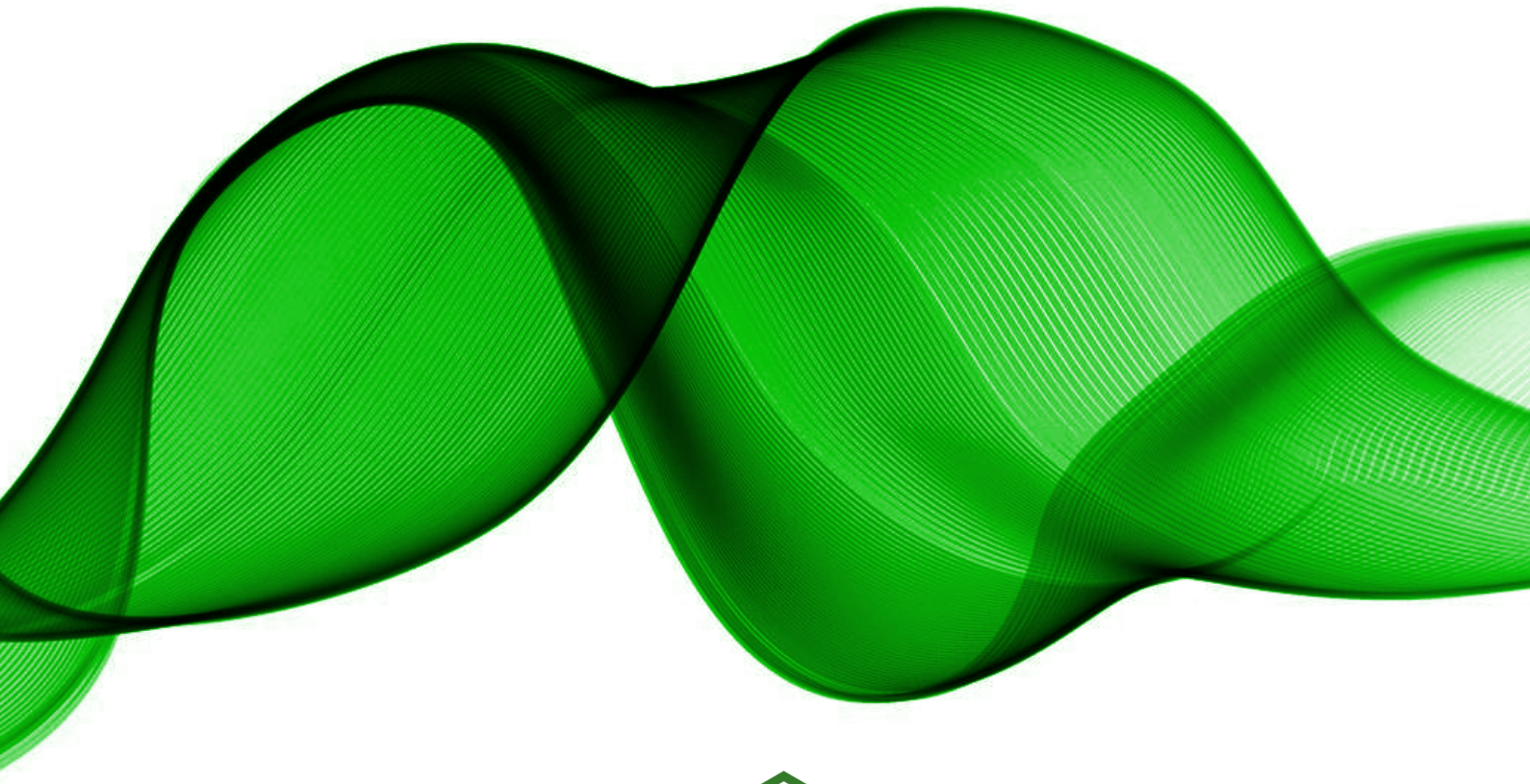

Manuel utilisateur

Projet GL

ELHAJ-LAHSEN Hugo
GONDET Matthias
LAVAL Jean
MAZOUZ Mahmoud
NADJA Mohammad

GRENOBLE INP UGA Ensimag

Table des matières

Limitations du compilateur

Notre compilateur soutient tout le langage Deca sans objet.

Pour le langage Deca avec objet, notre compilateur soutient la vérification syntaxique et contextuelle complète, et la génération de code pour les champs.

Il ne soutient pas les méthodes (pas de génération du code des méthodes mais génère la table des méthodes).

Il soutient les cast et partiellement le instanceof (ne fonctionne qu'avec a instanceof B avec class(a) = B) .

Pour toutes les limitations du compilateur, le compilateur crash avec une erreur interne.

Limitations de l'extension

L'extension ARM ne soutient que le langage Hello World.

Pour toutes les limitations du compilateur, le compilateur crash avec une erreur interne.

Utilisation du compilateur

Notre compilateur respecte la spécification Decac en interface.

Utilisez l'option **-a** activant le backend ARM, qui vous donnera un fichier `program.s`. Utilisez `arm-env.sh` pour exécuter ce programme sous un environnement ARM.

```
. -b (banner) : affiche une bannière indiquant le nom de l'équipe
. -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 vérification
(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)
. -a (ARM) : compilez en ARM (utilisez ./arm-env.sh run pour
lancez, vous devrez ./arm-env.sh install au préalable)
```

Mise en place

Mettre en place la toolchain: `./arm-env.sh install`

Vous aurez besoin d'installer le binaire qemu-arm. Sur Debian, il est donné par qemu-system-arm.

Exemple de compilation

- Compiler son fichier avec l'option -a. `decac -a hello.deca`
- Lancer avec la toolchain: `./arm-env.sh run hello.s`

Messages d'erreur

Cette section du document décrit les messages d'erreur possible du compilateur Deca. Pour chaque erreur, le format de l'erreur exacte est décrit. Les configurations qui peuvent engendrer ces erreurs sont décrites.

Erreurs de syntaxe contextuelle (vérification contextuelle)

Erreur de double class definition (pass 1)

(Rule 1.3)

decl_class ↓env_types ↑{name 7→ (class(super, {}),type_class(name))} ⊕ env_types → DeclClass[Identifier ↑name Identifier ↑super LIST_DECL_FIELD LIST_DECL_METHOD]] condition env_types(super) = (class(__), __)

Description: erreur si on essaye de définir une classe déjà définie .

Message d'erreur: double class definition, this is a verification bug.

Tests: src/test/deca/context/invalid/class/double_class_declaration.deca

Erreur de déclaration d'un Field de type void (pass 2)

(Rule 2.5)

decl_field ↓env_types ↓super ↓class ↑{name 7→ (field(visib, class), type)}

→ DeclField↑visib[

type ↓env_types ↑type

Identifier ↑name

INITIALIZATION]
condition type != void

Description: Erreur si on définit un field de type void.

Message d'erreur: TypeError: un champ ne peut être déclaré avec le type `void`

Tests: src/test/deca/context/invalid/class/field_void.deca

Description: définir le même champ plus qu'une fois.

Message d'erreur: ScopeError: tentative de définir le champ `x` deux fois.

Tests: src/test/deca/context/invalid/class/double_field_decl.deca

Erreur same name Field and Method (pass 2)

(Rule 2.5)

si ((class(__, env_exp_super), __) , env_types(super) et env_exp_super(name) est défini alors env_exp_super(name) = (field(__, __), __).

Description : Erreur si on définit un champ dans une classe et ensuite on essaye de définir une méthode dans cette même classe avec le nom de ce champ.

Message d'erreur: ScopeError: il n'est pas possible d'utiliser le même identificateur pour un champ et une méthode.

Tests: src/test/deca/context/invalid/class/same_name_field_method.deca

Erreur: même méthode a une signature incompatible avec sa methode héritée (pass 2)

(Rule 2.7)

CONDITION: (class(__, env_exp_super), __) , env_types(super)

Description: Erreur si on redéfinit une méthode déjà existante dans notre classe père dans notre classe fils mais avec une signature incompatible avec celle de la définition précédente.

Message d'erreur : ScopeError: la méthode `" + name.getName()

+ "` a une signature incompatible avec sa méthode héritée

Tests: src/test/deca/context/invalid/class/method_error_pere_fils.deca

Erreur: Même classe → (pass 2)

(Rule 2.7)

CONDITION: env_exp_super(name) est défini

Description: si on définit la même méthode identiquement 2 fois dans la même classe on aura une erreur .

Message d'erreur : ScopeError: tentative de définir la méthode `"

+ name.decompile() + "` deux fois.

Tests: src/test/deca/context/invalid/class/same_methode_2_times.deca

Erreur de déclaration d'un paramètre de type void (pass 2)

(Rule 2.9)

decl_param ↓env_types ↑type → DeclParam[type ↓env_types ↑type Identifier ↑__]
condition type 6= void

Description: erreur si on définit un paramètre de classe de type void

Message d'erreur: TypeError: un paramètre ne peut être déclaré avec le type `void`

Tests: src/test/deca/context/invalid/class/method_parametre_void.deca

Erreur: Super class n'est pas une classe (pass 3)

(Rule 3.5)

condition (class(__, env_exp), __) , env_types(class)

Description: si la classe dont on essaye de hériter n'est pas une classe qui existe, on aura un erreur. (exemple quand une classe extends d'un int)

Message d'erreur: TypeError: " + superClassName.decompile() + " n'est pas une classe.

Tests: src/test/deca/context/invalid/class/class_extends_int.deca

(Rule 0.2)

Description: erreur qui est similaire a cette erreur là , c'est quand on extend d'une classe non définie (exemple A extends B{} avec B non définie avant)

Message d'erreur: ScopeError: le type `..` n'est pas défini.

Tests: src/test/deca/context/invalid/class/extends_non_existing_class.deca

Erreur: utilisation du même paramètre plus qu'une fois (pass 3)

(Rule 3.13)

decl_param ↓env_types ↑{name 7→ (param, type)}

→ DeclParam [type ↓env_types ↑type Identifier ↑name]

Description: on ne peut pas utiliser deux paramètres du même nom dans une même méthode (ex: plus(int x , int x)) puisqu'on pourrait plus distinguer entre ces deux x .

Message d'erreur: ScopeError: le paramètre `" + varName.getName() + "` est défini plus qu'une fois.

Tests: src/test/deca/context/invalid/class/same_param_2_times.deca

Erreur: méthode de return type void ou il ya return (pass 3)

(Rule 3.24)

→ Return [rvalue ↓env_types ↓env_exp ↓class ↓return]

condition return != void

Description: on ne peut pas déclarer une méthode avec un type de retour void avec laquelle on met un return dedans.

Message d'erreur: TypeError: il est impossible de retourner une valeur quand la méthode a un type de retour `void`.

Tests: src/test/deca/context/invalid/class/return_in_void_method.deca

Erreur: return type of the method different from the expected return type (pass 3)

(Rule 3.28)

rvalue ↓env_types ↓env_exp ↓class ↓type1

→ expr ↓env_types ↓env_exp ↓class ↑type2

condition assign_compatible(env_types, type1, type2)

Description: on aura une erreur au cas la valeur retournée par la méthode n'est pas compatible avec le type de retour dont la méthode est supposée à retourner.

Message d'erreur: TypeError: type incorrect pour expression `"

+ this.decompile() + "`, attendu `" + expectedType

+ "` mais trouvé `" + exprType + "`.

Tests: src/test/deca/context/invalid/class/mauvaise_fonction.deca

Erreur: return type different then boolean (pass 3)

(Rule 3.29)

condition ↓env_types ↓env_exp ↓class

→ expr ↓env_types ↓env_exp ↓class ↑boolean

Description: on aura une erreur dans le cas ou le type de retour précisé est boolean tandis que le type retourné est différent de boolean .

Message d'erreur: TypeError: type incorrect pour expression `"

+ this.decompile() + "`, attendu `boolean`"

+ " mais trouvé `" + exprType + "`.

Tests: src/test/deca/context/invalid/class/mauvaise_fonction_boolean.deca

Erreur: print d'un type invalide (pass 3)

(Rule 3.31)

exp_print ↓env_types ↓env_exp ↓class

→ expr ↓env_types ↓env_exp ↓class ↑type

condition type = int ou type = float ou type = string

Description: on aura une erreur dans le cas ou on essaye de faire un print d'un type autre que les types : "int" , "float" , "string" .

Message d'erreur: TypeError: type(s) incorrect(s) dans "

+ "l'instruction `" + this.decompile()

+ "`, attendu `int` ou bien `float`, mais trouvé `"

+ exprType + "`.

Tests: src/test/deca/context/invalid/class/print_incompatible.deca

Erreur: Cast incompatible (pass 3)

(Rule 3.39)

Cast [

type ↓env_types ↑type

expr ↓env_types ↓env_exp ↓class ↑type2]

condition cast_compatible(env_types, type2, type)

Description: Si l'on essaye de cast vers un type qui n'est pas int, float ou d'une class définie.

Message d'erreur: `"type error" + TypeCastedTo.decompile() + "n'est pas int ou float ou une class"`

Tests: src/test/deca/context/invalid/class/Cast_to_class_from_int.deca

Description: Si l'on essaye de cast une expression qui n'est pas de type int, float ou d'une class définie.

Message d'erreur: `"Type: l'expression "+ rightOperand.decompile() + "doit être un int, float ou une class"`

Tests: src/test/deca/context/invalid/class/Cast_to_int_from_class.deca

Description: Si l'on essaye de cast vers un type qui ne correspond pas au superType attendu.

Message d'erreur: `message = "Type: l'expression `" + TypeCastedTo.decompile() + "` doit être un sous type de `" + classType + ".";`

Tests: src/test/deca/context/invalid/class/Cast_to_class_not_related.deca

Description: Si l'on essaye de cast vers un type qui n'est pas une super/sousclass du type de l'expression.

Message d'erreur: `"Type: les deux expressions sont de types incompatible.";`

Tests: Aucun

Erreur InstanceOf incompatible (pass 3)

(Rule 3.42)

→ InstanceOf[

expr ↓env_types ↓env_exp ↓class ↑type1

type ↓env_types ↑type2]

affectation type := type_instanceof_op(type1, type2)

→ method_call ↓env_types ↓env_exp ↓class ↑typ

→ New [type ↓env_types ↑type]

condition type = type_class(__)

Description: erreur dans le cas application d'instanceOf à une expression qui n'est pas une class ou null.

Message d'erreur: `"TypeError: " + expression.decompile() + "isn't an object or null";`

Tests: Aucun

Description: erreur dans le cas de vérification de instanceOf de quelque chose qui n'est pas une class (ex: a instanceOf int, a instanceOf classNonDefinie...) .

Message d'erreur: `"TypeError: " + typeInstanced.decompile() + "isn't a class";`

Tests: Aucun

Erreur d'utilisation de "this" hors class contexte (pass 3)

(Rule 3.43)

→ This condition class != 0 affectation type := type_class(class)

Description: erreur si on utilise "this" en dehors du contexte d'une classe .

Message d'erreur: TypeError: `this` ne peut être utilisée que dans le contexte d'une classe.

Tests : src/test/deca/context/invalid/class/mal_use_of_this.deca

Erreur : wrong use of fields outside of class (pass 3)

(Rule 3.65)

condition (class(__, env_exp2), __) , env_types(class2)

Description: le premier type d'erreur avec les fields est si le field n'existe pas déjà dans la classe et on essaye de l'utiliser dans main ce qui affiche une erreur.

Message d'erreur: TypeError: `" + attribute.decompile() + "` n'est pas un champ.

Tests : src/test/deca/context/invalid/class/non_existing_field.deca

(Rule 3.66)

→ Selection [

expr ↓env_types ↓env_exp ↓class ↑type_class(class2)

field_ident ↓env_exp2 ↑protected ↑class_field ↑type]

condition (class(__, env_exp2), __) , env_types(class2)

et subtype(env_types,type_class(class2),type_class(class))

et subtype(env_types,type_class(class),type_class(class_f ield))

Description: le premier type d'erreur qu'on peut avoir avec les champs et lorsqu'on utilise dans notre main un champ protégé ce qui nous affiche une erreur.

Message d'erreur: ScopeError: le champ `" + attribute.decompile() + "` est protégé.

Tests: src/test/deca/context/invalid/class/bad_use_protected.deca

---

Description: on ne peut pas déclarer une classe contenant une autre classe avec un champ protégé, pour tenter de contourner le protected d'un attribut (example page 91).

Message d'erreur: Type: l'expression `" + expression.decompile()

+ "` doit être un sous type de `" + classType + "`.

Tests: src/test/deca/context/invalid/class/bad_use_fields_inheritance.deca

Erreur : wrong use of lvalue (pass 3)

(Rule 3.67, 3.68, 3.69)

lvalue_ident ↓env_exp ↑type

→ identifier ↓env_exp ↑(field(__, __), type)

→ identifier ↓env_exp ↑(param, type)

→ identifier ↓env_exp ↑(var, type)

Description: erreur si on utilise des lvalue autres que les identificateurs champs, paramètres et variables.

Message d'erreur: ScopeError: seuls les identificateurs champs, paramètres et variables peuvent être des lvalue.`

Tests : src/test/deca/context/invalid/class/non_existing_field.deca

Erreur avec les appels des méthodes (pass 3)

(Rule 3.71 - 3.74)

method_call ↓env_types ↓env_exp ↓class ↑type

→ MethodCall [

expr ↓env_types ↓env_exp ↓class ↑type_class(class2)

method_ident ↓env_exp2 ↑sig ↑type

rvalue_star ↓env_types ↓env_exp ↓class ↓sig]]

condition (class(__, env_exp2), __) , env_types(class2)

Description: si on appelle la méthode avec une expression qui n'est pas même un objet .

Message d'erreur: TypeError: " + expression.decompile() + "` n'est pas un objet.

Tests: src/test/deca/context/invalid/class/no_object_call_method.deca

---

Description: si on appelle une méthode non existante dans la classe avec un objet de cette classe.

Message d'erreur: TypeError: `" + method.decompile() + "` n'est pas une méthode.

Tests: src/test/deca/context/invalid/class/no_method_call_method.deca

---

Description: si on appelle une méthode avec des paramètres qui ne sont pas compatibles avec sa signature initiale dans la classe . (nombre de parametre different et non pas le type)

Message d'erreur: TypeError: l'appel méthode `" + decompile() + "` est incompatible avec sa signature.

Tests: src/test/deca/context/invalid/class/wrong_signature_appel_methode.deca

Erreurs d'exécution (assembleur)

→ **Programmes incorrects:**

1) **Description**: division entière (et reste de la division entière) par 0 .

Message d'erreur: Erreur : division par zéro.

2) **Description**: débordement arithmétique sur les flottants (inclut la division flottante par 0.0) ;

Message d'erreur: Erreur : débordement pendant opération arithmétique entre deux flottants.

3) **Description:** absence de return lors de l'exécution d'une méthode ;

Message d'erreur: Erreur: méthode sans `return`.

4) **Description:** déréférencement de null.

Message d'erreur: Erreur: déréférencement d'un pointer `null`

5) **Description:** conversion de type impossible ;

Message d'erreur : Erreur: conversion de types impossible.

→ **Programmes corrects, dont l'exécution dépend de l'utilisateur:**

1) **Description:** erreur de lecture (valeur entrée pas dans le type attendu)

Message d'erreur: Erreur : valeur entrée pas dans le type attendu.

→**Programmes corrects, dont l'exécution dépasse les limites de la machine:**

Description: débordement mémoire (pile ou tas).

Message d'erreur: Erreur : débordement de la pile.