Rapport Projet PF&TDL

Introduction

Le but du projet est d'étendre le compilateur du langage RAT réalisé en TP de traduction des langages pour traiter de nouvelles constructions : les **pointeurs**, les **tableaux**, les **types nommées**, les **boucles "for"** et les **prototypes** de fonctions et de types. Ce rapport détail l'ajout des nouvelles constructions vis-à-vis du compilateur initialement développé pour chaque passe de celui-ci. Les modifications significatives apportées au code et les choix de conception choisis pour réaliser ces nouvelles constructions.

Table des matières

Preambule

Avant de détailler les nouveaux ajouts, nous avons réorganisé la structure du projet afin de faciliter l'intégration des nouveaux composants et la construction du code avec dune (jbuild).

L'architecture du projet est résumé comme ci-dessous :

> rat_paux_verstraeten/
> _build/
> rat/
> *.ml
> dune (ce fichier définie la libraire Rat ainsi que l'ensemble des tests)
> fichiersRat/
> main.ml
> main.sh (script permet d'executer du code rat)
> dune (ce fichier définie l'exécutable qui permet de compiler du code rat)

Nous avons choisi d'utiliser la nouvelle version de jbuild, nommé dune pour pouvoir se référer rapidement à la documentation en ligne[1]. Le soin de la compatibilité avec les machines de l'ENSEEIHT à soigneusement été vérifiée.

1 https://jbuilder.readthedocs.io/en/latest/

Jean-Baptiste PAUX
Baptiste VERSTRAETEN

1 L’arbre syntaxique et les types

L’arbre syntaxique (AST) s’est vu doté de nouveaux ajouts lors de l’intégration des nouveaux composants. De même, les types ont été étendus pour les accueillir comme les pointeurs et les tableaux. Nous détaillerons dans la suite les choix de conception faits ainsi que leurs justifications.

a) L’arbre syntaxique

L’arbre syntaxique a évolué de façon notable lors de l’ajout des pointeurs. En effet, la volonté d’avoir une grammaire non ambigüe a impliqué l’ajout d’un nouveau noeud nommé “Acces” répondant au besoin d’accèder aux valeurs pointées ou aux identifiants d’un programme. Il a été étendu aux accès des éléments dans les tableaux. Ce noeud c’est vu propagé au différentes passes étant donné qu’il nécessitait un traitement spécifique lors de la génération de code. Les prototypes ont également modifié la structure du noeud “Programme” offrant ainsi une souplesse au programmeur lors qu’il programme avec le langage Rat. Mais son but est simplement d’apporter un confort de programmation, il n’implique aucune modification sur la passe de génération. C’est pourquoi son traitement s’arrête à la passe de typage et la modification de l’AST n’implique que l’AST *Syntax* car les informations sont stockés par la suite dans le table des symboles.

Le reste des ajouts est mineur, ajoutant simplement des constructeurs aux types instruction et expression du langage. La table des symboles (TDS) à elle aussi subit des modifications par l’ajout des types nommés et prototypes car ces ajouts représentent essentiellement des “alias” au même titre que les variables du programme. Elles ont ainsi une place dans la TDS.

b) Les types

Les types ont évolué pour prendre en compte les pointeurs, les tableaux et les types nommés. Il n’y a pas grand chose de notable à dire sur eux, mise à part que leur structure est passé en récursif lors de l’ajout des pointeurs. Cela a complexifié l’analyse de la passe typage.

2 Les pointeurs

Les pointeurs fut un ajout important, impliquant plusieurs modifications au sein du compilateur. Le fait de les avoir traité en TD nous à permis de rapidement les intégrer. La grammaire que nous devions respecter impliquée une entré nommée affectable. Un affectable pouvait définir un identifiant (variable du programme) ou la valeur pointée. Chaque passe devait donc analyser un affectable. La passe TDS ajouter les identifiants dans la TDS, la passe de typage assurait que les accès et les affectables avaient le bon typage et celle de génération traduisait le comportement des pointeurs du point de vue du langage TAM. La passe la plus difficile est celle de génération. En effet, l’analyse des affectables consiste à analyser l’affectable de manière récursif en distinguant les accès à “gauche” de ceux à “droite” car dans un cas on enregistre une valeur dans la case mémoire pointée et dans l’autre on charge la valeur de la case mémoire pointée. Il fallait donc diviser l’analyse en distinguant la gauche de la droite.

Jean-Baptiste PAUX
Baptiste VERSTRAETEN

3 Les tableaux

L'implantation des tableaux a été réalisée assez rapidement car assez proche de celle des pointeurs, un tableau n'étant finalement qu'un pointeur sur son premier élément. Cependant afin de définir un typage fort dans le langage la distinction a été faite entre tableau et pointeur à tous les niveaux du compilateur, il n'est donc pas possible d'écrire quelque chose comme :

```
int* pt = (new int[5])
```

Comme cela aurait été possible en C. Le type tableau est donc différencié du type pointeur, ainsi que l'opération d'allocation de mémoire.

Comme pour l'accès à la variable pointé par un pointeur, l'accès d'un élément d'un tableau est une opération de type affectable, ce qui permet de chaîner les accès indice tableau et accès ponteur sans problème.

Le code tam de l'accès à la valeur d'un tableau est comme celui de l'accès à la variable pointée par un pointeur, avec un décalage en plus réalisé sur l'adresse afin d'accéder au bon indice.

4 Les types nommés

Les types nommés fut un ajout mineur, il apporte un confort de programmation au programmeur mais ne demande aucune modification pour la génération car les types nommés font juste références à de véritables types du langage (eg : tableau d'entier, pointeur de rationnel). Ainsi, comme les identifiants ils nécessitent une place dans la TDS. Un constructeur "InfoTyp" dans la TDS permet de retenir le véritable type désigné par le type nommé. D'autre part, nous avons gardé à l'esprit que ces types nommés peuvent être vus comme constants et ainsi traité comme les constantes du programme. Dans la passe TDS l'analyse des types permet donc de remplacer tous les types nommés par les véritables types stockés dans la TDS. Cela permet de conserver le comportement des passes en aval.

5 Les boucles "for"

L'idée à la base était de ne gérer les boucles for que dans les passes syntaxique et sémantique et de remplacer la boucle for par une déclaration (variable de boucle) et une boucle while contenant les instructions de la boucle for plus une affectation (incrémentation variable de boucle).

Cependant ceci n'était pas possible car la variable de boucle doit seulement être locale à la boucle et ne pas exister pour le code suivant la boucle comme ça l'aurait été avec cette méthode. L'implémentation de la boucle for a donc dû se faire jusqu'à la passe CodeTam.

Une instruction "Pour" a donc été ajoutée dans l'Ast, combinant les comportements des instructions "Declaration", "Affectation" et "TantQue". Pour la phase Tds une tds est créée pour analyser le code de la boucle en prenant en compte la variable de boucle. Pour la phase de typage, les types sont vérifiés comme ils le sont pour des Affectation, Declaration et boucle TantQue.

Enfin le code TAM de la boucle for est assez spécial car la place mémoire pour la variable de boucle est réservée juste avant la boucle, puis libérée juste après pour garantir que la variable est locale à la boucle.

6 Les Prototypes

Les prototypes sont une souplesse que peut offrir le compilateur pour le programmeur au même titre que les types nommées, mais n'entraîne, là encore, pas de modification sur la génération du code. Nous avons décidé comme ce qui est suggéré à la fin de l'énoncé du projet, l'ajout des types nommés au niveau des prototypes. Nous les avons appelé "types définis" car la grammaire des prototypes modélise un programme comme des *définitions* qui peuvent être des prototypes ou des fonctions (également des types définis) et entre les définitions, un bloc qui désigne le "main". *L'AST Syntax* a donc subit cette modification et la passe TDS se charge d'ajouter les informations des fonctions définis par les prototypes, de même que les types définis. Passé la passe TDS, nous avons voulu garder la modélisation originale d'un programme car effectivement un programme n'est fondamentalement qu'une liste de fonctions ainsi qu'un bloc principal. l'analyse du programme de la passe TDS se charge donc de rétablir la structure original enfin d'éviter de modifier les futures passes. La deuxième et dernière passe affectée est celle de typage car il est nécessaire de vérifié que d'une part les prototypes soient bien implantés et d'autre part que leurs implantations respectent les signatures de leurs prototypes.

7 Conclusion

Le projet a été réalisé dans sa globalité, l'ajout supplémentaire des types définis permet de mieux factorisé le code Rat. Nous l'avons suffisamment manipulé lors des séances de TP que nous n'avons pas eu des séances dédiées au projet. Les difficultés rencontrés n'ont pas été nombreuses, la mise en place de l'intégralité des passes ainsi que l'ajout des pointeurs en sont les seules. Nous sommes satisfaits de notre travail, nos tests couvrent majoritairement l'ensemble des cas de base, nous sommes donc sûrs que le travail rendu est de qualité, bien qu'il reste encore énormément de travail pour faire du langage Rat un véritable langage de programmation.

Sans compter les erreurs non traitées comme le déréférencement de pointeurs et celles des indices des tableaux, il serait intéressant d'y ajouter les caractères, les réels, l'ensemble des opérateurs de comparaison pour enrichir la génération de code, mais également y ajouter la partie générique grâce à des modules ou encore la compilation de plusieurs fichiers pour le compilateur Rat.