DOCUMENTATION DE CONCEPTION

Hassou Karim
Boussemid Youssef
Jaoudar Reda
Ait Taleb Aymane
Bendou Safouane
Groupe 46

January 2022

Contents

1 Etape A **3**
- 1.1 Analyse lexicale . . . 3
- 1.2 Analyse syntaxique . . . 3

2 Etape B (Analyse Contextuelle) **3**

3 Etape C (Génération de code) **4**

4 Classes ajoutées **4**
- 4.1 Classe New . . . 4
- 4.2 Classe Null . . . 5
- 4.3 Classe Condition . . . 5
- 4.4 Classe ListDeclMethod . . . 5
- 4.5 Classe AbstractDeclMethod . . . 6
- 4.6 Classe DeclMethod . . . 6
- 4.7 Classe MethodBody . . . 6
- 4.8 Classe ListDeclField . . . 6
- 4.9 Classe AbstractDeclField . . . 6
- 4.10 Classe DeclField . . . 7
- 4.11 Classe ListDeclParam . . . 7
- 4.12 Classe AbstractDeclParam . . . 7
- 4.13 Classe DeclParam . . . 8
- 4.14 Classe Selection . . . 8
- 4.15 Classe This . . . 8

5 Choix d'algortihmes et de structures de données **9**
- 5.1 Opération booléennes . . . 9
- 5.2 Environnement des types . . . 9
- 5.3 Table de méthodes . . . 9
- 5.4 Gestion des registres . . . 10

1 Etape A

L'analyse lexicale et l'analyse syntaxique nous permet, à partir d'un code Deca lexicalement et syntaxiquement correct, d'obtenir un arbre abstrait qui va être utilisé et décoré par la suite lors de l'analyse contexctuelle.

1.1 Analyse lexicale

Les tokens du langage Deca sont définis dans le fichier DecaLexer.g4 contenu dans le répertoire **src/main/antlr4/fr /ensimag/deca/syntax**. On y définie les lexèmes du langage Deca.

1.2 Analyse syntaxique

On définie la syntaxe du langage Deca dans le fichier DecaParser.g4 contenu dans le même répertoire que DecaLexer.g4 à savoir **src/main/antlr4/fr/ensimag/ deca/syntax**. Ce fichier a été énormément enrichi lorsqu'on a défini la syntaxe du langage Deca avec Objet et ce en s'inspirant de ce qui nous a été fourni précédemment dans ce même fichier. Des classes ont été également ajoutées dans le répértoire **src/main/java/fr/ensimag/deca/tree** pour permettre d'instancier et d'afficher les noeuds et les feuilles ajoutées par la syntaxe nouvellement définie.

2 Etape B (Analyse Contextuelle)

Cette étape prend comme paramètre d'entrée l'arbre abstrait fourni par l'étape A. Les feuilles et les noeuds de cet arbre sont des classes contenues dans le répertoire

src/main/java/fr/ensimag/deca/tree. Ces classes sont toutes des classes filles de la classe Tree. L'analyse contextuelle d'un programme Deca se fait en 3 passes. Ces passes sont faites par le biais de la fonction verifyProgram de la classe Program. On parcours dés lors l'arbre abstrait et on fait des vérification contextuelles sur les différents noeuds de l'arbre tout en le décorant.

3 Etape C (Génération de code)

Cette étape consiste à prendre l'arbre décoré comme paramètre d'entrée, et à générer à partir de celui-ci du code assembleur ima. La génération de code se fait en deux passes et ce par le biais de la fonction codeGenProgram de la classe Program. Chaque classe du packetage **src/main/java/fr /ensimag/deca/tree** contient une ou plusieurs fonctions de type codeGenXXX qui permettent de générer du code assembleur.

4 Classes ajoutées

Dans cette section, on listera les classes qu'on a ajoutées lors de notre implémentation (c à d les classes hors celles qui nous ont été fournies lors du début du projet) et on détaillera leurs dépendances.

4.1 Classe New

Il s'agit d'un noeud de l'arbre abstrait. Cette classe étend la classe AbstractExpr. Elle permet de gérer l'analyse contextuelle (Vérifier par exemple que l'identifiant précédé

par new désigne bien une classe...) et la génération de code de l'expression new Identifiant().

4.2 Classe Null

Il s'agit là d'une feuille de l'arbre abstrait. Elle permet d'instancier le littéral NULL dans l'arbre abstrait et de gérer l'analyse contextuelle et la génération de code de ce littéral.

4.3 Classe Condition

Il s'agit là d'une interface qu'on a ajoutée afin d'effectuer un polymorphisme de fonction. Elle est contenue dans le packetage **src/main/java/fr/ensimag/deca/tree**. Cette interface contient une seule et unique fonction (codeGenCond). Elle est la classe parent de toute expression dont le type est un booléen (à savoir les expression booléennes And Or et Not, les litteraux booléens BooleanLitteral et les appels de méthodes dont le type de retour est un booléen MethodCall). La fonction codeGenCond prend un paramètre de plus que les autres fonctions de type codeGenXXXX à savoir un Label ce qui permet de générer le code de ces expressions et de gérer les branchement à faire.

4.4 Classe ListDeclMethod

C'est une classe qui étend Treelist AbstractDeclMethod. Elle est contenue dans le packetage **src/main/java/fr /ensimag/deca/tree**.

4.5 Classe AbstractDeclMethod

C'est une classe abstraite contenue dans le packetage **src/main/java/fr/ensimag/deca/tree**. Elle étend la classe Tree. Elle a été conçue pour qu'elle soit héritée par la classe DeclMethod.

4.6 Classe DeclMethod

Cette classe permet d'effectuer l'analyse contextuelle et la génération de code des déclarations de méthode. Elle étend comme mentionné précédemment la classe AbstractDeclMethod. Elle est également contenue dans le packetage **src/main/java/fr/ensimag/tree**.

4.7 Classe MethodBody

Cette classe contenue également dans le packetage **src/main/java/fr /ensimag/tree** permet d'effectuer une analyse contextuelle et de générer le code assembleur du corps des méthodes. Elle étend directement la classe Tree.

4.8 Classe ListDeclField

C'est une classe qui étend Treelist AbstractDeclField. Elle est contenue dans le packetage **src/main/java/fr /ensimag/deca/tree**.

4.9 Classe AbstractDeclField

C'est une classe abstraite contenue dans le packetage **src/main/java/fr/ensimag/deca/tree**. Elle étend la

classe Tree. Elle a été conçue pour qu'elle soit héritée par la classe DeclField.

4.10 Classe DeclField

Cette classe permet d'effectuer l'analyse contextuelle (Vérifier qu'il n'y a pas de doublons dans la déclaration des champs, que les types utilisés pour déclarer les champs sont des types qui existent dans l'environnement des types etc... et décorer l'arbre en donnant une définition aux champs et un index qui correspondra à la position du champs dans le tas...) et la génération de code des déclarations de champs dans une classe. Elle étend comme mentionné précédemment la classe AbstractDeclField. Elle est également contenue dans le packetage **src/main/java/fr /ensimag/deca/tree**.

4.11 Classe ListDeclParam

C'est une classe qui étend Treelist AbstractDeclParam. Elle est contenue dans le packetage **src/main/java/fr /ensimag/deca/tree**.

4.12 Classe AbstractDeclParam

C'est une classe abstraite contenue dans le packetage **src/main/java/fr/ensimag/deca/tree**. Elle étend la classe Tree. Elle a été conçue pour qu'elle soit héritée par la classe DeclParam.

4.13 Classe DeclParam

Cette classe permet d'effectuer l'analyse contextuelle de la déclaration des paramètres dans une méthode. (Vérifier qu'il n'y a pas de doublons dans la déclaration de paramètres etc...) Elle étend comme mentionné précédemment la classe AbstractDeclParam. Elle est également contenue dans le packetage **src/main/java/fr/ensimag/deca/tree**.

4.14 Classe Selection

Cette classe contenue dans le packetage **src/main/java/fr /ensimag/deca/tree** permet d'effectuer une analyse contextuelle (Vérifier si l'expression sur laquelle on effectue une sélection est bien de type ClassType, si l'objet de la classe en question a bien comme champs celui utilisé dans la sélection...) et de générer le code des sélections de champs. Elle étend la classe LValue.

4.15 Classe This

Cette classe est contenue dans le packetage **src/main/java/fr /ensimag/deca/tree**. Elle permet d'effectuer une analyse contextuelle de l'utilisation de l'identifiant this (Vérifier par exemple que this n'est pas utilisé dans la partie main du programme...) et de générer le code assembleur de celui-ci.

5 Choix d'algortihmes et de structures de données

5.1 Opération booléennes

Les opérations booléénnes sont évaluées de gauche à droite comme spécifié dans le cahier de charge. Par contre, elles ne sont pas faites de manière paresseuse. Pour l'opération And, on a remarqué que le résultat de l'expression A and B est équivalent au résultat de la multiplication de A et de B (0 = faux et 1 = vrai). Pour l'opération Or, on a remarqué que le résultat de l'expression A or B est équivalent au résultat de la comparaison de la somme de A et de B (0 = faux et 1 = vrai) et de 1 (A or B est vrai si la somme de A et de B est supérieure ou égale à 1).

5.2 Environnement des types

L'environnement des types est défini dans la classe DecacCompiler située dans le packetage **src/main/java/fr/ensimag/deca**. C'est un objet de la classe EnvironmentExp et contient les définitions des types prédéfinis et de toutes les classes définies dans un programme Deca donné.

5.3 Table de méthodes

Dans notre implémentation de la table des méthodes, on a ajouté un attribut à la classe ClassDefinition qui nous permettait de connaître l'adresse de début de la table de méthode de chaque classe. Ceci s'est avéré utile pour remplir la première case de la table des méthodes. La génération de la table de méthode d'une classe donnée se

fait de manière récursive grâce à la fonction generation de la classe DeclClass du packetage **src/main/java/fr/ensimag/deca/tree**. Celle-ci place récursivement les méthodes de toutes les classes parents de la classe en question dans la table des méthodes avant de placer les méthodes de la classe en question dans cette dernière.

5.4 Gestion des registres

On a ajouté deux attributs à la classe Registre du packetage **fr.ensimag.ima.pseudocode.Register**. Un attribut isFull qui indique si le registre est occupé et un attribut isPushed qui indique si le contenu du registre a été empilé pour le libérer temporairement. Une fonction getEmptyReg a également été implémentée. Elle retourne le registre avec l'indice le plus petit pouvant être utilisé. Si tout les registres sont pleins (isFull = true pour tout les registres), on empile le contenu du dernier registre, on met l'attribut isPushed à true et on retourne son indice. Les attributs isPushed et isFull des registres sont modifiés adéquatement dans le corps des méthodes.(si un registre n'est plus utilisé et que isPushed est false, on met l'attribut isFull à false etc...)