TD3.1 TdL

Traduction des langages

Typage

Objectif :
- Définir la nouvelle structure d'arbre obtenue après la passe de typage
- Définir les actions à réaliser par la passe de typage

1 Rappel : le typage du langage Rat

1.1 Grammaire du langage RAT

1. $PROG' \rightarrow PROG\$$
2. $PROG \rightarrow FUN\ PROG$
3. $FUN \rightarrow TYPE\ id\ (\ DP\ )\ \{IS\ return\ E\ ;\}$
4. $PROG \rightarrow\ id\ \ BLOC$
5. $BLOC \rightarrow\ \{\ IS\ \}$
6. $IS \rightarrow\ I\ \ IS$
7. $IS \rightarrow \Lambda$
8. $I \rightarrow\ TYPE\ \ id\ =\ E\ ;$
9. $I \rightarrow\ id\ =\ E\ ;$
10. $I \rightarrow\ const\ \ id\ =\ entier\ ;$
11. $I \rightarrow\ print\ \ E;$
12. $I \rightarrow\ if\ \ E\ \ BLOC\ \ else\ \ BLOC$
13. $I \rightarrow\ while\ \ E\ \ BLOC$
14. $DP \rightarrow \Lambda$
15. $DP \rightarrow TYPE\ id\ DP$
16. $TYPE \rightarrow\ bool$
17. $TYPE \rightarrow\ int$
18. $TYPE \rightarrow\ rat$
19. $E \rightarrow\ call\ id\ (\ CP\ )$
20. $CP \rightarrow \Lambda$
21. $CP \rightarrow E\ CP$
22. $E \rightarrow\ [\ E\ /\ E\ ]$
23. $E \rightarrow\ num\ \ E$
24. $E \rightarrow\ denom\ \ E$
25. $E \rightarrow\ id$
26. $E \rightarrow\ true$
27. $E \rightarrow\ false$
28. $E \rightarrow\ entier$
29. $E \rightarrow\ (\ E\ +\ \ E\ )$
30. $E \rightarrow\ (\ E\ *\ \ E\ )$
31. $E \rightarrow\ (\ E\ =\ \ E\ )$
32. $E \rightarrow\ (\ E\ <\ \ E\ )$

1.2 Jugements de typage du langage RAT

Axiomes

- $\sigma :: \{x : \tau\} \vdash x : \tau$
- $\sigma \vdash\ \ : void$
- $\dfrac{\sigma \vdash x : \tau}{\sigma :: \{y : \tau'\} \vdash x : \tau}$
- $\sigma \vdash true : bool$
- $\sigma \vdash false : bool$
- $\sigma \vdash entier : int$

Expression

- $\dfrac{\sigma \vdash E_1 : int \quad E_2 : int}{\sigma \vdash [\ E_1\ /\ E_2\ ] : rat}$
- $\dfrac{\sigma \vdash E : rat}{\sigma \vdash num\ E : int}$

— $\dfrac{\sigma \vdash E : rat}{\sigma \vdash denom\ E : int}$

— $\dfrac{\sigma \vdash E : \tau}{\sigma \vdash (E) : \tau}$

— $\dfrac{\sigma \vdash E_1 : int \quad E_2 : int}{\sigma \vdash (E_1 + E_2) : int}$

— $\dfrac{\sigma \vdash E_1 : rat \quad E_2 : rat}{\sigma \vdash (E_1 + E_2) : rat}$

— $\dfrac{\sigma \vdash E_1 : int \quad E_2 : int}{\sigma \vdash (E_1 * E_2) : int}$

— $\dfrac{\sigma \vdash E_1 : rat \quad E_2 : rat}{\sigma \vdash (E_1 * E_2) : rat}$

— $\dfrac{\sigma \vdash E_1 : int \quad E_2 : int}{\sigma \vdash (E_1 = E_2) : bool}$

— $\dfrac{\sigma \vdash E_1 : bool \quad E_2 : bool}{\sigma \vdash (E_1 = E_2) : bool}$

— $\dfrac{\sigma \vdash E_1 : int \quad E_2 : int}{\sigma \vdash (E_1 < E_2) : bool}$

— On se limitera à ces signatures.

Structures de contrôle

— $\dfrac{\sigma \vdash E : bool \quad \sigma \vdash BLOC_1 : void \quad \sigma \vdash BLOC_2 : void}{\sigma \vdash if\ E\ BLOC_1\ else\ BLOC_2 : void, \{\}}$

— $\dfrac{\sigma \vdash E : bool \quad \sigma \vdash BLOC : void}{\sigma \vdash while\ E\ BLOC : void, \{\}}$

Déclaration / affectation

— $\dfrac{\sigma \vdash TYPE : \tau_1 \quad \sigma \vdash E : \tau_2 \quad (estCompatible\ \tau_1\ \tau_2)}{\sigma \vdash TYPE\ id = E : void, \{id : \tau_1\}}$

— $\dfrac{\sigma \vdash id : \tau_1 \quad \sigma \vdash E : \tau_2 \quad (estCompatible\ \tau_1\ \tau_2)}{\sigma \vdash id = E : void, \{\}}$

Instructions

— $\dfrac{}{\sigma \vdash const\ \ id\ =\ entier : void, \{id : int\}}$

— $\dfrac{\sigma \vdash E : \tau}{\sigma \vdash print\ \ E : void, \{\}}$

Déclaration de fonction

— $\dfrac{A \quad B \quad C \quad D \quad E}{\sigma \vdash TYPE\ id\ (DP)\ \{IS\ return\ E;\} : void, \{id : \tau_2 \to \tau_1\}}$

— A : $\sigma \vdash TYPE : \tau_1$
— B : $\sigma \vdash DP : \tau_2, \sigma_p$
— C : $\sigma :: \sigma_p :: \{id : \tau_2 \to \tau_1\} \vdash IS : void, \sigma_l$
— D : $\sigma :: \sigma_p :: \sigma_l :: \{id : \tau_2 \to \tau_1\} \vdash E : \tau_3$
— E : $estCompatible\ \tau_1\ \tau_3$

— $\dfrac{\sigma \vdash TYPE : \tau_1 \quad \sigma :: \{id : \tau_1\} \vdash DP : \tau_2, \sigma_p}{\sigma \vdash TYPE\ id\ DP : \tau_1 \times \tau_2, \{id : \tau_1\} :: \sigma_p}$

— $\dfrac{\sigma \vdash TYPE : \tau_1}{\sigma \vdash TYPE\ id : \tau_1, \{id : \tau_1\}}$

Appel de fonction

$$\frac{\sigma \vdash id : \tau_1 \to \tau_2 \quad CP : \tau_3 \quad (estCompatible\ \tau_1\ \tau_3)}{\sigma \vdash call\ (id\ CP) : \tau_2}$$

$$\frac{\sigma \vdash E : \tau_1 \quad CP : \tau_2 \quad \tau_2 \neq void}{\sigma \vdash E\ CP : \tau_1 \times \tau_2}$$

$$\frac{\sigma \vdash E : \tau_1 \quad CP : void}{\sigma \vdash E\ CP : \tau_1}$$

Suite d'instructions

$$\frac{\sigma \vdash IS : void, \sigma'}{\sigma \vdash \{IS\} : void}$$

$$\frac{\sigma \vdash I : void, \sigma' \quad \sigma :: \sigma' \vdash IS : void, \sigma''}{\sigma \vdash I\ IS : void, \sigma' :: \sigma''}$$

Le programme

$$\frac{\sigma \vdash FUN : void, \sigma' \quad \sigma :: \sigma' \vdash PROG : void, \sigma''}{\sigma \vdash FUN\ PROG : void, \sigma' :: \sigma''}$$

$$\frac{\sigma \vdash BLOC : void}{\sigma \vdash id\ BLOC : void}$$

2 Passe de typage

Nous rappelons qu'un compilateur fonctionne par passes, chacune d'elle réalisant un traitement particulier (gestion des identifiants, typage, placement mémoire, génération de code,...). Chaque passe parcourt, et potentiellement modifie, l'AST.

La seconde passe est une passe de typage. C'est elle qui vérifie la conformité des types déclarés et associe aux identifiants leurs informations de type.

2.1 Structure de l'AST post passe de typage

La passe de typage réalise des vérifications de types qui nécessite une mise à jour des informations de types de identificateurs. Elle prépare également les passes suivantes, par exemple en choisissant la "version" de l'opérateur à utiliser en cas de surcharge des opérateurs.

▷ **Exercice 1** *Définir la structure de l'AST post passe de typage*

2.2 Actions à réaliser lors de la passe de résolution des identifiants

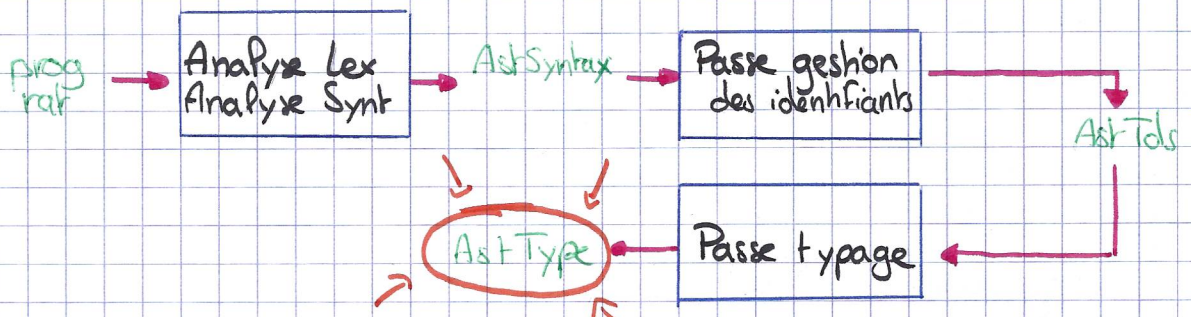
▷ **Exercice 2** *Définir les actions à réaliser lors de la passe de typage.*

TD3.2 TdL


```
(*************************************************)
(* AST après la phase d'analyse des identifiants *)
(*************************************************)
module AstTds =
struct

  (* Expressions existantes dans notre langage *)
  (* ~ expression de l'AST + informations associées aux identificateurs *)
  type expression =
    | AppelFonction of string * expression list * Tds.info_ast
    | Rationnel of expression * expression
    | Numerateur of expression
    | Denominateur of expression
    | Ident of Tds.info_ast
    | True
    | False
    | Entier of int
    | Binaire of AstSyntax.binaire * expression * expression

  (* instructions existantes dans notre langage *)
  (* = instruction de l'AST + informations associées aux identificateurs + suppr
ession de noeuds (const) *)
  type bloc = instruction list
  and instruction =
    | Declaration of typ * expression * Tds.info_ast
    | Affectation of expression * Tds.info_ast
    | Affichage of expression
    | Conditionnelle of expression * bloc * bloc
    | TantQue of expression * bloc
    | Empty (* les noeuds ayant disparus: Const *)


  (* Structure des fonctions dans notre langage *)
  (* type de retour, nom, liste des paramètres, corps, expression de retour, inf
ormations associées à l'identificateur *)
  type fonction = Fonction of typ * string * ( typ * Tds.info_ast ) list * bloc *
 expression * Tds.info_ast

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

end
```

TD3.1
TdL

Typage



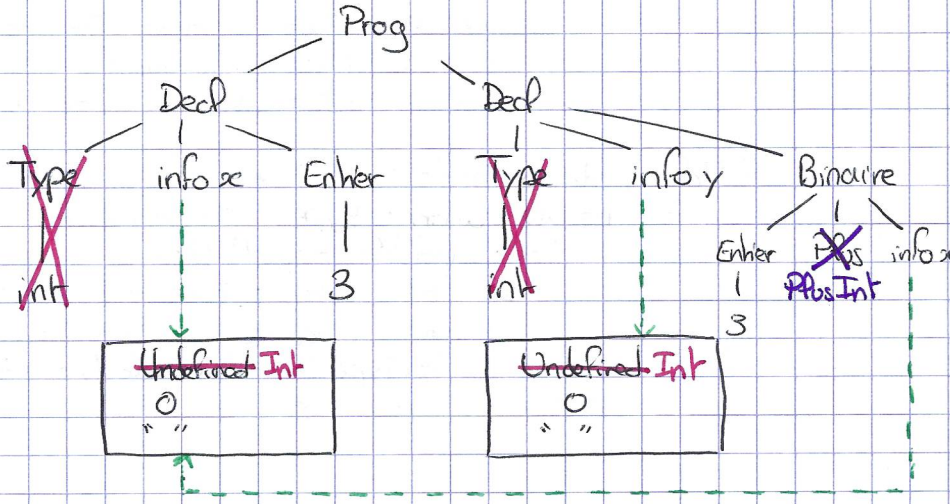
Exercice 1 : Structure de l'AST post typage

La passe de typage va :

- mettre à jour les informations lors de la déclaration des variables
- supprimer le "typ" dans l'ast.
- vérification de typage
- résolution de surcharge.

exemple : int x = 3,
int y = 3 + x,

----> pointeur



Résolution de la surcharge :

Affichage of expression → Affichage_Int of expression
Affichage_Rat of expression
Affichage_Bool of expression

type op = PPlus | MultI | Eq | Inf → type op = PPlusInt | PPlusRat | MultInt | MultRat | EqInt | EqRat | EqBool | Inf

Exercice 2 : Actions à réaliser lors de la passe de typage

```
(* AstTds.instruction -> AstType.instruction *)

let analyse_type_instruction i =
    match i with
    | AstTds.Declaration (t,e,info) ->
        mise_a_jour_type t info;
        let (ne,te) = analyse_type_expression e in
            if (estCompatible te t) then
                AstType.Declaration (ne, info)
            else raise ErreurType

    | AstTds.Affectation (e,info) ->
        let (ne,te) = analyse_type_expression e in
            if (estCompatible te (getType info)) then
                AstType.Affectation (ne, info)
            else raise ErreurType

    | AstTds.Affichage e ->
        let (ne,te) = analyse_type_expression e in
            (match te with
                | Int -> AstType.Affichage_Int ne
                | Rat -> AstType.Affichage_Rat ne
                | Bool -> AstType.Affichage_Bool ne)

    | AstTds.Conditionnelle (e,b1,b2) ->
        let (ne,te) = analyse_type_expression e in
            if (estCompatible te bool) then
                let nb1 = analyse_type_bloc b1 in
                let nb2 = analyse_type_bloc b2 in
                    AstType.Conditionnelle (ne, nb1, nb2)
            else raise ErreurType

    | AstTds.TantQue (e,b) ->
        let (ne,te) = analyse_type_expression e in
            if (estCompatible te bool) then
                let nb = analyse_type_bloc b in
                    AstType.TantQue (ne, nb)
            else raise ErreurType

    | Empty -> Empty
```

TD3.2
TdL

```
(* AstTds.expression -> (AstType.expression x AstType.typ) *)
let rec analyse_type_expression e =
  match e with
  | AstTds.Rationnel (e1, e2) ->
      let (ne1, t1) = analyse_type_expression e1 in
      let (ne2, t2) = analyse_type_expression e2 in
      if (t1 <> Int || t2 <> Int) then
        raise ErreurType
      else (AstTds.Rationnel (ne1, ne2), Rat)

  | AstTds.AppelFonction (s, el, info) ->
      let tr = getRetour info in
      let tp = getTypeParam info in
      let l = List.map analyse_type_expr el in
      (* l : (AstType.expression x typ) list *)
      let le = List.map fst l in
      let lt = List.map snd l in
      if (estCompatible lt tp) then
        (AstType.AppelFonction (s, le, info), tr)
      else raise ErreurType
```