Rapport du projet de la programmation fonctionnelle et traduction de langage

Kawtar LYAMOUDI et Nihal BELMAKHFI

Département Sciences du Numérique - classe hpc et big data - deuxième année
Programmation fonctionnelle et Traduction de Langage
2022-2023

Table des matières

1 Introduction

Ce projet a pour objectif de construire un compilateur pour le langage RAT. Au cours des séances de travaux pratiques, nous avons mis en place les quatre passes de gestion d'identifiants, de typage, de placement mémoire et de génération de code. Notre objectif dans ce projet sera de le traitement de nouvelles constructions : les pointeurs, le bloc else optionnel dans la conditionnelle, la conditionnelle sous la forme d'un opérateur ternaire, les boucles "loop" 'a la Rust. Nous allons maintenant vous présenter les différentes étapes de notre projet détaillées.

2 Definition d'une arbre AST

Les arbres de syntaxe abstraite (AST) représentent la structure sémantique d'un programme source, en éliminant les informations superflues telles que les parenthèses et les symboles de ponctuation. Ils sont souvent utilisés pour l'analyse statique et la génération de code cible, car ils fournissent une vue claire de la sémantique du programme.

3 Les différentes passes : partie TP

Pour définir les différentes passes aux TPs, il faut définir les fonction analyse_expression, analyse_instruction, analyse_fonction et analyser (analyse_programme)

3.1 Passe TDS

Lors de cette passe, on transforme les éléments de type : AstSyntax aux éléments de type AstTDS, afin de nous assurer de la bonne utilisation des identifiants(Tout identifiant est bien déclaré une fois et une seule, les identifiants sont déclarés correctement...)

3.2 Passe typage

Lors de cette passe, on transforme les éléments de type : AstTDS aux éléments de type AstType, qui a pour intéret la vérification des types

3.3 Passe placement mémoire

Lors de cette passe, on transforme les éléments de type : AstType aux éléments de type AstPlacement. Cette passe calcule l'adresse des variables dans la pile. Elle doit etre mise à jour dans les informations associées aux identificateurs.

3.4 Passe code

Lors de cette passe, on transforme les éléments de type : AstPlacement aux éléments de type string

4 pointeur

4.1 Modification de la grammaire

Avant d'ajouter le traitement des pointeurs, il faut tout d'abord ajouter les non teminaux (new,*,&) au lexer et les règles de grammaires nécessaire au traitement et permettant la définition et la manipulation des pointeurs.

— $A \rightarrow (* A)$: déréférencement : accès en lecture ou écriture à la valeur pointée par A ;
— $TYPE \rightarrow TYPE *$: type des pointeurs sur un type TYPE ;
— $E \rightarrow null$: pointeur null ;
— $E \rightarrow (new\ TYPE)$: initialisation d'un pointeur de type TYPE ;
— $E \rightarrow \&\ id$: accès à l'adresse d'une variable.

4.2 Evolution de la structure de l'AST

On ajoute un nouveau cas de type :
Pointeur of type
On ajoute un nouveau type :
affectable = Ident of string — Valeur of affectable
On remplace le type Ident de Expression par Affectable qui factorise l'identifiant et le déréferencement d'un affectable. On ajoute le type NEW of typ qui est l'initialisation d'un Pointeur sur un type typ. On ajoute Adresse d'un identifiant. On ajoute la valeur NULL d'un Pointeur. Ceci se résume en :
Affectable of affectable — Null — New of typ — Adresse of string
On a aussi ajouté dans le fichier type.mli Pointeur of typ au typ :
type typ = Bool — Int — Rat — Undefined — Pointeur of typ
On change le type Affectation de l'instruction en ajoutant Affectable :
Affectation of affectable * expression

4.3 Modification des passes

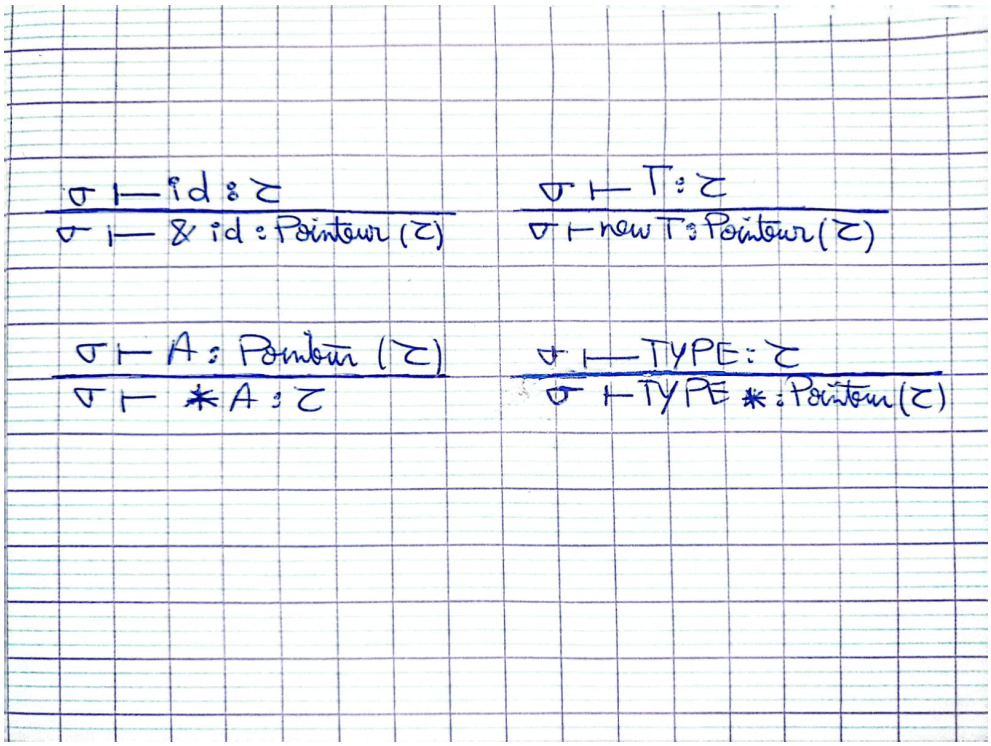
La passe TDS concerne l'ajout des différentes expressions et des affectables pour vérifier et ajouter les valeurs voulues à la TDS créee par le programme. Dans cette passe, nous avons changé les string par des Tds.infoast .Nous avons ensuite ajouté une méthode pour analyser les affectables . Aprés nous avons traité les cas du type Affectable ajouté à l'AST dans les méthodes d'analyse des expressions et des instructions.

La passe de typage a pour objectif la vérification de la compatibilité des types et de leur sauvegarder dans la TDS, aprés avoir ajouté de nouveaux types et mis à jour les méthodes d'analyse de type pour prendre en concidération les pointeurs.

Concernant la passe de placement mémoire, un pointeur prend une place en mémoire de 1, ce qui est indépendant du type pointé.

Pour la passe de code, on adapte l'accès à la mémoire en fonction des pointeurs. Pour cela, on récupère les adresses mémoires au lieu des valeurs des variables lorsque cela est nécessaire.

4.4 jugement de typage



4.5 Test

Pour tester qu'on a bien ajouté la partie pointeur, on a commencé par transformer l'exemple du sujet en test dans le fichier testPointeur1.rat qui se trouve dans le dossier gestion_id, sans_fonction. Le meme test, on l'a ajouté à type, sans_fonction, fichierRat. Et tam, sans_fonction, fichierRat. On a ajouté d'autres tests par la suite dans les memes dossiers. Finalement on a modifié le fichier

Exemple de programme valide

```
main{
  int * px = (new int);
  (* px) = 42;
  print (*px);
  int x = 3;
  px = &x;
  int y = (*px);
  print y;
}
```

test, en lui ajoutant des lignes reflettant la définition des tests des pointeurs

5 Bloc else optionnel

5.1 Modification de la grammaire

Avant d'ajouter le traitement du bloc else optionnel, il faut tout d'abord ajouter les règles de grammaires nécessaire au traitement et permettant la définition et la manipulation du bloc else optionnel.

$$13.\ I \rightarrow if\ \ E\ \ BLOC$$

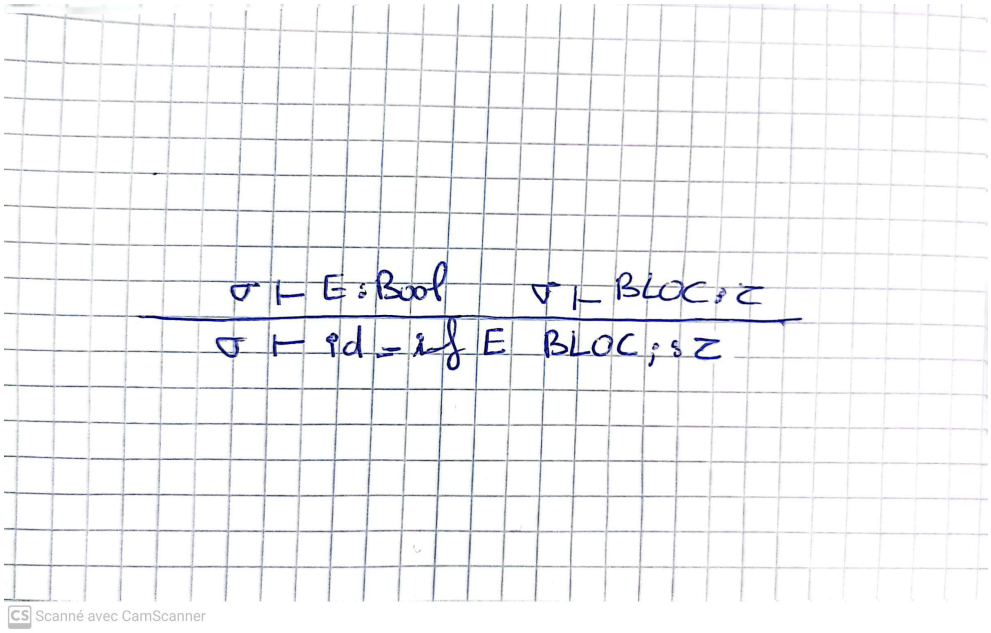
5.2 Evolution de la structure de l'AST

On a ajouté — ConditionnelleOpt of expression * bloc à la définition d'instruction dans la structure de l'AST.

5.3 Modification des passes

Dans chacune des passes : TDS, type, placement mémoire et code, on doit ajouter un traitemengt pour ConditionnelleOpt : AstSyntax.ConditionnelleOpt, AstTds.ConditionnelleOpt, AstType.ConditionnelleOpt, AstPlacement.ConditionnelleOpt

5.4 jugement de typage



5.5 Tests

Pour tester qu'on a bien ajouté la partie bloc optionnel, on a commencé par transformer l'exemple du sujet en test dans le fichier testConditionnelleOpt.rat qui se trouve dans le dossier gestion_id, sans_fonction. Le meme test, on l'a ajouté à type, sans_fonction, fichierRat. Et tam, sans_fonction, fichierRat. On a ajouté d'autres tests par la suite dans les memes dossiers. Finale-

Exemple de programme valide

```
test{
  int i = 0 ;
  while (i <11){
    if (denom [i/2] = 1){print i;}
    i = (i+1);
  }
}
```

ment on a modifié le fichier test, en lui ajoutant des lignes reflettant la définitions des tests de else conditionnelle

6 La conditionnelle sous la forme d'un operateur ternaire

6.1 Modification de la grammaire

Pour ajouter le traitement de la conditionnelle sous la forme d'un operateur ternaire, nous avons tout d'abord modifié notre lexer et parser afin d'ajouter les non teminaux (? et :) et les règles de grammaires nécessaire au traitement :

42. $E \rightarrow (E \ ? \ E : E)$

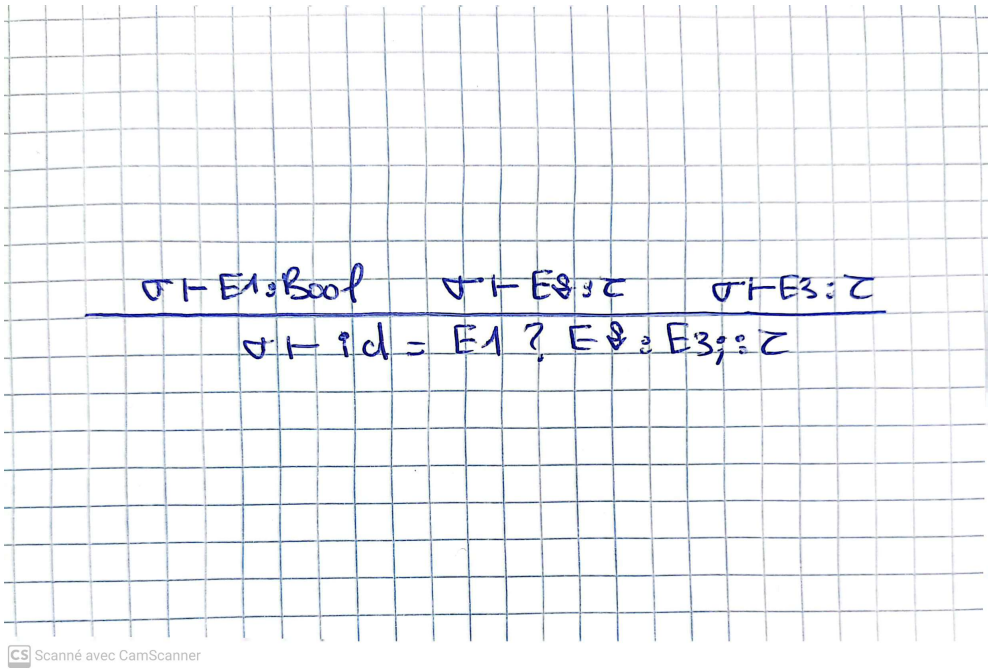
6.2 Evolution de la structure de l'AST

On a ajouté — ConditionnelleTer of expression * expression * expression à la définition d'expression dans la structure de l'AST.

6.3 Modification des passes

Dans chacune des passes : TDS, type, placement mémoire et code, on doit ajouter un traitemengt pour ConditionnelleTer : AstSyntax.ConditionnelleTer, AstTds.ConditionnelleTer, AstType.ConditionnelleTer, AstPlacement.ConditionnelleTer.

6.4 Jugement de typage



6.5 Tests

Pour tester qu'on a bien ajouté la partie conditionnelle sous la forme d'un operateur ternaire, on a commencé par transformer l'exemple du sujet en test dans le fichier testConditionnelleTer.rat qui se trouve dans le dossier gestion_id, sans_fonction. Le meme test, on l'a ajouté à type, sans_fonction, fichierRat. Et tam, sans_fonction, fichierRat. On a ajouté d'autres tests par la suite dans les memes dossiers.

Exemple de programme valide

```
int min (int a int b){
  return ((a<b)?a : b);
}

test{
  rat a =[2/3];
  rat y = ((denom a = 3) ? [1/3] : [0/3]);
  print ((call min ((num y) 1)=1) ? true : false );
}
```

7 Conclusion

En conclusion, les tps et le projet de construction d'un compilateur pour le langage RAT a été très enrichissant.Ils nous ont permis de découvrir le fonctionnement précis d'un compilateur et sa construction.

Or on a eu quelques difficultés au début en ce qui concerne la familiarisation avec la structure des différents fichiers. Avec le temps, on est peu à peu devenues plus alaises à manipuler les différentes ressources.

Nous avons mis en place les quatre passes de gestion d'identifiants, de typage, de placement mémoire et de génération de code. Nous avons également ajouté de nouvelles constructions telles que les pointeurs, le bloc else optionnel dans la conditionnelle et la conditionnelle sous forme d'opérateur ternaire. Bien que ces parties aient été difficiles à mettre en place, nous avons réussi à les tester avec succès à l'aide de fichiers test. Malheureusement, nous n'avons pas pu terminer la dernière partie concernant les boucles "loop" en raison de contraintes de temps. Nous espérons mieux gérer notre temps pour les projets à venir. En résumé, ce projet nous a permis de développer nos compétences en programmation et en traduction de langage.