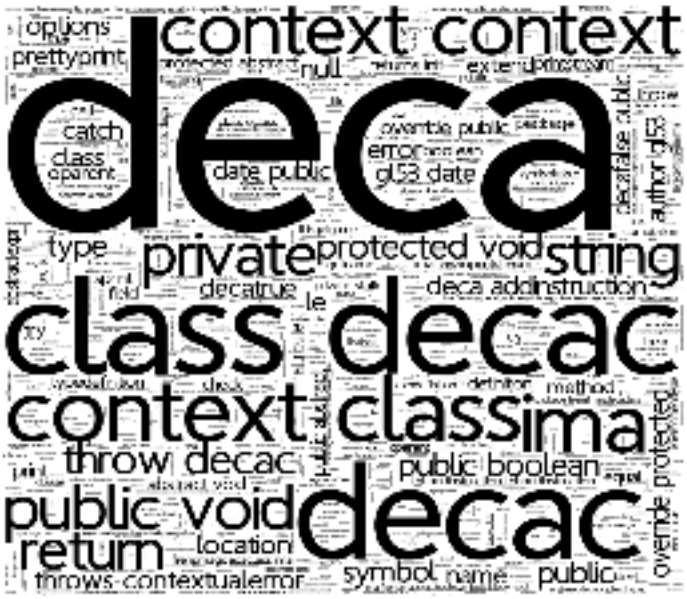
Manuel Utilisateur



GL53

Ayman ABOUELOULA

Nabil BENSRHIER

Kacem JEDOUI

Omar BENCHEKROUN

Redouane YAGOUTI

SOMMAIRE

1. Utiliser deca

1.1. A propos de deca

Deca est un langage de programmation permettant de fournir des fonctionnalités avancées de langages de programmations orientés objet, il ne nécessite des threads ou de garbage collector donc la performance est liée systématiquement au code utilisé.
Cette documentation, présentera les limitations propre à notre compilateur, les messages d'erreur retournés à l'utilisateur (lexicographique, syntaxique, contextuelle, et sur la génération de code), une description de l'extension Trigo et les limitations liée (quantification de l'erreur).

1.2. Configuration requise

Pour utiliser notre compilateur, il faut avoir l'exécutable decac: Permet de compiler et ima préalablement installé sur l'ordinateur, et un fichier .deca où figure le code à compiler et exécuter.

2. Compilation/Exécution avec decac

Pour lancer la compilation il faut donc lancer la commande depuis le terminal :

decac [Options] [_fichier_.deca]

Notons qu'il y a la possibilité de compiler plusieurs fichiers (voir l'option -P pour amélioration des performances) et que le fichier sur lequel on passe la commande, doit être un ".deca", si la compilation est réussie: Pas d'erreurs lexicales, syntaxiques ou contextuelles, un fichier ".ass" doit être généré, dans le même répertoire du fichier .deca
On peut maintenant exécuter notre fichier en lançant la commande :

ima [_fichier_.ass]

3. Un point sur les options deca

Le programme fourni est un compilateur Deca complet, qui en plus de la fonction principale de compilation d'un fichier .deca, permet à l'utilisateur de débugger le programme, exécuter les différentes étapes de compilation (pour les utilisateurs expérimentés), et compiler en parallèle plusieurs fichiers .deca. Les options implémentées sont décrites ci-dessous.

3.1. -b (banner):

affiche simplement un banner contenant le nom de l'équipe

3.2. -p (parse):

décompilation et affichage du résultat dans la sortie standard

3.3. -v (vérification):

arrête la compilation dans l'étape de la vérification

3.4. -n (no check):

ignore les tests de débordement arithmétique et débordement mémoire

3.5. -r R_Max (registers):

limite les registre banalisés disponibles à R0...R_Max, avec le nombre de registres compris entre 4 et 16.

3.6. -d (debug) :

pour débugger le programme, répéter plusieurs fois pour avoir plus de traces.

3.7. -P (parallel):

lance la compilation de plusieurs fichiers en parallèle (à donner en arguments après decac, i.e: decac -P file1.deca file2.deca ..)

Notons que les options -p et -v sont incompatibles, et qu'il faut respecter la syntaxe decrite dans la section 2.

4. Limites de Decac

Deca a été défini précédemment comme étant un sous-langage de Java. Il présente pourtant nombreuses limites par rapport à celui ci, la liste suivante n'est pas exhaustive, et il faut se référer à [Sémantique] pour le fonctionnement complet du langage:

- absence de notion de classe abstraite ou interface
- obligation de définir les variables avant d'exécuter les instructions à l'intérieur du programme principal (main)
- absence de la notion de protected, public ou private dans le compilateur qu'on a implémenté
- les types de variables sont limités à : int, float, string, boolean et null

On présente également des limitations propres à notre compilateur, notamment :

- Le Cast ne marche que dans le sens int->float.
- L'instruction instanceof n'est pas supportée.
- mauvaise gestion du 'return' dans les méthodes: La méthode continue l'exécution des instructions après return.
- mauvaise gestion de la récursivité dans des cas complexes.
- la nécessité de 5 registres (au lieu de 4) dans des instances de tests extrêmes

Ceci dit, le compilateur présente d'une manière générale des bugs (constatés dans des fichiers .deca complexes par exemple) qui devront être corrigés par la suite.

4.1. Contextuelles

Les messages d'erreurs :

- Le type doit être une classe déjà définie ou un type prédéfini (int, float, boolean)

```
BadStringOperation.deca:2:4: Type "String" n'est pas un type prédéfini (règle 0.2)
```

- Le type doit être différent de void :

```
BadVoidUsage.deca:2:9: Type de l'indentificateur "a" doit être différent de void (règle 3.17)
Type du field "a" doit être différent de "void" (règle 2.5)
```

- L'utilisatation d'une variable non déclarée:

```
NoDeclarationOfVariable2.deca:2:4: Identificateur "Stringss" non déclaré (règle 0.1)
```

- On ne peut pas redéclarer une variable dans le main ou dans une méthode:

```
Identificateur "x" est déjà déclaré à [2, 8] (règle 3.17)
```

- De même pour la déclaration des champs, méthodes et paramètres:

```
Le field "a" est déjà déclaré à [3, 6] (règle 2.4)
Type du paramètre "a" doit être différent de void (règle 2.9)
```

- La variable (res field, method, classe) ne peut pas être un type prédéfini:

```
L'identificateur "int" est un type prédéfini (essayer de le renommer)
```

- L'identificateur super doit être une classe et qui est déjà déclarée:

```
La classe mère "B" n'est préalablement pas déclarée (règle 1.3)
L'identificateur "int" n'est pas une classe (règle 1.3)
```

- La condition du “if” et du “while” doit être un booléen:

```
"2020" est de type "int". Le type doit être un booléen (règle 3.29)
```

- On doit respecter la compatibilité des types (cas d’initialisation, assign ou return):

```
Initialisation d'une variable de type float par une valeur de type boolean : non autorisée (règle 3.8)

On ne peut pas affecter une expression de type "float" à un identificateur de type "int" (règle 3.28)
```

- On ne peut “printer” que des “String”, “int” et des “float”:

```
L'expression "a" est de type "boolean". Le type doit être un String, un int ou un float (règle 3.31)
```

- On doit respecter la compatibilité des types (cas du Cast):

```
Le Cast  (boolean)(x.a) : (boolean)(int) : n'est pas autorisée (règle 3.39)
```

- L’initialisation d’une variable classe : A a = new B() : le B doit être une classe:

```
L'identificateur "int" doit être une classe (règle 3.42)
```

- L’appel tout court d’une méthode au sein du main soit avec ou sans “this”:

```
"this" ne doit pas apparaître dans le programme principale (règle 3.43)

Le programme principal n'a aucune méthode
```

- On peut redéfinir un champ dans une classe fille en tant que champ seulement:

```
Le field "b" est déja défini dans une super classe en tant que "method" à [3, 2] (règle 2.5)
```

- On peut redéfinir une méthode dans une classe fille seulement en tant que méthode :

```
La méthode "b" est déja définie dans une super classe en tant que "field" à [3, 8] (règle 2.7)
```

- La méthode redéfinie doit être de même signature et doit avoir le même type de retour:

```
La méthode "b" est déja définie dans une super classe à [3, 2]. Le type de retour "int" n'est pas un sous type de "float" (règle 2.7)

La méthode "a" est déja définie dans une super classe à [7, 2] avec une autre signature (règle 2.7)
```

- On gère la visibilitée des champs lors d'un appel (**class.field**):

```
Le type de l'expression "x" doit être un sous type de la classe courante "B" : problème de visibilité (règle 3.66)

la classe courante "X" doit être un sous type de la classe "C" où le champ protégé "x" est déclaré : problème de visibilité (règle 3.66)
```

- On s'assure qu'il s'agit bien d'une méthode lors d'un methodCall **(class.x()**):

```
L'identificateur "x" n'est pas une méthode (règle 3.72)
```

- On s'assure qu'il s'agit bien d'une classe lors d'un methodCall **(a.x()**):

```
L'identificateur "a" n'est pas une classe (règle 3.71)
```

- Et finalement on doit insérer des paramètres compatibles:

```
Veuillez respecter le nombre de paramètres pour la méthode "b" définie à [3, 2] règle(3.74)
```

4.2. Codegen

L'implémentation du compilateur donne 4 figures d'erreur quand on compile le fichier .ass :

- Le débordement de la pile : Stack Overflow
- Débordement arithmétique : Overflow during arithmetic operation
- Erreur d'entrée sortie : I/O error
- Division par 0 : Zero division error

Notons qu'on peut ignorer les tests pour les 2 premières erreurs avec l'option noCheck (voir section 3.4).

5. Extension : Trigo

5.1. Description

Cette extension permet de calculer les fonctions trigonométriques de base des flottants en Deca.

- float sin(float x);
- float cos(float x);
- float atan(float x);
- float asin(float x);

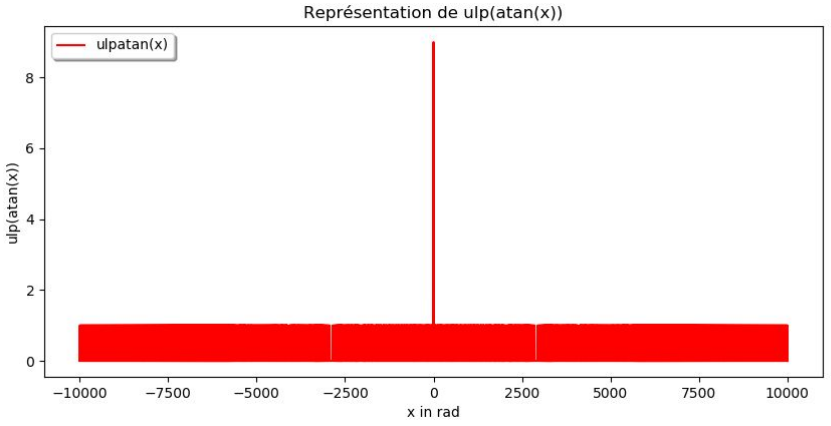
5.2. Utilisation

Il faudra inclure le fichier "Math.decah" dans le script pour accéder aux fonctionnalités en ajoutant #include "Maths.decah".

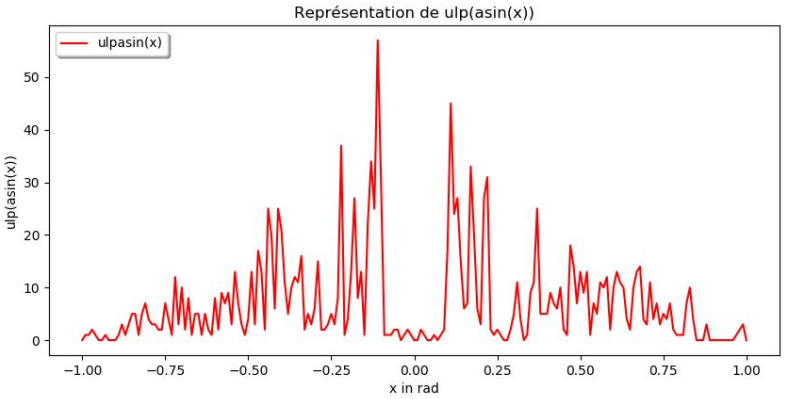
5.3. Résultats en Unit in the last place (ULP)

L'extension, comprend des erreurs qui sont dues aux calculs des flottants en deca et des algorithmes utilisés, qui donnent des petites variations.
On a utilisé une implémentation de la fonction ulp qui nous permet de quantifier cette erreur.

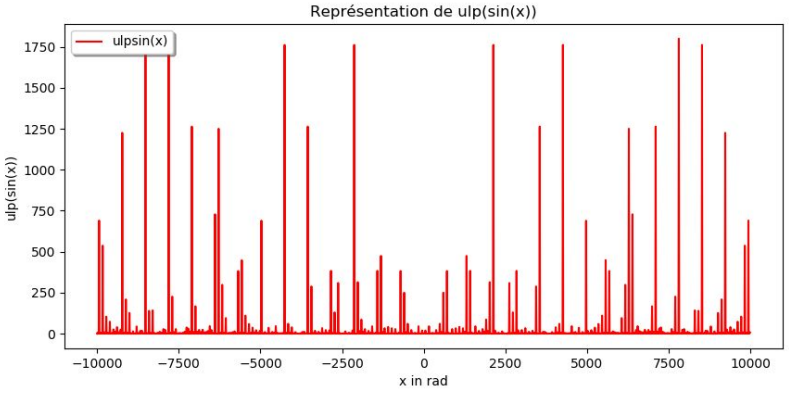
Résultats pour la fonction arctan(x) : x in [-1000;1000];

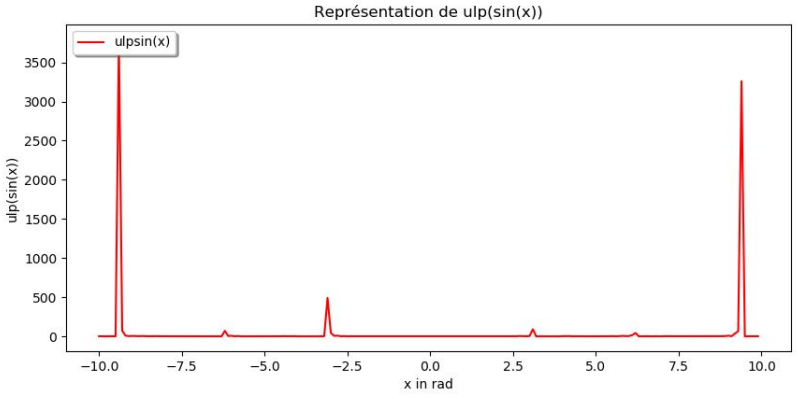


Résultats pour la fonction arcsin(x) : x in [-1;1]

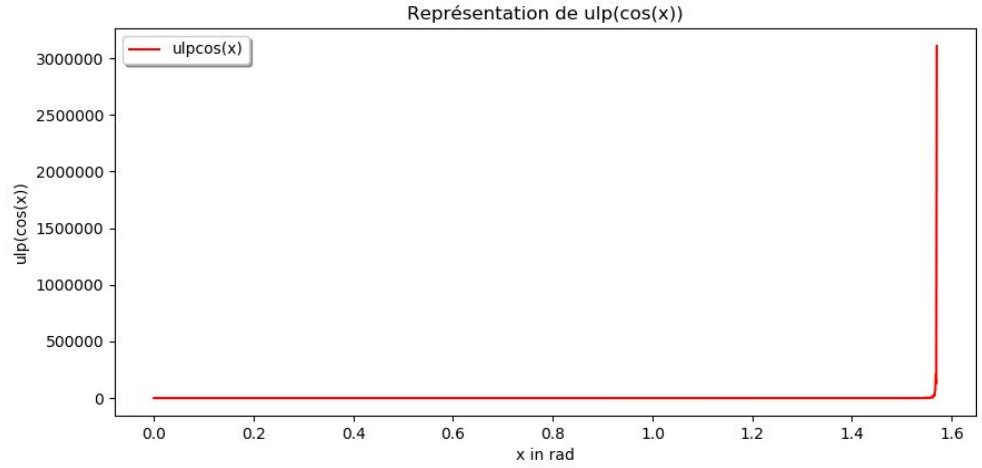


Résultats pour la fonction sin(x) : x in -10000;10000]





Résultats pour la fonction cos(x) : x in [0;π/2]



5.4. Limites de notre implémentation

La fonction arctan donne des ulp égales à 1 ou bien 0 entre -1000 et 1000 avec une erreur de 9 ulp au voisinage de zéro. Elle donne même des valeurs correcte jusqu'à 2 à la puissance 127.

La fonction sinus donne des erreurs en ulp entre 0 et 5 ulp entre -10 000 et 10 000 avec des pics dans un nombre assez petits de points.

La fonction cosinus donne des ulps égales à 0 ou bien à 1 entre 0 et Pi/2, mais pour les grands nombre il y a de grandes erreurs en Ulp, et dans d'autres grand nombre (à l'ordre de 10 000) on trouve tout de même 0 ou bien 1 ulp.

La fonction arcsin donne des erreurs en ulp au maximum égales à 50 ulp sur [-1, 1]. On a calculé cette fonction à partir de l'arctan(qui est bien précise) et de la racine.

6. Annexe

On vous présente un exemple d'une implémentation de la fonction abs(float x) en deca.

```
class A{

  float abs(float x){
      if (x > 0.0F){
         return x;
      }
          else{
            return -x;
          }
   }
}

{
          A a= new A();
          println(a.abs(-5.2), a.abs(2));
}
```

qui donne le fichier **.ass** suivant

```
          TSTO #11
          BOV pile_pleine
          ADDSP #6
; table des methodes
; construction de la table des methodes de Object
          LOAD #null, R0
          STORE R0, 1(GB)
          LOAD code.Object.equals, R0
          STORE R0, 2(GB)
; construction de la table des methodes de A
          LEA 1(GB), R0
          STORE R0, 3(GB)
          LOAD code.A.abs, R0
          STORE R0, 5(GB)
          LOAD code.Object.equals, R0
          STORE R0, 4(GB)
; Main program
; Beginning of main instructions:
; instruction new
          NEW #1, R2
          LEA 3(GB), R0
          STORE R0, 0(R2)
          PUSH R2
          BSR init.A
```

```
        POP R2
        STORE R2, 6(GB)
; appel de methode2
        ADDSP #2
        LOAD 6(GB), R2
        STORE R2, 0(SP)
        LOAD #0x1.4cccccp2, R3
        OPP R3, R3
        STORE R3, -1(SP)
        LOAD 0(SP), R2
        CMP #null, R2
        BEQ pile_pleine
        LOAD 0(R2), R2
        BSR 2(R2)
        SUBSP #2
        LOAD R0, R3
        LOAD R3, R1
        WFLOAT
        WSTR " "
; appel de methode2
        ADDSP #2
        LOAD 6(GB), R3
        STORE R3, 0(SP)
        LOAD #2, R2
        FLOAT R2, R2
        STORE R2, -1(SP)
        LOAD 0(SP), R3
        CMP #null, R3
        BEQ pile_pleine
        LOAD 0(R3), R3
        BSR 2(R3)
        SUBSP #2
        LOAD R0, R2
        LOAD R2, R1
        WFLOAT
        WNL
        HALT
init.A:
        RTS
code.A.abs:
        PUSH R4
        PUSH R3
        PUSH R2
        LOAD -3(LB), R3
        LOAD #0x0.0p0, R4
        SUB R4, R3
        LOAD R3, R1
        BGT OpCmp_if_in_a512
        LOAD #1, R1
        BRA OpCmp_fin_in_a512
OpCmp_if_in_a512:
        LOAD #0, R1
OpCmp_fin_in_a512:
        LOAD R1, R2
        LOAD R2, R1
        BNE IfThenElse_else_a512
        LOAD -3(LB), R4
```

```
        LOAD R4, R0
        BRA IfThenElse_fin_a512
IfThenElse_else_a512:
        LOAD -3(LB), R4
        OPP R4, R4
        LOAD R4, R0
IfThenElse_fin_a512:
        POP R2
        POP R3
        POP R4
        RTS
code.Object.equals:
        PUSH R2
        PUSH R4
        LOAD -2(LB), R2
        LOAD -3(LB), R4
        CMP R2, R4
        SNE R0
        POP R4
        POP R2
        RTS
pile_pleine:
        WSTR "Error: Stack Overflow"
        WNL
        ERROR
over_flow:
        WSTR "Error: Overflow during arithmetic operation"
        WNL
        ERROR
i0_error:
        WSTR "Error: Input/Output error"
        WNL
        ERROR
divisionErr:
        WSTR "Error :Division par 0"
        WNL
        ERROR
```