TD2 TdL

Traduction des langages

Arbre syntaxique abstrait, table des symboles et résolution des identifiants

Objectif :
- Définir l'AST (Abstract Syntax Tree) pour le langage RAT
- Comprendre le traitement réalisé par la passe de résolution des identifiants
- Comprendre l'intérêt de la table des symboles
- Définir la nouvelle structure d'arbre obtenue après la passe de résolution des identifiants
- Définir les actions à réaliser par la passe de résolution des identifiants

1 Le 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 Exemple

```
bool less ( rat a rat b ){
  return ((num a * denom b )<( num b * denom a ));
}

prog{
  rat a = [3/4];
  rat b = [4/5];
  const n = 5;
```

```
  int i = 0;
  while(i<n){
    a=(a+a);
    b=(b*b);
    i=(i+1);
  }
  if(call less(a b)){print a;} else {print b;}
}
```

2 Structure de l'AST pour RAT

▷ **Exercice 1** *Définir la structure de l'AST pour RAT*

3 Passe de résolution des identifiants

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 première passe est une passe de résolution des identifiants. C'est elle qui vérifie la bonne utilisation des identifiants. En particulier, dans le cas du langage RAT, c'est elle qui vérifie que :

- tout identifiant utilisé est bien déclaré ;
- les identifiants ne sont pas déclarés deux fois dans le même bloc ;
- les identifiants sont utilisés correctement (les constantes et fonctions ne sont pas modifiées, par exemple)
- ...

Pour réaliser cette passe, nous avons besoin d'une table des symboles : structure de données associant aux identifiants leurs informations.

3.1 Table des symboles

L'interface du module Tds utilisé en TP est la suivante :

```
type info =
  | InfoConst of int
  | InfoVar of typ * int * string
  | InfoFun of typ * typ list

type tds

type info_ast

(* Création d'une table des symboles à la racine *)
val creerTDSMere : unit -> tds

(* Création d'une table des symboles fille *)
(* Le paramètre est la table mère *)
val creerTDSFille : tds -> tds
```

```
(* Ajoute une information dans la table des symboles locale *)
(* tds : la tds courante *)
(* string : le nom de l'identificateur *)
(* info : l'information à associer à l'identificateur *)
(* retour : unit *)
val ajouter : tds -> string -> info_ast -> unit

(* Recherche les informations d'un identificateur dans la tds locale *)
(* Ne cherche que dans la tds de plus bas niveau *)
val chercherLocalement : tds -> string -> info_ast option

(* Recherche les informations d'un identificateur dans la tds globale *)
(* Si l'identificateur n'est pas présent dans la tds de plus bas niveau *)
(* la recherche est effectuée dans sa table mère et ainsi de suite *)
(* jusqu'à trouver (ou pas) l'identificateur *)
val chercherGlobalement : tds -> string -> info_ast option

(* Affiche la tds locale *)
val afficher_locale : tds -> unit

(* Affiche la tds locale et récursivement *)
val afficher_globale : tds -> unit

(* Créer une information à associer à l'AST à partir d'une info *)
val info_to_info_ast : info -> info_ast

(* Récupère l'information associée à un noeud *)
val info_ast_to_info : info_ast -> info

(* Modifie le type si c'est une InfoVar, ne fait rien sinon *)
val modifier_type_info : typ -> info_ast -> unit

(* Modifie les types de retour et des paramètres si c'est une InfoFun, ne fait rien sinon *)
val modifier_type_fonction_info : typ -> typ list -> info_ast -> unit

(* Modifie l'emplacement (dépl, registre) si c'est une InfoVar, ne fait rien sinon *)
val modifier_adresse_info : int -> string -> info_ast -> unit
```

3.2 Structure de l'AST post passe de résolution des identifiants

La passe de résolution des identifiants réalise des vérifications mais prépare également les passes suivantes. Elles auront besoin d'avoir accès aux informations des identifiants, il faut donc modifier l'AST de façon à rendre ces informations accessibles.

▷ **Exercice 2** *Définir la structure de l'AST post passe de résolution des identifiants.*

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

▷ **Exercice 3** *Définir les actions à réaliser lors de la passe de résolution des identifiants.*

TL
TD2

Traduction des langages

Arbre syntaxique abstrait, table des symboles et résolution des identifiants.

→ Définir l'arbre abstrait pour manipuler le langage RAT. (en CamP)

```
type programme = Programme of fonction list * bloc
and fonction = Fonction of typ * string * (string * typ) list * bloc * expression
and bloc = Bloc of instruction list
and instruction =
        | Declaration of typ * string * expression
        | Affectation of string * expression
        | Constante of string * int
        | Affichage of expression
        | Conditionnelle of expression * bloc * bloc
        | Tant_Que of expression * bloc
and typ = Bool | Int | Rat | Undefined
and expression =
        | Rationnel of expression * expression
        | Ident of string
        | Numerateur of expression
        | Denominateur of expression
        | True
        | False
        | Entier
        | ExpressionBinaire of expression * operateurbinaire * expression
        | Call of string * expression list
and operateurbinaire = Plus | Moins | Inf | Equ
```

TDS : Associe un nom à sa valeur / information.
- Lors d'une déclaration, on ajoute l'élément à la TDS.
- Lors d'une affectation, on vérifie que l'elt ∈ à la TDS.

⚠ p 73 poly cours TDS par bloc.

Programme Rat —(analyseur syntaxique & lexical)→ Ast type programme —(passe gestion identificateurs + Vérification)→ Ast type **Programme**

→ Exercice 2 : On reprend la structure précédente avec qq modifs

```
Programme ...
        | Declaration of typ * info_ast * expression
        | Affectation of info_ast * expression
        | Empty
        ...

expression =
        ...
        | Ident of info_ast
        ...
```

→ Exercice 3 : Code pour la passe sur les identifiants

(programme → programme)

```
let analyser tds (Prog (Pf, Bloc Pi)) =
    let tds = creerTDS () in
    let nPf = analyser_tds_fonctions Pf tds in
    let nPi = analyser_tds_instructions Pi tds in
        Prog (nPf, Bloc nPi)

    instruction -> tds -> instruction
let rec analyse_tds_instruction i tds =
    match i with
      | Declaration (t, n, e) ->
            match chercherLocalement tds n with
              | Some _ -> raise (DoubleDeclaration n)
              | None -> let i = InfoVar (Undefined, 0, "") in
                        let ia = info_to_info_ast i in
                        let ne = analyser_tds_expr tds e in
                            ajouter tds n ia;
                            Declaration (t, ia, analyser_tds_expression tds e)
      | Affectation (n, e) ->
            match chercher Globalement tds n with
              | None -> raise (Pas Declarree n)
              | Some i -> vérifier que c'est une variable
                          ↳ Affecter (i, analyse expression)
      | Constante (n, @r) -> chercher localement
                             Some -> crie
                             None -> Ajoute à tds en créant InfoConst et renvoyer Empty

      | Affichage -> Affichage (analyse e)
```

Dans analyse nouveau bloc, nouvelle table de fille