT (UL_IDENT "b")])]);
              ACNT (SuiteTermb, [])]);
            ACNT (SuiteExpr, [])]);
          ACT UL_ALORS;
          ACNT (Expr,
           [ACNT (Termb,
             [ACNT (Facteurb,
               [ACNT (Relation,
                 [ACT (UL_IDENT "c"); ACNT (Op, [ACT UL_EGAL]);
                  ACT (UL_IDENT "d")])]);
              ACNT (SuiteTermb, [])]);
            ACNT (SuiteExpr, [])]);
          ACT UL_SINON;
          ACNT (Expr,
           [ACNT (Termb,
             [ACNT (Facteurb,
               [ACNT (Relation,
                 [ACT (UL_IDENT "a"); ACNT (Op, [ACT UL_EGAL]);
                  ACT (UL_IDENT "d")])]);
```

```
              ACNT (SuiteTermb, [])]);
            ACNT (SuiteExpr, [])]);
          ACT UL_FSI]);
        ACNT (SuiteTermb, [])]);
      ACNT (SuiteExpr, [])]);
    ACT UL_EOF])
val listeVide : unite_lexicale list = []
*)

(*************************************)
(* Test 3 : Si on oubli, par exemple, le fsi, le test plante : si a = b alors c = d sinon a = d*)
let (arbConc,listeVide) =  analyse_caractere(Non_term S,[UL_SI;UL_IDENT "a";UL_EGAL;UL_IDENT
"b";UL_ALORS;UL_IDENT "c";UL_EGAL;UL_IDENT "d";UL_SINON;UL_IDENT "a";UL_EGAL;UL_IDENT "d";UL_EOF]);;
(*
val construit_arbre_abstrait : arbre_concret -> arbre_abstrait = <fun>
#   *         Exception: Failure "hd".
#
*)

(*************************************)
(* Test 4 : Ici on test : (a <= b) ou a = c *)
let (arbConc,listeVide) =  analyse_caractere(Non_term S,[UL_OUVR;UL_IDENT "a";UL_INFEGAL;UL_IDENT
"b";UL_FERM;UL_OU;UL_IDENT "a";UL_EGAL;UL_IDENT "c";UL_EOF]);;
(*
#   *         val arbConc : arbre_concret =
  ACNT (S,
   [ACNT (Expr,
     [ACNT (Termb,
       [ACNT (Facteurb,
         [ACT UL_OUVR;
          ACNT (Expr,
           [ACNT (Termb,
             [ACNT (Facteurb,
               [ACNT (Relation,
                 [ACT (UL_IDENT "a"); ACNT (Op, [ACT UL_INFEGAL]);
                  ACT (UL_IDENT "b")])])]);
            ACNT (SuiteTermb, [])]);
          ACNT (SuiteExpr, [])]);
         ACT UL_FERM]);
        ACNT (SuiteTermb, [])]);
      ACNT (SuiteExpr,
       [ACT UL_OU;
        ACNT (Expr,
         [ACNT (Termb,
           [ACNT (Facteurb,
             [ACNT (Relation,
               [ACT (UL_IDENT "a"); ACNT (Op, [ACT UL_EGAL]);
                ACT (UL_IDENT "c")])])]);
          ACNT (SuiteTermb, [])]);
        ACNT (SuiteExpr, [])])])]);
    ACT UL_EOF])
val listeVide : unite_lexicale list = []
*)

(*************************************)
(* Test 5 : On test la transformation d'un arbre concret en un arbre abstrait
   Ici on lui passe l'arbre concret du test 3 *)
let arbAbstr = construit_arbre_abstrait(arbConc);;
(*
#     val arbAbstr : arbre_abstrait =
  Cond (Comp ("a", UL_EGAL, "b"), Comp ("c", UL_EGAL, "d"),
   Comp ("a", UL_EGAL, "d"))
*)

(*************************************)
(* Test 6 : On test la transformation d'un arbre concret en un arbre abstrait
   Ici on lui passe l'arbre concret du test 4 *)
let arbAbstr = construit_arbre_abstrait(arbConc);;
(*
#     val arbAbstr : arbre_abstrait =
  Ou (Comp ("a", UL_INFEGAL, "b"), Comp ("a", UL_EGAL, "c"))
*)
```