Traduction des langages

1h30 avec documents

Année 2022-2023

Le barème est donné à titre indicatif. Les quatre exercices sont indépendants.

1 Compréhension des TP (3pt)

1. Dans les TP, comment sont gérées les constantes (évolution des AST et actions réalisées à chaque passe) ?
2. Lorsque le compilateur analyse une instruction "return", il doit s'assurer que cette instruction est bien dans le corps d'une fonction et que le type est compatible avec le type de retour déclaré de la fonction. Dans les TP, comment est résolu ce problème ?
3. Implanter une analyse permettant d'avoir la garantie que tous les appels à une fonction rencontreront bien un "return" est un problème complexe. De le cas où l'instruction "return" n'est pas appelée, l'exécution du code généré a un comportement non défini. Comment est-t'il possible de traiter ce problème ?

2 Gestion des identifiants et typage (6pt)

Le traitement sémantique de RAT étudié en TD et implanté en TP ne permet pas la surcharge de fonction (deux fonctions de même nom mais dont le type des paramètres est différent).

Nous souhaitons autoriser cette surcharge de fonctions.

Il n'y a aucune modification de la grammaire.

Exemple de programme valide

```
int f (int i1 int i2) { return 1; }
int f (rat r1 rat r2) { return 2; }

main{
  int i = call f (1 2) ;
  print i;
  i = call f ([1/2] [1/3]);
  print i;
}
```

Pour chaque question, donner des explications complétées par un bout de code OCaml ou pseudo code.

1. **Gestion des identifiants**
   (a) Comment est modifiée la passe de gestion d'identifiants pour la définition de fonction.
   (b) Comment est modifiée la passe de gestion d'identifiants pour l'appel de fonction.
2. **Typage**
   (a) Comment est modifiée la passe de typage pour la définition de fonction.
   (b) Comment est modifiée la passe de typage pour l'appel de fonction.

3 Analyse syntaxique et typage (4pt)

Nous souhaitons compléter notre langage RAT avec l'opérateur += qui additionne deux valeurs (expression à droite et valeur de la variable à gauche) et range le résultat dans la variable (à gauche).

L'opérateur s'utilise de la façon suivante :

```
prog{
  int x = 1;
  x += 2;
  print x;
  rat y = [1/2];
  y += [1/3];
  print y;
}
```

1. Donner la ou les nouvelle(s) règle(s) de production à ajouter à la grammaire de RAT.
2. Donner le ou les nouveau(x) **jugement(s) de typage** à ajouter au système de type de Rat.
3. Est-ce que ASTType.instruction doit être modifié ? Si oui, comment ? Si non, pourquoi ?

4 Placement mémoire (4pt)

RAT étendu permet de définir et manipuler les enregistrements :
— $TYPE \rightarrow struct\ \{\ DP\ \}$: définition d'un enregistrement (DP est une liste des types et noms des champs) ;
— $A \rightarrow (A.id)$: accès à un champ de l'enregistrement ;
— $E \rightarrow \{\ CP\ \}$: création d'un enregistrement avec la liste des valeurs de ses champs.

Exemple de programme valide (il doit afficher `3[1/4]`) :

```
main{
  struct {int x rat y} p = {3 [3/4]};
  print (p.x);
  (p.y) = [1/((p.x)+1)];
  rat z = (p.y);
  print z;
}
```

1. Les enregistrement seront-ils stockés dans la pile ou le tas ? Justifier la réponse.
2. Donner les adresses de p, p.x, p.y et z.
3. Généralisation : comment connaître l'adresse de id.nom_champ ?

5 Génération de code (3pt)

Nous voulons ajouter les affectations multiples au langage RAT : :
— $I \rightarrow LID = CP;$ (CP est une liste d'expression)
— $LID \rightarrow id$
— $LID \rightarrow id\ LID$

Exemple de programme valide :

```
main{
  int x = 0;
  rat y = [1/2];
  x y = 2 [1/3];
}
```

Nous supposons que le type ASTPlacement.instruction a été complété par AffectationMultiple of info_ast list * expression list . Après la passe de placement mémoire, nous avons la garantie que les deux listes sont de même taille et que les types ont été vérifiés.

1. Donner le code TAM associé au programme RAT précédent.
2. Généralisation : donner le code ou pseudo code du traitement du cas des instructions AffectationMultiple lors de la passe de génération de code.