Traduction des langages

1h30 avec documents

Année 2020-2021

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

1 Détection de similitudes (4 points)

Les outils de comparaison de code n'ont aucun souci à identifier comme similaires des codes identiques :

1. à l'indentation près ;
2. au renommage des variables près ;
3. à l'inversion des conditions dans le if (et donc inversion des blocs then / else) près ;
4. à l'ordonnancement des instructions et des définitions de fonctions près ;

Vous êtes prévenus...

Pour chacun des quatre points précédents, et en vous basant sur les concepts étudiés dans le cours, expliquez comment ces outils peuvent être implantés.

1. *La passe d'analyse lexicale élimine les blancs. La séquence des tokens est donc la même pour deux codes identiques à l'indentation près.*
2. *Après la passe de placement mémoire, il suffit de comparer les TDS pour voir si leur contenu est identique ⇒ mêmes variables. On peut aussi comparer le code assembleur engendré, vu que le nom des variables n'y a pas d'influence.*
3. *Transformer* `if (c) b1 b2` *en* `if (!c) b2 b1`.
4. *Comparer les AST (à n'importe quelle passe), en vérifiant si chaque sous-arbre présent dans l'un est présent dans l'autre.*

2 Gestion des identifiants (4 points)

1. Pour chacun des codes RAT suivant, est-ce que votre compilateur lève une exception pour double déclaration de `a` ? Expliquez pourquoi ?

(a)
```
int f (int a) {
    if (a=3) {
        rat a = [3/2];
        print a;
    } else {
        print a;
    }
    return 0;
```

```
}

main {
    print (call f (1));
}
```

(b)

```
int f (int a) {
    rat a = [3/2];
    print a;
    return 0;
}

main {
    print (call f (1));
}
```

(a) Pas de double déclaration : la variable a est dans un bloc qui possède sa propre table des symboles

(b) Ça dépend : double déclaration si une seule table des symboles est utilisée pour les paramètres et le corps de la fonction ; pas de double déclaration s'il y a une table des symboles pour les paramètres + une table fille pour le corps (ce qui est naturel si on considère le corps comme un bloc).

2. Si dans le code (b) une exception n'est pas levée, expliquez quelles modifications doivent être faites au compilateur pour qu'elle soit levée. Inversement, si une exception est levée, expliquez quelles modifications doivent être faites pour qu'elle ne soit pas levée.
   Comme expliqué, il faut soit utiliser la même table des symboles pour paramètres et corps (pour avoir l'exception), soit deux tables (pour ne pas avoir l'exception).
3. Donnez la nouvelle version du code du compilateur en ce qui concerne la déclaration de variables pour obtenir le comportement souhaité.
   Modifier `analyse_tds_fonction` *pour créer / ne pas créer de tds fille pour le coprs.*

3 Typage (4 points)

Nous souhaitons ajouter au langage RAT le traitement des tableaux. Les opérations sur les tableaux sont :

- la déclaration : **tab**<bool> t
- l'initialisation : **new tab**<int>(6) où 6 est la taille du tableau
- une opération pour l'accès (en lecture et en écriture) à un élément : **item** t 2 permet l'accès au troisième élément du tableau (on numérote à partir de 0).
- un "for each" sur les tableaux : **for** (rat elt : t){ ... }

Par exemple, on doit pouvoir écrire :

```
progTab1 {
    tab<int> t1 = new tab<int>(2);
    item t1 0 = 1;
    item t1 1 = 10;
    for (int elt : t1) {
        int a = elt;
```

```
        print a;
    }
    int b = item t1 0;
}
```

On ajoute pour cela les règles de grammaire suivante :
— $TYPE \rightarrow tab < TYPE >$
— $I \rightarrow item\ id\ E = E;$
— $I \rightarrow for\ (TYPE\ id : id)\ BLOC$
— $E \rightarrow new\ tab < TYPE > (E)$
— $E \rightarrow item\ id\ E$

Questions

1. Est-ce que la grammaire autorise de créer des tableaux de tableaux ? Justifiez votre réponse.
   Oui, par la première règle : $tab < tab < int >>$ est légal.
2. Donnez les jugements de typage à ajouter au système de type du langage RAT.

$$\frac{\sigma \vdash TYPE : \tau}{\sigma \vdash tab < TYPE > : tab(\tau)}$$

$$\frac{\sigma \vdash id : tab(\tau) \qquad \sigma \vdash E_1 : int \qquad \sigma \vdash E_2 : \tau}{\sigma \vdash item\ id\ E_1 = E_2 : void, []}$$

$$\frac{\sigma \vdash TYPE : \tau \qquad \sigma \vdash id_2 : tab(\tau) \qquad (id_1, \tau) :: \sigma \vdash BLOC : void, []}{\sigma \vdash for\ (TYPE\ id_1 : id_2)\ BLOC : void, []}$$

$$\frac{\sigma \vdash TYPE : \tau \qquad \sigma \vdash E : int}{\sigma \vdash new\ tab < TYPE > (E) : tab(\tau)}$$

$$\frac{\sigma \vdash id : tab(\tau) \qquad \sigma \vdash E : int}{\sigma \vdash item\ id\ E : \tau}$$

3. Comment doit être modifié l'AST issu de la phase d'analyse syntaxique pour prendre en compte ces nouvelles constructions ?
   — *Ajouter dans instruction deux nouveaux cas $Foreach\ of\ string * string * bloc$ et $SetItem\ string * expression * expression$.*
   — *Ajouter dans expression deux nouveaux cas $NewTab\ of\ typ * expression$ et $GetItem\ of\ string * expression$.*
   — *Ajouter dans typ un nouveau cas $Tab\ of\ typ$.*

4 Placement mémoire (4 points)

Nous souhaitons ajouter au langage RAT le traitement des tableaux. Les opérations sur les tableaux sont :
— la déclaration : **tab**<bool> t
— l'initialisation : **new tab**<int>(6) où 6 est la taille du tableau
— une opération pour l'accès (en lecture et en écriture) à un élément : **item** t 2 permet l'accès au troisième élément du tableau (on numérote à partir de 0).
— un "for each" sur les tableaux : **for** (rat elt : t){ ... }

On ajoute pour cela les règles de grammaire suivante :
— $TYPE \rightarrow tab < TYPE >$
— $I \rightarrow item\ id\ E = E;$
— $I \rightarrow for\ (TYPE\ id : id)\ BLOC$
— $E \rightarrow new\ tab < TYPE > (E)$
— $E \rightarrow item\ id\ E$

Questions

1. Est-ce que la taille du tableau est connue à la compilation ? Justifiez votre réponse.
   Pas connue car new prend une expression. On peut écrire `new tab<int>(x+3)` *où* `x` *est une variable entière.*
2. Où doit être stocké le tableau : dans la pile ou dans le tas ? Justifiez votre réponse.
   Vu qu'on ne peut pas connaître la taille à la compilation, le tableau lui-même est stocké dans le tas. Une variable de type tableau contient un pointeur vers cette zone.
3. Donnez les adresses des variables du programme suivant :

```
rat f (rat a tab<int> b rat c){
    rat res = c;
    return res;
}

progTab2 {
    tab<int> t = new tab<int>(2);
    int a = item t 0 ;
    for(int elt : t){
        int b = elt;
        print b;
    }
    rat c = [2 / 3];
    if (c = [4/5]) {
        rat d = [4/5];
        int e = 3;
    } else {
        int f = 6;
        rat g = c;
    }
    int h = 0;
}
```

   Fonction f : c : -2[LB], b : -3[LB], a : -5[LB], res : 3[LB]
   Prog principal : t : 0[SB], a : 1[SB], elt : 2[SB], b : 3[SB], c : 2[SB], d : 4[SB], e : 6[SB], f : 4[SB], g : 5[SB], h : 2[SB]
   Un pointeur est de taille 1 donc t et b (param fonction) sont de taille 1.
   elt et b disparaissent après le bloc for.
4. Indiquez ce qu'il faut changer à la passe de placement mémoire pour prendre en compte les tableaux.
   — *Ajouter dans GetTaille le cas tableau (taille 1)*
   — *Ajouter le traitement de l'instruction $ForEach(id, \_, bloc)$: id est placé au déplacement courant et bloc est analysé avec le déplacement courant + taille du type de id.*
   — *L'instruction SetItem n'a aucun effet sur le déplacement courant.*

— *Rien à changer pour la déclaration de variable ou la gestion des paramètres (seul la taille du type intervient).*

5 Génération de code (4 points)

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). On ajoute à la grammaire une instruction $I \rightarrow id \mathrel{+}= E$.

L'opérateur s'utilise de la façon suivante :

```
prog {
    int x = 6;
    x += 5;
    print x;
    rat y = [4 / 5];
    y += [2 / 3];
    print y;
}
```

1. Donnez le code TAM du programme précédent.

```
; int x = 6;
PUSH 1
LOADL 6
STORE (1) 0[SB]
; x += 5;
LOAD (1) 0[SB]
LOADL 5
SUBR Iadd
STORE (1) 0[SB]
; print x;
LOAD (1) 0[SB]
SUBR IOut
; rat y = [4 / 5];
PUSH 2
LOADL 4
LOADL 5
STORE (2) 1[SB]
; y += [2 / 3];
LOAD (2) 1[SB]
LOADL 2
LOADL 3
CALL (SB) RAdd
STORE (2) 1[SB]
; print y;
LOAD 2 1[SB]
CALL (SB) Norm
CALL (SB) Rout
```

2. Dans le cas général, quel code doit être généré pour une instruction de la forme `id += expression;` ?

Le code est le même que celui pour `id = id + expression;` *:*
avec les infos de la variable id : (type, reg, dep)
"LOAD (taille de type) dep[reg]"
+ (analyser_code_expression e)
+ "SUBR IAdd" ou "CALL (SB) RAdd" selon type
+ "STORE (taille de type) dep[reg]"