Rapport
Projet PF-TDL

Adrien Chevallereau
Ghislain Réveiller

Département Sciences du Numérique
Deuxième année - Architecture, Systèmes et Réseaux
2021-2022

Table des matières

1 Introduction

Le projet avait pour but d'étendre le compilateur du langage RAT réalisé en TP de traductions des langages en utilisant ce que nous avons vu en programmation fonctionnelle et en traduction des langages.

Nous devions donc rajouter les fonctionnalités suivantes :

1. Les Pointeurs
2. L'opérateur d'assignation d'addition
3. Les Types Nommés
4. Les Enregistrements

A l'heure où nous écrivons ces lignes nous n'avons pas encore fini Enregistrement, mais le reste est complet.

2 Les types

Pour l'AstSyntax et l'AstTds nous avions besoin de créé 2 types, l'affectable et nommes. L'affectable a été créé pour répondre aux règles 20 à 22 et à leurs utilisations dans les instructions et expressions. nommes permet de répondre à la règle 5 et à son utilisation dans le programme. L'affectable est présent dans toute les Ast mais pas les nommes.

Dans expression, nous avons ajouté $Null$, $Affectable$, $Adresse$, New et $Enregistrement$ qui sont présents dans toutes les Ast. Elles permettent de répondre aux règles 45 à 49 de plus Ident a été supprimé remplacé dans affectable.

Dans instruction, nous avons ajouté Affectation, Addition, Typedeflocal qui répondent aux règles 12, 13 et 19. Addition et Typedeflocal disparaissent après la phase de typage, car addition est transformé en une affectation et on a plus besoin de typdeflocal.

Enfin dans programme, on ajoute un élément qui est une liste de nommes qui là aussi disparaît après la phase de typage.

3 Les jugements de typage

Nous avons décidé de ne pas redonner les jugements de typage déjà donnés dans le TD3. Nous avons seulement ajouté les jugements qu'il faut pour ajouter les 4 fonctionnalités (en mentionnant les jugements supprimé dû à l'évolution de la grammaire)

3.1 Ajout pour les pointeurs

— ajout des Expressions suivantes

$$\sigma \vdash null : Pointeur(Undefined)$$

$$\frac{\sigma \vdash TYPE : \tau}{\sigma \vdash (new\ TYPE) : Pointeur(\tau)}$$

$$\frac{\sigma \vdash id : \tau}{\sigma \vdash \&id : Pointeur(\tau)}$$

— ajout des Affectables (remplace E -> id)

$$\sigma \vdash id : \tau$$

$$\frac{\sigma \vdash A : Pointeur(\tau)}{\sigma \vdash (*A) : \tau}$$

3.2 Ajout pour l'opérateur d'assignation d'addition

— ajout d'une instruction

$$\frac{\sigma \vdash A : \tau \qquad \sigma \vdash E : \tau}{\sigma, \tau_r \vdash A += E : void, []}$$

3.3 Ajout pour les types nommés

— instruction ajoutée pour définir un type nommé local

$$\frac{\sigma \vdash tid : \tau \qquad \sigma \vdash TYPE : \tau}{\sigma, \tau_r \vdash typedef\ tid = TYPE\ :\ void, [tid, \tau]}$$

— ajout suite de types nommés globaux TD

$$\frac{\sigma \vdash tid : \tau \qquad \sigma \vdash TYPE : \tau \qquad \sigma, \tau_r \vdash TD : void, \sigma'}{\sigma, \tau_r \vdash typedef\ tid = TYPE\ ;\ TD\ :\ void, \sigma' @ [tid, \tau]}$$

$$\sigma, \tau_r \vdash\ :\ void, []$$

— remplacement du jugement de typage du programme précédent

$$\frac{\sigma \vdash TD\ : void, \sigma' \qquad \sigma' @ \sigma \vdash FUN\ : void, \sigma' \qquad \sigma'' @ \sigma' @ \sigma \vdash PROG\ : void, \sigma'''}{\sigma \vdash TD\ FUN\ PROG : void, \sigma''' @ \sigma'' @ \sigma'}$$

3.4 Ajout pour les enregistrements

— ajout dans Affectables

$$\frac{\sigma \vdash A : \sigma_p, struct\ \{\tau_1 \times \tau_2\ ...\ \times \tau_n\} \quad \sigma_p @ \sigma \vdash id\ : \tau_x}{\sigma \vdash A.id : \tau} \quad avec\ \tau_x \in\ \{\tau_1, ...\tau_n\}$$

4 Les Pointeurs

4.1 Modification du Lexer et Parser

Pour les pointeurs, il a fallu ajouter dans le lexer les mots-clés *new* et *null* plus le caractère spécial & qui sont des nouveaux éléments de langage utilisé pour les pointeurs avec RAT.

Pour le Parser nous avons ajouté les différentes règles comme l'instruction Affectation pour la règle 12 et les expressions Affectable, New, Null et Adresse pour les règles 45 à 48 de la grammaire étendue du langage RAT.

Nous avons aussi ajouté l'attribut affectable composé de Deref et Ident conformément aux règles 20-21. Et par conséquent nous avons supprimé l'expression Ident qui a aussi été supprimé sur toute les AST et toute les passe.

Nous avons aussi ajouté le type Pointeur conformément à la règle 28

4.2 Modification de la Passe Tds

Pour Cette passe nous avons créé une fonction analyse_tds_affectable afin de traiter :
- Ident : vérifie qu'il est déclaré et bien utilisé
- Deref : analyse de l'affectable pointé

Nous avons aussi ajouté les éléments suivants :
- expression Affectable : analyse son contenu
- expression Null et New : ne fait rien à part la transformation en AstTds
- expression Adresse : vérifie l'existence de l'identifiant et de sa bonne utilisation
- instruction Affectation : analyse l'expression et l'affectable qu'il contient

4.3 Modification de la PasseType

Premier chose à dire, les analyses de la passe type rat renvoie à chaque fois un AstType avec le type qu'elle contient à l'exception de l'analyse de la fonction, analyser et analyse d'une instruction.

Nous avons aussi créé des fonctions qui récupère les types des info_ast c'est-à-dire les types retour (get_type_return) et des paramètres des fonctions (get_type_param) si infofun et des ident si infovar (get_type).

Pour cette passe analyse_type_affectable traite :
- Ident : récupère le type dans son info_ast. Il y a des raise mais ne devrait pas être levé car déjà vérifié dans la passe précédente
- Deref : analyse de l'affectable pointé avec analyse_type_affectable

Enfin nous avons aussi ajouté dans les différentes analyses différents éléments que l'on doit traiter :
- expression Affectable : analyse son contenu
- expression Null et New : ne fait rien à part la transformation en AstType Null à pour type Undefined et pout new on récupère le type qu'il contient
- expression Adresse : ne fait rien, récupère le type avec get_type de son info_ast
- instruction Affectation : analyse l'expression et l'affectable qu'il contient et vérifie que leur type sont compatibles (pour le pointeur la fonction est_compatible récupère récursivement le type pointé).

4.4 PassePlacement

Nous n'avons pas eu besoin de modifier cette passe, car les pointeurs occupent une seule place dans la pile comme un int par exemple et les données vers lequel il pointe sont stockées dans le tas donc pas besoin de calculer leur emplacement.

4.5 PasseCodeRatToTam

Le principal changement est la fonction $analyse\_tam\_affectable$ qui va s'occuper de générer le code de tous les affectable qu'il soit du côté gauche ou droit de l'égalité. Après réflexion, on se retrouve avec une fonction qui fait 2 choses distinctes donc on aurait pour en faire 2 fonctions, mais elles auraient eu la même structure.

Ainsi on a 4 possibilités différentes :
- expression sans pointeur : pas de changement, on charge la variable dans la pile
- affectation sans pointeur : pas de changement on stocke la variable dans la pile à son emplacement de stockage
- expression avec un pointeur : on charge dans la pile l'adresse du pointeur et on charge ensuite ses données stockées dans le tas
- affectation avec pointeur : on charge dans la pile l'adresse du pointeur puis on stocke dans le tas les informations de la pile

Actuellement on ne prend pas en charge les pointeurs de pointeurs, mais il suffirait de mettre un compteur à la place d'un true/false du paramètre pointeur dans la fonction précédente

Le second changement est la modification de $analyse\_tam\_expression$ avec l'ajout de :
- Null : alors on renvoie la valeur 0 par définition du pointeur null.
- New : alors on réserve de l'espace en mémoire de la taille demandé par le pointeur
- Adresse : alors on charge l'adresse du pointeur
- Affectable : il remplace ident et alors on appelle la fonction $analyse\_tam\_affectable$ avec iOUe à false pour indiquer que c'est une expression.

5 L'opérateur d'assignation d'addition

Pour L'opérateur il a fallu apporter les modifications suivantes :
— Ajouter l'instruction Addition dans le parser, l'AstSyntax et AstTds
— Ajouter le traitement de Addition dans la passe tds consistant en une simple analyse de l'affectable et de l'expression, les composants de l'instruction
— Ajouter le traitement Addition dans la passe type qui la transforme en une Affectation en fonction du type de l'affectable

6 Les Types Nommés

6.1 Modification du Lexer et Parser

Pour les types nommés, il a fallu ajouter le mot clé $typedef$. On a aussi ajouté un tident pour les identifiants des types nommés qui fonctionne de la même manière que ident dans le lexer.
Nous avons aussi remplacé la règle 2 par la règle 3 en ajoutant la liste des types nommés globalement

Dans le parser nous avons ajouté l'attribut nommes pour les règles 4 et 5. La règle 5 a été nommés Typedefglobal. Nous avons aussi ajouté l'instruction Typedeflocal pour la règle de grammaire 19. Et nous avons ajouté le type Tident dans l'attribut typ pour la règle 29

6.2 Modification de la PasseTds

Pour la passe tds nous avons ajouté dans l'analyse_tds_instruction le traitement de typedeflocal. Il vérifie que le type nommé n'existe pas localement et créé une info_ast composé d'une info de type InfoType (un type d'info ajouté dans la tds pour les types nommés) ; cette info_ast est ajouté dans la tds on ne modifie pas les composants de typedeflocal.

Ensuite dans analyser nous avons traité la liste des types nommés globalement avec une nouvelle fonction analyse_tds_td qui vérifie que le types nommés n'existe pas déjà globalement et qui ajoute à la tds un info_ast créé(un infotyp).

6.3 Modification de la PasseType

Nous avons d'abord modifié analyser où l'on crée une liste qui contient les couples (nom, type) des types nommés. Cette liste est ensuite envoyé dans analyse_type_{fonctionretour / param / bloc / instruction}. Si un type dans l'analyse des paramètres où dans la déclaration dans analyse des instructions où dans l'analyse de fonctionretour est un Tident (type ajouté dans Type.ml/mli) alors on appelle la fonction changer_type. Celle-ci recherche dans la liste des types nommés le type correspondant au Tident et renvoie le bon type que l'on met dans l'info_ast du paramètre/déclaration. Donc en soit on transforme les types nommés en des simples types Int, Rat, Bool.

Aussi dans analyse_type_bloc le code a été modifié de telle sorte que si un type est définie localement ce type est ajouté à la liste.

Il n'y a ainsi pas de modification dans la passe de placement ni dans la passe de génération de code

7 Les Enregistrements

7.1 Modification du Lexer et Parser

Dans le lexer nous avons ajouté le mot clés *struct* pour créé l'enregistrement et nous avons aussi ajouté le caractère spécial de rat point pour l'accès aux éléments de l'enregistrement

Ainsi dans la Parser, nous avons ajouté l'expression enregistrement pour le respect de la règle 49 du langage RAT. Nous avons aussi ajouté l'affectable accès pour le respect de la règle 22 qui permet d'accéder au élément d'un enregistrement. Et enfin nous avons ajouté le type Record pour respecter la règle 30 de la grammaire afin de créer des enregistrements.

7.2 Modification de la Passetds

Dans l'analyse_tds_instruction pour la déclaration si le type est un record. On appelle la fonction analyse_tds_record qui permet d'ajouter dans la tds et dans l'inforecord(une info ajouté dans tds.ml/mli) les différentes infovar de l'enregistrement.

Dans analyse_tds_affectable on ajoute Acces. On vérifie que l'identifiant n'existe pas déjà localement puis on analyse l'affectable que contient Acces. Ensuite on appelle trouverIAdeN qui permet avec la fonction trouvern de trouver l'info_ast correspondant à l'identifiant que contient Acces.

8 Conclusion

Les principales difficultés rencontrées est la vérification des identifiants pour les enregistrements, car en effet lors de l'appel de p.x il faut vérifier que :

— x n'est pas déjà redéfini localement
— p est bien défini
— et x est bien défini dans p

Il faut donc que les champs d'enregistrement de p soit stocké sur p. De plus, lorsque l'on cherche localement, s'il n'est pas redéfini, il faut que l'on trouve x de p. Mais si on redéfinit un x au même niveau que la déclaration il faut lever une erreur. Voilà le point auquel nous n'avons pas réussi à trouver de solution.

Les autres problèmes rencontrés ont été de bien penser à changer les types nommés par leurs vrais types dans les paramètres des fonctions et dans les types retour.

Enfin il a été difficile de se répartir le travail car les fonctionnalités doivent pratiquement être traités dans l'ordre et on est obligé de traiter les passe dans l'ordre. On aurait pu commencer à implémenter enregistrement avant type nommé, mais on en voyait pas vraiment l'intérêt de l'implémenter sans pouvoir le nommer. On a ainsi donc travaillé sur le même espace de travail partagé via liveshare pour coder sur le même code. Cela n'a pas forcément été une perte de temps. Je dirai même que l'on a été complémentaire grâce à une bonne communication.

Sinon nous avons trouvé le sujet très intéressant, car on a eu l'impression de faire quelque chose d'utile et qui pourra nous resservir plus tard (pas forcément le code en tant que tel, mais plutôt la façon de coder par passe)