Manuel utilisateur - Compilateur decac

Équipe 17

Anaïs Hadjazem - Lucas Grellier - Théo Cachet - Clément Vanhemeleryck - César Dumas

-

Janvier 2018

Table des matières

1 Limitations du compilateur

- Parser:
  - Le parser ne lève pas d'erreur en cas d'overflow, par exemple pour une expression arithmétique avec trop de parenthèses. Les erreurs de débordement de pile ne sont pas toujours rattrappées
- Génération du code assembleur
  - Les casts fonctionnent seulment entre des littéraux (et non des classes)
  - Instanceof pas implémenté
  - L'option "-r"(cf 3.) fonctionne dans les cas globaux mais certains bugs ne sont pas fixés
  - La manipulation d'objets peut parfois renvoyé des erreurs nonj rattrappées
  - L'option "-d" (cf 3.) n'affiche pas toutes les traces de debug

Dans l'ensemble, la partie propre à la génération du code assembleur (partie C) n'est totalement stable et non optimale.

2 Erreurs levées lors de la compilation

2.1 Erreurs levées lors de l'étape A

Emplacement : `DecaParseur.g4`

Générée si : certaines expressions ne sont pas conforme à la grammaire concernant les LValue

Message : *Left-hand side of assignment is not an lvalue*

Emplacement : `DecaParseur.g4`

Générée si : le float entré est inférieur au float minimal

Message : *Float too near to zero*

Emplacement : `DecaParseur.g4`

Générée si : le float entré prend une valeur infinie

Message : *Float cannot be infinite*

Emplacement : `DecaParseur.g4`

Générée si : le float entré contient un trop grand nombre de digit

Message : *Float too large*

2.2 Erreurs levées lors de l'étape B

Toutes les erreurs mentionnées ci-dessous sont du type `fr.ensimag.deca.context.ContextualError`. Certains messages d'erreurs dépendent du contexte. On utilisera le symbole '$' pour écrire ces messages.

Identificateur d'expressions : (0.1)

Emplacement : `tree/Identifier.java`

Générée si : la variable identifiée n'est pas dans l'environnement

Message : *Undefined identifier :* $idName

Identificateur de type : (0.2)

Emplacement : `tree/Identifier.java`

Générée si : le type identifié n'est pas dans l'environnement

Message : *Undefined type :* $typeName

Déclaration de classe : (1.3)

Emplacement : `tree/DeclClass.java`

Générée si : la classe hérite d'un type qui n'est pas une classe (int, float...)

Message : *Class cannot inherit from litteral type*

Générée si : la classe est déjà dans l'environnement des types

Message : *Class already defined*

Déclaration de champs : (2.5)

Emplacement : `tree/DeclFields.java`

(2.9) Générée si : un champ est déclaré avec le même nom qu'une méthode de la classe parente

Message : *Field already declared as method in super class*

(2.9) Générée si : un champ est déclaré et son nom est déjà présent dans l'environnement

Message : *Field already declared in current class* ou
Identifier already declared as method in current class

Emplacement : `tree/DeclFieldSet.java`

Générée si : un champ est déclaré avec le type void

Message : *Unsupported type in field set: Field cannot be of type void*

Déclaration de méthode : (2.7/9, 3.12)

(2.7) Emplacement : `tree/AbstractDeclMethod.java`

Générée si : la méthode a le nom d'un champ de la classe parente

Message : *Method already declared as field in super class*

Générée si : la méthode redéfinie n'a pas la même signature que la méthode parente

Message : *Overriden method must have the same signature as parent method*

Générée si : le type de retour de la méthode redéfinie doit être un sous-type de celui de la méthode parente

Message : *Returned type of overriden method must be a subtype of the parent method return type*

Générée si : une méthode est déclaré et son nom est déjà présent dans l'environnement

Message : *Method already declared in current class* ou
Identifier already declared as field in current class

Emplacement : `tree/DeclParam.java`

(2.9) Générée si : un paramètre d'une méthode est de type void

Message : *Returned type of overriden method must be a subtype of the parent method return type*

(3.12) Générée si : deux paramètres de la méthode ont le même nom

Message : *Returned type of overriden method must be a subtype of the parent method return type*

Déclaration de variables : (3.17)

Emplacement : `tree/DeclVar.java`

Générée si : le type de la future variable est void

Message : *Variable cannot be declared with void type*

Générée si : la variable est déjà déclarée

Message : *Variable is already declared*

Expressions - Rvalue : (3.28)

Emplacement : `tree/AbstractExpr.java`

Générée si : l'expression est une "rvalue" et sont type n'est pas compatible avec le type attendu

Message : *Right value expression must be of type* $expectedType

Expressions - Condition : (3.29)

Emplacement : `tree/AbstractExpr.java`

Générée si : l'expression n'est pas de type boolean

Message : *Condition must be of type boolean*

Expressions - Print : (3.31)

Emplacement : `tree/AbstractPrint.java`

Générée si : les arguments ne sont pas de type int, float ou string

Message : *Wrong type of argument: Argument must be of type int, float or string*

Expressions - Cast : (3.39)

Emplacement : `tree/Cast.java`

Générée si : la condition *cast_compatible* n'est pas vérifiée

Message : *Incompatible type to cast*

Expressions - InstanceOf : (3.40)

Emplacement : `tree/InstanceOf.java`

Générée si : l'identifiant de droite n'est pas une instance de classe

Message : *Unsupported type for this operation : Cannot use instanceof on a litteral*

Expressions - New : (3.42)

Emplacement : `tree/New.java`

Générée si : l'identifiant suivant le "new" n'est pas une classe

Message : *Cannot declare a new instance with litteral type*

Expressions - This : (3.43)

Emplacement : `tree/This.java`

Générée si : "this" n'est pas appelé dans la méthode de sa classe

Message : *Cannot use "this" identifier outside of a method*

Opérations binaires arithmétiques : (3.49/50/52/53)

Emplacement : `tree/AbstractOpArith.java`

Générée si : les opérandes ne sont pas de type float ou int

Message : *Unsupported types for this operation: Operands must be of type float or int*

Opérations binaire modulo : (3.51)

Emplacement : `tree/Modulo.java`

Générée si : les opérandes ne sont pas de type int

Message : *Unsupported types for this operation: Operands must be of type int*

Opérations binaires logiques : (3.54/55)

Emplacement : `tree/AbstractOpBool.java`

Générée si : les opérandes ne sont pas de type float

Message : *Unsupported types for this operation: Operands must be of type float*

Opérations binaires de comparaison exacte : (3.56/57)

Emplacement : `tree/AbstractOpCmp.java`

Générée si : les opérandes ne sont pas de type float ou int, ou, dans l'affirmative, elles ne sont pas de même type

Message : *Unsupported types for this operation: Operands must either be of same type or be of type float or int*

Opérations binaires d'inégalité : (3.58/59/60/61)

Emplacement : `tree/AbstractOpCmp.java`

Générée si : les opérandes ne sont pas de type float ou int, ou, dans l'affirmative, elles ne sont pas de même type

Message : *Unsupported types for this operation: Operands must of type float or int*

Opération unaire "unary minus" : (3.62)

Emplacement : `tree/UnaryMinus.java`

Générée si : l'opérande n'est pas de type float ou int

Message : *Unsupported type for this operation: Operand must of type float or int*

Opération unaire "not" : (3.63)

Emplacement : `tree/Not.java`

Générée si : l'opérande n'est pas de type boolean

Message : *Unsupported type for this operation: Operand must of type boolean*

Expressions affectables - Selection : (3.65/66)

Emplacement : `tree/DotSelection.java`

(3.65) Générée si : l'opérande gauche n'est pas une instance de classe

Message : *Unsupported type for this selection: Left operand must be a class instance*

(3.66) Générée si : l'opérande droite de la sélection n'est pas un champ de la classe

Message : *Unsupported operand for this selection: Right operand must be a field*

Générée si : la règle d'appel à un champ protégé est violée

Message : *Forbidden access to protected field*

Appels de méthode : (3.71/74)

(3.71) Emplacement : `tree/DotMethod.java`

Générée si : la méthode appelée n'est pas définie dans la classe ou les super classes de l'objet correspondant

Message : *Method '*$methodName*' is unknown to class $className*

(3.74) Emplacement : `tree/ListExpr.java`

Générée si : l'appel d'une méthode se fait avec la mauvaise signature

Message : *Parameter number* $indexParameter *: expression must be of type int or float* ou
Parameter number $indexParameter *: expression must be a subtype of* $expectedType
ou
Wrong number of parameters in method signature

2.3 Erreurs levées lors de l'étape C

Débordements

Emplacement : `tree/Program.java`

Générée : lors d'un débordement de pile

Message : *Error: Stack overflow*

Générée : lors d'un débordement de la queue

Message : *Error: Heap overflow*

Générée : lors d'un débordement durant une opération arithmétique

Message : *Error: Overflow during arithmetic operation*

Générée : lors d'une division ou d'un modulo par zéro

Message : *Error : Division or modulo by zero*

Générée : lors du déréférencement de null

Message : *Error : Dereferencing null object*

Déclaration de méthodes

Emplacement : `tree/DeclMethod.java`

Générée : si la méthode n'a pas d'instruction "return" et est déclarée avec un type de retour différent de void

Message : *Error: Method* $methodName *expects a non-void returned expression*

3 Mode opératoire

3.1 Utilisation

Pour utiliser le compilateur, il suffit d'utiliser la commande decac avec le nom du fichier .deca en argument. Si la commande est utilisée seule, un message présentant les options disponibles s'affiche. Les différentes options sont :

-b : affiche le numero du groupe

-p : construit l'arbre abstrait le décompile en un programme deca

-v : effectue l'étape de vérification uniquement

-n : ne fait pas les tests de débordement

-r X : exécute le compilateur avec seulement X registres utilisés

-d : active les traces de debug et d'info du compilateur (plusieurs fois pour avoir plus d'informations)

-P : exécute la compilation des fichiers deca en parallèle

L'option -b doit être utilisée seule et les options -v et -p ne sont pas compatibles. Aucun message ne s'affiche lors d'un compilation classique avec la commande decac si tout s'est bien passé.

Le compilateur est aussi fourni avec une base de test. Pour pouvoir lancer ces tests, des scripts sont aussi fournis pour faciliter la vérification des propriétés du compilateur. En lançant la commande `mvn test` après avoir exécuté `mvn test-compile`, tous les scripts de test seront lancés à la suite. Il est aussi possible de lancer les scripts propres à chaque étape du compilateur:

stage_A.sh : lance test_synt sur la base des tests deca propres à l'analyse lexico-syntaxique)

stage_b.sh : lance test_context sur la base des tests deca propres à l'analyse syntaxique contextuelle

stage_c.sh : lance decac sur la base des tests propre à la génération des fichies assembleurs

decompile.sh : lance decac sur tous les tests valides avec l'option -p

option_r.shoption_v.sh : lance decac avec les options correspondantes sur la base des tests propre à la génération des fichies assembleurs

4 Limitations de l'extension

4.1 Limites de précision des algorithmes

Il est difficile de donner une précision fixe pour nos algorithmes, l'erreur étant dépendante de l'ensemble de définition utilisé. Nous pouvons dire que de manière générale, nos algorithmes renvoient une valeur entre 0 et 2 ulp dans plus de 90% des cas sur des arguments de taille raisonnable (plus de 10^5). Voici par exemple, le nombre d'ulp d'écart entre notre valeurs et celles de la classe java.lang.Math sur la fonction arctangente.

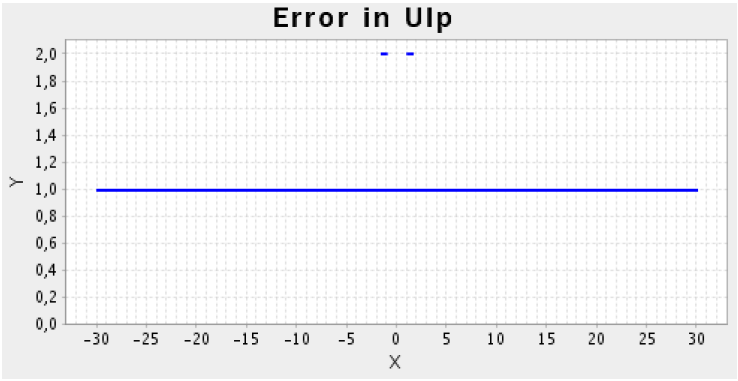

Figure 1: Erreur en ULP de Atan avec 100 000 points

Pour les fonctions sinus et cosinus, nous avons remarqué que, malgré une erreur absolue assez stable d'environ $6E^{-8}$, des pics d'erreurs en ulp se créent autour des abscisses où les fonctions valent 0. Cela s'explique très bien car comme la densité des nombres flottants représentables en machines est très élevée vers 0, une légère erreur en valeur absolue entraine un grand nombre d'ulp d'écart.

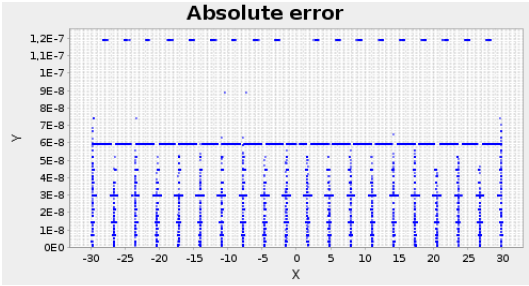

Figure 2: Erreur en valeur absolue de Cosinus avec 100 000 points

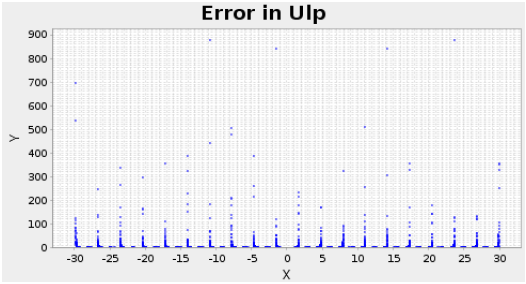

Figure 3: Erreur en ULP de Cosinus avec 100 000 points

Comme mentionné ci-dessus, l'erreur des algorithmes est dépendante de l'ensemble de définition utilisé. En effet pour cosinus et sinus, nous sommes amenés à réduire l'intervalle sur lequel nous approximons les fonctions à un intervalle dont une des bornes est un multiple de π. (Par exemple $[0, \frac{\pi}{4}]$). C'est un problème difficile car, π n'étant pas représentable en machine directement, les modulos entrainent de grosses erreurs. Nous avons réduit celles-ci grâce à un algorithme de *Range*

Reduction et l'instruction FMA. Malgré cela, une erreur de réduction des arguments est présente, et elle grandi linéairement par rapport à la taille de l'argument.
Voici par exemple l'erreur absolue obtenue pour la fonction cosinus entre $[10^6, 10^6+100]$ par rapport à java.lang.Math.

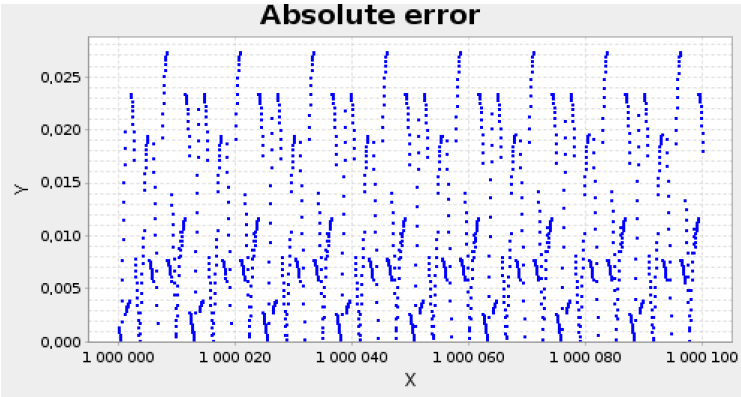

Figure 4: Erreur absolue de cosinus pour de gros arguments

Les limites de précision de la class Math s'expliquent par deux grands facteurs: les erreurs mathématiques des algorithmes et par les erreurs informatiques dues à la représentation discrète des flottants dans une machine. Pour avoir les détails de ces erreurs et en savoir plus sur les algorithmes implémentés, nous vous invitons à lire **la Documentation de la classe Math**.

4.2 Limites de l'implémentation en Déca

Les difficultés de l'implémentation en Déca ont été sous-estimé par notre groupe. Tout d'abord, la class Math utilisant l'instruction FMA des processeurs (Math.fma() en Java), nous devions dans un premier temps gérer la méthode ASM du compilateur Decac, afin de nous permettre d'introduire une méthode en assembleur dans notre classe Math.

Ceci étant fait, nous nous sommes rendu compte que malgré nos 250 scripts de tests, il restait beaucoup, beaucoup d'erreur dans des cas auxquels nous n'avions pas pensé. La plupart de ces erreurs viennent de la génération de code assembleur dans la partie C et ont été révélés par le code de la classe Math.

Ceci étant, nous ne sommes pas parvenu à une implémentation totale de la classe Math (qui est tout à fait opérationnelle en Java) en Déca. Tout le code compile, mais il y a certaines fonctions essentielles qui provoquent des erreurs à l'exécution.
Notons tout de même que le code de Math.deca devrait très bien fonctionner sur un compilateur plus abouti que le notre.