Manuel Utilisateur

Projet Génie Logiciel

Fabien Galzi, Abdelmlek Mrad, Lisa Cornu,

Matéo Bedel, Eliot Clerc

Groupe GL 31

20/01/20025

Sommaire

1. Introduction

Ce manuel utilisateur est destiné à aider les utilisateurs du compilateur Deca, il s'adresse à des utilisateurs ayant déjà une connaissance des spécifications détaillées du langage Deca et souhaitant exploiter pleinement les fonctionnalités du compilateur. Notre implémentation de ce compilateur inclût également l'extension OPTIM qui vise à optimiser le code généré par le compilateur aussi bien en termes de performance que de consommation de ressources. Ce document décrit les différentes options de compilations du compilateur, les erreurs qui peuvent être retournées par les différentes parties, l'utilisation de l'extension et les limitations du compilateur.

2. Options de compilations

2.1 Génération du fichier assembleur

Soit un fichier écrit en Deca file.deca la commande pour générer le fichier assembleur correspondant est :

```
decac [[-p | -v] [-n] [-r X] [-d]* [-P] [-w] file.deca]
```

Par exemple pour générer un fichier assembleur sans options de compilation particulières la commande est :

```
decac file.deca
```

Si la compilation de ce fichier ne retourne pas d'erreur, alors un fichier `file.ass` est généré dans le répertoire dans lequel est situé `file.deca`

2.2 Options de compilations

La syntaxe d'utilisation de l'exécutable `decac` étant la suivante :

```
decac [[-p | -v] [-n] [-r X] [-d]* [-P] [-w] ...] | [-b]
```

Voici la spécification des différentes options de compilation

`-b` (banner) : affiche une bannière indiquant le nom du groupe

`-p` (parse) : arrête decac après l'étape de construction de l'arbre, et affiche la décompilation de ce dernier (i.e. s'il n'y a qu'un fichier source à compiler, la sortie doit être un programme deca syntaxiquement correct)

`-v` (verification) : arrête decac après l'étape de vérifications (ne produit aucune sortie en l'absence d'erreur)

`-n` (no check) : supprime les tests à l'exécution spécifiés dans les points 11.1 et 11.3 de la sémantique de Deca.

`-r X` (registers) : limite les registres banalisés disponibles à R0 ... R{X-1}, avec 4 <= X <= 16

`-d` (debug) : active les traces de debug. Répéter l'option plusieurs fois pour avoir plus de traces.

`-P` (parallel) : s'il y a plusieurs fichiers sources, lance la compilation des fichiers en parallèle (pour accélérer la compilation)

3. Erreurs

3.1 : Lexeur

Le lexer, si il ne reconnaît pas un token comme un token valide du langage Deca affiche l'erreur suivante :

```
<chemin vers le fichier .deca>:<ligne du token non reconnu>:<colonne du token non reconnu>: token recognition error at: <token non reconnu>
```

Par exemple pour le fichier Deca invalide suivant

```
{

    int µ;

}
```

l'erreur affichée est la suivante :

```
…/strange_identifier.deca:11:8: token recognition error at: 'µ'
```

Le lexeur ne peut pas retourner d'autres erreurs.

3.2 : Parseur

Les erreurs que le parseur peut retourner sont les suivantes :

- Dans le cas ou le parseur n'arrive par à parser le programme Deca compilé :

  ```
  <chemin vers le fichier .deca>:<ligne>:<colonne>: no viable alternative at input <token problématique>
  ```

  Par exemple pour le programme Deca suivant :

  ```
  idf _autre_idf

  {

  }

  (

  )
  ```

  l'erreur retournée est :

  ```
  …/simple_lex.deca:11:0: no viable alternative at input 'idf'
  ```

- Dans le cas où un programme Deca tente d'assigner une valeur non représentable sur 31 bit à une variable :

  ```
  <chemin vers le fichier .deca>:<ligne>:<colonne>: Number overflow error
  ```

  Par exemple, pour le programme suivant :

  ```
  {

      int x = 2147483648;

  }
  ```

  L'erreur retournée est :

  ```
  …/int_overflow.deca:10:12: Number overflow error
  ```

3.3 Vérification contextuelle

Il existe quelques erreurs qui ne peuvent pas être atteintes (l'erreur est relevée avant dans le code), ainsi la couverture n'est pas parfaite.

1ère Passe :

- Erreur si la super classe n'est pas de type classe : `"The superClass is not a class"`
- Erreur de redéfinition d'une classe : `"This class has already been defined [nom de la classe]"`

2ème Passe :

- Erreur relevée si un champ est de type void : `"The type of this field is void."`
- Erreur de redéfinition d'un champ : `"The field as already been declared before."`
- Erreur de redéfinition d'une méthode dans la même classe : `"The method as already been declared before."`
- Erreur de redéfinition d'une méthode héritée avec un mauvais type : `"You overwrite a method without good type"`
- Erreur de redéfinition d'une méthode héritée avec des signatures différentes (nombre de paramètres différent) : `"They are not the same number of parameters : [parentClassName] for parent class [className] for class."`
- Erreur de redéfinition d'une méthode héritée avec des signatures différentes (type des paramètres différent) : `"The signatures are not the same : [typeParam(i)] should be [typeParamParent(i)]"`
- Erreur de redéfinition d'une méthode : `"The method as already been declared before."`

3ème Passe :

- Vérification de la rvalue pour le corps des classes :
    - Erreur de type, la classe n'étant pas une sous-classe du type demandé : `"The classType [type de la classe] is not a subclassType of [type demandé]"`
    - Erreur de type non compatible (int en string,...) : `"They are not compatible (not same type or float->int) [type] is not [type demandé]"`
- Vérification du corps des méthodes :
    - Vérification des paramètres :
        - Erreur du type du paramètre :`"Type is not defined" ou "It's void type"`
        - Erreur de redéfinition : `"The Param has already been defined before."`
    - Vérification du corps de la méthode :
        - Vérification des Variables et Instructions (voir au-dessous)
- Vérification du main :
    - Vérification du type d'une déclaration de variable : `"type is void"`
    - Erreur de redéfinition d'une variable : `"The type as already been defined for the variable [nomVariable]"`
    - Vérification Initialization (déjà vu pour les champs)
    - Vérification instructions : dans le AbstractExpr :
        - Vérification d'une expression : décoration pour les terminaux
        - Si mauvais type : `"This is not inst type"`

- Vérification des expressions :
    - Opération Arithmétique : Les types ne sont pas compatibles pour une opération : `"Both are not float or int"`
    - Opération booléen : Les types ne sont pas des booléens : `"Both are not boolean : [typeGauche] and [typeDroite]"`
    - Opération de comparaison : Les types sont différents (int/float ou booléens) : `"Both are not same type : [typeGauche] and [typeDroite]"`
    - Assignation : vérification des expressions avec convfloat,.. (voir dessous)
    - Type Booléen : déclaration d'un nouveau type booléen
    - Cast : vérification de type et expression, erreur de conversion avec types incompatibles : `"You cant convert [type de l'expression] into [type du cast]"`
    - ConvFloat : nouvelle définition d'un type float
    - FlaotLiteral : nouvelle définition d'un type float
    - Identifier : Erreur : pas une expression : `"This is not an expression."` et il n'existe pas de définition pour ce nom : `"There is no definition for this name : [nom]"`
    - InstanceOf : L'expression n'est pas de la "même famille de types": `"You cant do Instanceof [type expression]"`
    - IntLiteral : nouvelle définition d'un type int
    - MethodCall :
        - vérification des expressions
        - Pas le même nombre de paramètres : `"They are not the same number of param"`
        - Paramètre de la méthode ne sont pas corrects : `"ParamType is different than type you passed as argument"`
        - Définition inconnue de la méthode : `"The method was not defined before [nom méthode]"`

- Modulo : vérification que les types des opérandes sont des int : `"You cant divide things that are not int"`
- New : vérification que le type est une classe : `"Type is not a class type"`
- Not : vérification d'un type booléen : `"Operand is not boolean"`
- Null : déclaration d'un nouveau type Null
- ReadFloat : déclaration d'un nouveau type float
- ReadInt : déclaration d'un nouveau type int
- Selection :
    - Le type n'est pas une classe : `"The class is not defined"`
    - la classe n'est pas définie : `"You cant get a protected type"`
    - le type n'est pas le même que celui de la classe : `"You cant get visibility because your types dont correspond"`
    - le type n'est pas le même que celui de la classe : `"Subtype(expr type) is not a subtype of super class(currentClass)"`
    - le type n'est pas dans le field type : `"Subtype(currentClass) is not a subtype of super class(field Class)"`
- StringLiteral : déclaration d'un nouveau typ string
- This : vérification qu'on est dans une classe : `"Class is not defined"`
- UnaryMinus : vérification du bon type : `"You are trying to make something negative but it's not legal on this type"`

4. Manuel de l'extension

- ### Utilisation de l'extension

Il n'y a pas de manière particulière d'utiliser l'extension car elle est directement utilisée dans le compilateur sans avoir d'options en paramètres. (Optimisation dans tous les cas : détection code mort,...)

- ### Limitations de l'extension

Nous n'avons pas réussi à implémenter la forme SSA, nous avons donc optimisé le code assembleur le plus possible avec le temps imparti.

5. Limitations du compilateur

- **Partie B :**

Il reste des erreurs qui ne sont pas relevées, et probablement quelques erreurs concernant la partie avec objet notamment au niveau des casts, méthodes,... Les programmes devant compiler normalement semblent fonctionner, cependant des programmes ne devant pas passer cette partie peuvent passer outre les exceptions. Il faut donc être méfiant si un programme peut être mal écrit contextuellement. L'utilisateur peut donc utiliser le compilateur pour tout programme correct. Les limitations se trouvent plutôt dans le cas où le programme ne devrait pas compiler.

- **Partie C :**

  Pour ce qui est de la génération de code, les expressions utilisant instanceof et les cast d'objets ne sont pas fonctionnelles. Le code de l'expression instanceof est écrit mais nous n'avons pas eu le temps de le débugger lors du test d'instanceof avec de l'héritage, tandis que nous n'avons eu le temps de rien faire pour le cast. L'expression instanceof "direct" (A a; puis a instanceof A;) est cependant fonctionnelle.

  De plus, le débordement de la pile est géré de manière fixe avec une grande valeur de TSTO, sans calculer au fur et à mesure la taille de pile nécessaire à l'utilisation des fonctions.

  Enfin, nous avons remarqué un bug qui apparait occasionnellement lorsque l'on affiche le résultat d'une méthode, mais nous n'avons pas eu le temps d'investiguer ce bug.