Compilateur Decac
Manuel Utilisateur

pour machine virtuelle IMA et architecture ARMv6

TEIMUR ABU ZAKI, ADRIEN BOUCHET, TROY FAU, PAUL MARTHELOT, HUGO ROBERT

Introduction

Le compilateur Decac est destiné aux utilisateurs du langage Deca, d'après sa dernière spécification diffusée par l'Ensimag. Ce manuel suppose la connaissance de cette spécification par l'utilisateur. Decac produit des fichiers *.ass* directement exécutable par la machine virtuelle IMA, ou des fichiers *.s* pour un *cross compiler* ARMv6.

Lancement et Options

Le compilateur est lancé dans un terminal avec la commande `decac <fichier deca> <...>`. Il peut prendre plusieurs fichiers en arguments ; ceux-ci doivent obligatoirement posséder le suffixe `.deca`. Ils seront compilés en fichier *.ass* dans le même répertoire.

La commande `decac` utilisée sans argument affiche les options disponibles. Ces options sont :

-b Affiche la bannière de l'équipe de développement de Decac. Doit être utilisé sans argument.

-p Décompilation du fichier source, c'est-à-dire reconstruction du code source à partir de l'arbre généré lors de la phase d'analyse syntaxique de la compilation. Ce code reconstruit est affiché dans le terminal, dans l'ordre des fichiers passés en arguments.

-v Arrête le compilateur après l'étape de vérification, qui a lieu après la phase d'analyse contextuelle. Ne produit aucune sortie hors message d'erreur.

-n Supprime les tests à l'exécution spécifiés dans les sections 11.1 et 11.3 de la sémantique du Deca.

-r X Limite les registres disponibles pour la machine IMA à `R0...R{X-1}`, avec $4 \leq X \leq 16$. Par défaut, les 16 registres sont disponibles.

-d Active les traces de debug.

-P Si plusieurs fichiers sources sont passés en arguments, lance la compilation de ceux-ci en parallèle.

-a Au lieu d'un fichier *.ass*, Decac génère un fichier *.s* pour un compilateur ciblant une architecture ARMv6.

-w Affiche les messages d'avertissement lors de la compilation.

NB : Les options incompatibles sont :
— **p** et **v**
— **b** et toute autre option
— **a** et **r X**

Les fichiers sources et les options peuvent être entrés sur la ligne de commande dans n'importe quel ordre.

Erreurs à la compilation

Decac détecte toutes les erreurs statiques d'un programme écrit en Deca. Le compilateur s'arrête dès la première erreur et affiche un message d'erreur détaillant celle-ci, avec sa position dans le fichier en cours de traitement.

Erreurs lexicales

Les caractères reconnus par l'analyseur lexical (*lexer*) du Deca en dehors des chaînes de caractères sont un sous-ensemble des caractères de la norme ASCII, dérivée de la norme ISO/CEI 646. Dans les chaînes de caractères, tous les caractères sont utilisables. Un caractère non reconnu par le *lexer* déclenche une erreur et l'arrêt de la compilation.

Message d'erreur : *token recognition error at :* <premier caractère non reconnu>

Erreurs syntaxiques

Un programme Deca syntaxiquement incorrect lève des exceptions durant la phase d'analyse syntaxique. Les trois premières sont définies par ANTLR [1], le générateur d'analyseur lexical et syntaxique utilisé pour développer ces parties de Decac.

- **Lexème manquant** Message d'erreur :
  - *missing* <lexème manquant> *at* <lèxeme lu>
- **Lexème bloquant une décision du *parser*** Message d'erreur :
  - *no viable alternative at input* <lexème lu>
- **Lexème non-attendu lors de la dérivation** Messages d'erreur :
  - *mismatched input* <lexème lu> *expecting* <ensemble de lexèmes attendus>
  - *extraneous input* <lexème lu> *expecting* <lexème attendu>
- **Partie gauche d'une affectation n'est pas une *lvalue*** Message d'erreur :
  - *left-hand side of an assignment is not an lvalue.*
- **Dépassement *int*** Le type *int* est codé sur 32 bits. Une valeur entière supérieure à 2 147 483 647 lève une exception. Message d'erreur :
  - *int literal out of expected range.*
- **Dépassement *float*** Le langage Deca suit la norme IEEE-754 pour les nombres flottants. En 32 bits, une valeur supérieure à environ 3.4028235×10^{38} lève une exception. Message d'erreur :
  - *float literal out of expected range.*
- **Arrondi *float* à zero** Message d'erreur :
  - *float literal underflow.*

1. https ://www.antlr.org/

Erreurs contextuelles

Un programme Deca bien formé syntaxiquement peut néanmoins être « mal typé » : il contient des erreurs qui enfreignent les règles sémantiques du langage. Ces erreurs lèvent des exceptions durant la phase d'analyse contextuelle de Decac.

- **Non-déclaration d'une variable ou d'une méthode** Message d'erreur :
  - *(RULE 0.1) Variable|Method* <identifiant> *has not been declared.*
- **Utilisation d'un type non-déclaré** Message d'erreur :
  - *(RULE 0.2) Type* <identifiant> *does not exist.*
- **Déclaration d'une classe déjà déclarée** Message d'erreur :
  - *(RULE 1.3) Classe* <identifiant> *has already been declared.*
- **Déclaration d'un champ déjà déclaré** L'erreur consiste à redéclarer dans la classe courante. Message d'erreur :
  - *(RULE 2.4) Method or field* <identifiant> *has already been declared.*
- **Utilisation illégale du type *void*** Déclaration d'un champ de type *void*, d'un paramètre formel de méthode de type *void*, ou d'une variable de type *void*. Messages d'erreur :
  - *(RULE 2.5) Field cannot be of type void : void* <identifiant>
  - *(RULE 2.9) Parameter cannot be of type void : void* <identifiant>
  - *(RULE 3.17) Variable cannot be of type void : void* <identifiant>
- **Tentative de surcharge d'une méthode par un champ** Message d'erreur :
  - *(RULE 2.5) Illegal override of* <identifiant> *: method → field.*
- **Déclaration d'une méthode déjà déclarée** Message d'erreur :
  - *(RULE 2.6) Method or field* <identifiant> *has already been declared.*
- **Tentative de surcharge d'un champ par une méthode** Message d'erreur :
  - *(RULE 2.7) Illegal override of* <identifiant> *: field → method.*
- **Surcharge de méthode erronée** La nouvelle méthode n'a pas la même signature ou le même type de retour que la méthode d'origine. Message d'erreur :
  - *(RULE 2.7) Invalid method override. Signature of* <identifiant> *is* <signature> *, return type is* <type>
- **Utilisation erronée de *return*** Utilisation du mot-clé *return* dans le programme principal, ou dans une méthode de type *void*. Messages d'erreur :
  - *(RULE 3.24) 'return' statement inapplicable in main program.*
  - *(RULE 3.24) 'return' statement inapplicable in method of return type void.*
- **Affectation mal typée** Concerne les initialisations, affectations et instructions *return* mal typées. Message d'erreur :
  - *(RULE 3.28) Incompatible type assignment. Lvalue is of type* <type attendu>, *assigned value is of type* <type *rvalue*>

- **Condition non-booléenne** La condition d'une structure de contrôle *While* ou *If-Then-Else* n'est pas de type booléen. Message d'erreur :
  — *(RULE 3.29) Condition must be a boolean expression :* (<type effectif>)
- **Affichage erronée** Tentative d'affichage d'un type ne pouvant être affiché. Message d'erreur :
  — *(RULE 3.31) Cannot print* <type> *type.*
- **Opérande mal typé** Erreur de typage dans une opération arithmétique, dans une opération de comparaison, dans une opération booléenne, dans une opération unaire, ou dans une opération *instanceof*. Dans une opération binaire, si les deux opérandes sont mal typés, le compilateur lève l'exception au premier opérande (et indique le type de cet opérande). Messages d'erreur :
  — *(RULE 3.33) Illegal operand type : is* <type> *, int or float required.*
  — *(RULE 3.33) Incompatible operand types.*
  — *(RULE 3.33) Illegal operand type.* [Une classe dans une opération de comparaison]
  — *(RULE 3.33) Illegal operand type : is* <type> *, boolean required.*
  — *(RULE 3.37) Illegal operand type : is* <type effectif> *,* <type attendu> *required.*
  — *(RULE 3.40) Illegal operand type.*
- **Transtypage interdit** En cas d'incompatibilité pour le transtypage (voir spécifications du langage). Message d'erreur :
  — *(RULE 3.39) Illegal cast.*
- **Appel de méthode erroné** Utilisation d'un nom de champ valide comme nom de méthode, ou appel de méthode à partir d'une expression en partie gauche qui n'est pas une instance de classe, ou appel avec une liste d'arguments incompatible. Messages d'erreur :
  — *(RULE 3.41) Invalid method call :* <identifiant> *is not a method.*
  — *(RULE 3.71) Method selection applied to expression of non-class type :* (<type>).<méthode>
  — *(RULE 3.72) Invalid argument list : signature of* <identifiant> *is* <signature>
- **Instantiation erronée** Tentative d'initialisation d'une instance d'un type existant mais n'étant pas un type de classe. Message d'erreur :
  — *(RULE 3.42) New instance must be of class type : new* <type>()
- **Mauvaise utilisation de *this*** Utilisation de la référence *this* en dehors d'une définition de classe. Message d'erreur :
  — *(RULE 3.43) 'this' reference used outside of class scope.*
- **Sélection erronée** Sélection de champ à partir d'une expression en partie gauche qui n'est pas une instance de classe. Message d'erreur :
  — *(RULE 3.65) Field selection applied to expression of non-class type :* (<type>).<champ>

- **Sélection interdite** Sélection d'un champ protégé dans le programme principal, ou sélection d'un champ protégé dans une classe qui n'hérite pas de ce champ. Messages d'erreur :
  — *(RULE 3.66) Protected field is not visible in the current scope. Field* <champ> *declared in class* <classe>*, current scope is main program.*
  — *(RULE 3.66) Protected field is not visible in the current scope. Field* <champ> *declared in class* <classe>*, current scope is* <classe courante>
- **Sélection interdite** Sélection d'un champ protégé à partir d'une instance de classe qui n'est pas une classe enfant de la classe courante. Message d'erreur :
  — *(RULE 3.66) Selecting class is not a subclass of the current class. Selecting class is* <classe>*, current class is* <classe courante>

Erreurs à l'exécution

Un programme Deca qui compile et fournit un fichier *.ass* exécutable par la machine virtuelle IMA peut échouer à l'exécution. Ces cas d'échec sont définis dans la spécification du langage ; Decac les prend en compte en générant pour chacun d'eux un message d'erreur dans le fichier *.ass*. Lorsque l'erreur est rencontré au *runtime*, IMA peut alors afficher le message correspondant et arrêter le programme.

La spécification du Deca n'impose pas de comportement au programme en cas d'accès à une variable non initialisée ou d'exécution de méthode écrite en code assembleur incorrecte/incompatible. En conséquence, le compilateur Decac ne traite pas ces cas, et le comportement du programme est indéfini.

- **Division par zéro** Message d'erreur :
  — *ERROR : Division by zero*
- **Transtypage impossible** Message d'erreur :
  — *ERROR : Illegal cast*
- **Débordement lors d'un calcul flottant** Message d'erreur :
  — *ERROR : Float arithmetic overflow*
- **Débordement de pile** Message d'erreur :
  — *ERROR : Stack overflow*
- **Problème à la lecture de données ou à l'affichage** Message d'erreur :
  — *ERROR : Input/Output*
- **Accès à une réference nulle** Message d'erreur :
  — *ERROR : Null dereferencing*
- **Sortie de méthode sans *return*** Dans une méthode avec un type de retour non-*void*. Message d'erreur :
  — *ERROR : No return statement*

Avertissements

Avec l'option `-w`, Decac affiche un message d'avertissement s'il détecte certaines anomalies dans le code. Ces anomalies ne sont pas considérées par la spécification du langage comme des erreurs devant arrêter la compilation, mais peuvent provoquer une erreur à l'exécution. Decac continue à compiler après l'avertissement et un fichier *.ass* sera produit si aucune erreur est détectée par ailleurs.

- **Division par la constante zéro** Message d'avertissement :
  — *Division by zero.*
- **Absence de *return*** S'il n'y a aucune instruction dans une méthode de type non-*void*, ou si la dernière instruction d'une telle méthode n'est pas une instruction de retour. Messages d'avertissement :
  — *No return statement in non-void method.*
  — *Last instruction of non-void method is not a return statement.*

Extension ARM : Decac -a

Utilisation

La compilation d'un programme Deca en fichier *.s* se fait avec la commande `deca -a <fichier deca>`. Pour ensuite compiler le fichier *.s* généré en exécutable, deux outils sont nécessaires :

- **Un cross compilateur :** Decac cible le compilateur *gcc-arm-linux-gnueabihf*.
- **Un simulateur :** nécessaire pour exécuter sur PC executer un fichier binaire 32-bit ARM. L'équipe Decac conseille *qemu-user*, qui a été utilisé pendant le développement du compilateur.

Installation des outils

Sur des machine Linux Debian (Kali/Ubuntu), les deux outils s'installent avec les commandes suivantes :

- **Un cross compilateur :** *sudo apt-get install gcc-arm-linux-gnueabihf*
- **Un simulateur :** *sudo apt-get install qemu-user*

Exécution

Pour exécuter le programme Deca en ARMv6, Decac est fourni avec un script shell qui utilise les outils susmentionné pour compiler un fichier *.s* et l'exécuter sur le simulateur. Ce fichier se trouve sur le chemin **src/test/script/ARM-exec.sh**.

Périmètre de l'extension

Pour l'extension ARM, l'équipe Decac a implémenté le sous-ensemble du langage dit « Sans-Objet », moins l'affichage des flottants au format hexadécimal. Toutes les fonctions d'affichage `Print<suffix>` affichent les flottants au format décimal.

Au niveau de la gestion d'erreurs, le fichier *.s* généré gère bien les erreurs dans le cas d'une division par zéro, mais pas les cas de débordement en calcul flottant, de débordement de pile, ou de données mal typées fournies en entrée par l'utilisateur aux fonctions `readInt` et `readFloat`.

Hormi ces exceptions, le comportement du programme est équivalent au comportement du programme en « IMA Sans-Objet ».