CAFEBABE

Refactoring & Optimisations

Laffineur Gérôme

12 Décembre 2016

Université de Namur

La détection de patterns

Fonctionnement de la JVM

La JVM est stack based:

Toutes les opérations se font en fonctions des valeurs présentes sur la pile. Les valeurs sont systématiquement lues et écrites sur la pile.

$\rightarrow$ **Consomme beaucoup de ressources** ☹

Fonctionnement de la JVM

La JVM est stack based:

Toutes les opérations se font en fonctions des valeurs présentes sur la pile. Les valeurs sont systématiquement lues et écrites sur la pile.

→ **Consomme beaucoup de ressources** ☹

Illustration :

iload_1
iload_2
iadd
istore_0

local_var_0	local_var_1	local_var_2
0	42	10

Fonctionnement de la JVM

La JVM est stack based:

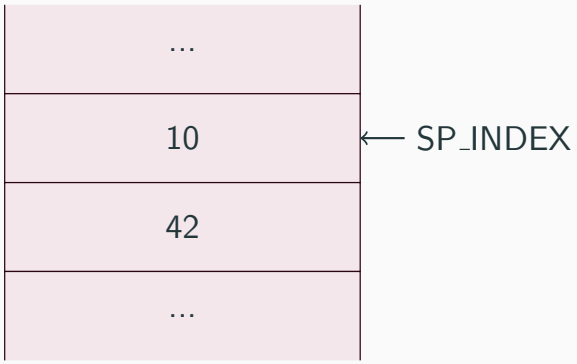
Toutes les opérations se font en fonctions des valeurs présentes sur la pile. Les valeurs sont systématiquement lues et écrites sur la pile.

$\rightarrow$ **Consomme beaucoup de ressources** ☹

Illustration :

iload_1
iload_2
iadd
istore_0

local_var_0	local_var_1	local_var_2
0	42	10



Fonctionnement de la JVM

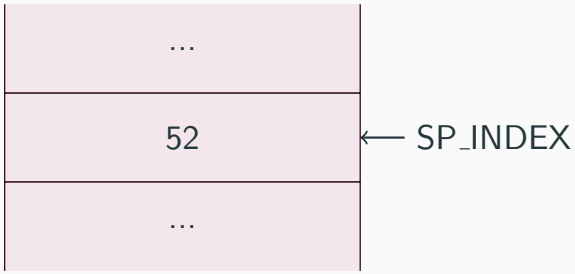
La JVM est stack based:

Toutes les opérations se font en fonctions des valeurs présentes sur la pile. Les valeurs sont systématiquement lues et écrites sur la pile.

$\rightarrow$ **Consomme beaucoup de ressources** ☹

Illustration :

iload_1
iload_2
iadd
istore_0



local_var_0	local_var_1	local_var_2
0	42	10

Fonctionnement de la JVM

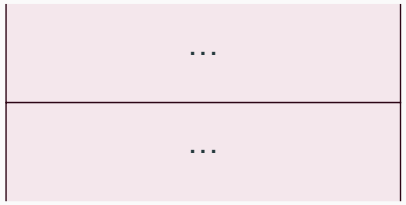
La JVM est stack based:

Toutes les opérations se font en fonctions des valeurs présentes sur la pile. Les valeurs sont systématiquement lues et écrites sur la pile.

$\rightarrow$ **Consomme beaucoup de ressources** ☹

Illustration :

iload_1
iload_2
iadd
istore_0



local_var_0	local_var_1	local_var_2
52	42	10

La représentation CLP

Stack frame *Stack d'opérandes + Tableau des variables locales pour une méthode donnée*

La représentation CLP :

Stack frame déroulé et représenté comme une suite de variables.

Plus de *"frontière"* entre le tableau des variables locales et la pile d'opérandes.

$\rightarrow$ Dans certains cas il est donc possible de travailler directement avec les variables représentant les variables locales, sans devoir passer les valeurs par la pile.

La représentation CLP

Stack frame *Stack d'opérandes + Tableau des variables locales pour une méthode donnée*

La représentation CLP :
Stack frame déroulé et représenté comme une suite de variables.
Plus de *"frontière"* entre le tableau des variables locales et la pile d'opérandes.

$\rightarrow$ Dans certains cas il est donc possible de travailler directement avec les variables représentant les variables locales, sans devoir passer les valeurs par la pile.

Idée : Détection de patterns.

Illustration : Soustraction de 2 variables

iconst_5

istore_2

iconst_1

istore_3

iload_2

iload_3

isub

istore_1

local_var_1	local_var_2	local_var_3
0	0	0

Illustration : Soustraction de 2 variables

iconst_5

istore_2

iconst_1

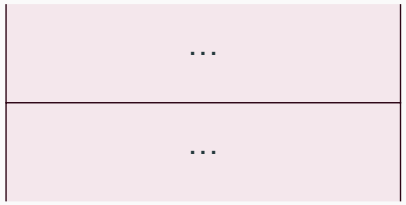
istore_3

iload_2

iload_3

isub

istore_1



local_var_1	local_var_2	local_var_3
0	5	0

Illustration : Soustraction de 2 variables

iconst_5

istore_2

iconst_1

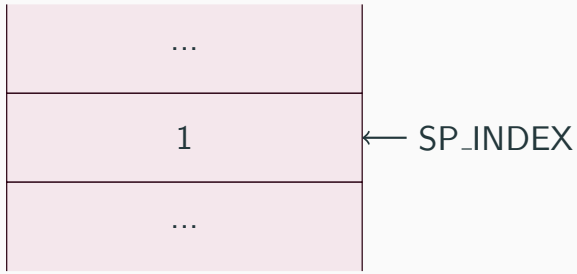
istore_3

iload_2

iload_3

isub

istore_1



local_var_1	local_var_2	local_var_3
0	5	0

Pattern d'initialisation de variable :
(const→store)

Illustration : Soustraction de 2 variables

iconst_5

istore_2

iconst_1

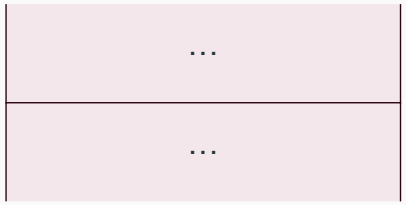
istore_3

iload_2

iload_3

isub

istore_1



local_var_1	local_var_2	local_var_3
0	5	1

Pattern d'initialisation de variable :
(const→store)

Illustration : Soustraction de 2 variables

iconst_5

istore_2

iconst_1

istore_3

iload_2

iload_3

isub

istore_1

local_var_1	local_var_2	local_var_3
0	5	1

Pattern d'initialisation de variable :
(const→store)

Illustration : Soustraction de 2 variables

iconst_5

istore_2

iconst_1

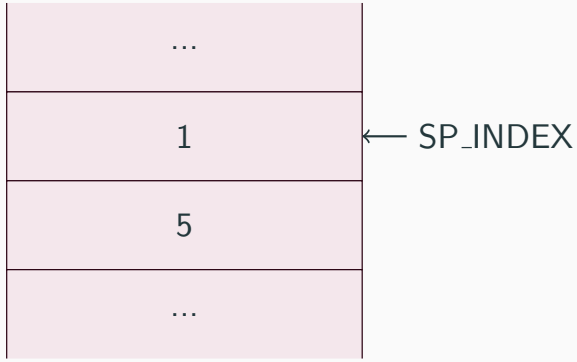
istore_3

iload_2

iload_3

isub

istore_1



local_var_1	local_var_2	local_var_3
0	5	1

Pattern d'initialisation de variable :
(const→store)

Illustration : Soustraction de 2 variables

iconst_5

istore_2

iconst_1

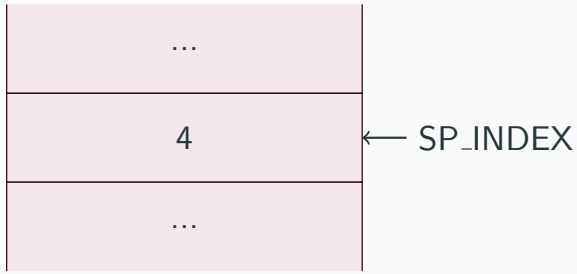
istore_3

iload_2

iload_3

isub

istore_1



local_var_1	local_var_2	local_var_3
0	5	1

Pattern d'initialisation de variable :
(const→store)

Illustration : Soustraction de 2 variables

iconst_5

istore_2

iconst_1

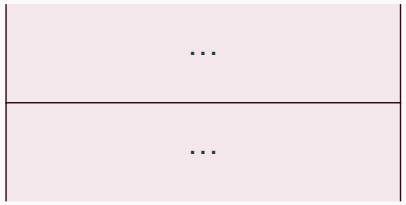
istore_3

iload_2

iload_3

isub

istore_1



local_var_1	local_var_2	local_var_3
4	5	1

Pattern d'initialisation de variable :
(const→store)

Illustration : Soustraction de 2 variables

iconst_5

istore_2

iconst_1

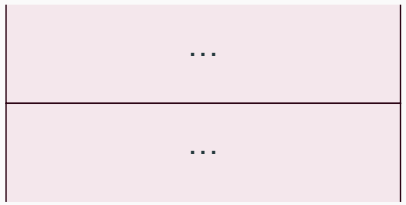
istore_3

iload_2

iload_3

isub

istore_1



local_var_1	local_var_2	local_var_3
4	5	1

Pattern d'initialisation de variable :

(const→store)

Pattern d'opération arithmétique :

(load→load→op→store)

On peut représenter chaque pattern par une seule clause CLP !

Représentation du Heap (V2)

Nouvelle représentation du Heap (writeMemory)

Write Memory (IN,IN,IN)

```
writeMemory(OBJECTREF,FIELD,VALUE) :- { },
    retract(object(OBJECTREF,FIELD,X)),
    assert(object(OBJECTREF,FIELD,VALUE)).
writeMemory(OBJECTREF,FIELD,VALUE) :- { },
    assert(object(OBJECTREF,FIELD,VALUE)).
```

Nouvelle représentation du Heap (writeMemory)

Write Memory (IN,IN,IN)

```
writeMemory(OBJECTREF,FIELD,VALUE) :- { },
    retract(object(OBJECTREF,FIELD,X)),
    assert(object(OBJECTREF,FIELD,VALUE)).
writeMemory(OBJECTREF,FIELD,VALUE) :- { },
    assert(object(OBJECTREF,FIELD,VALUE)).
```

Write Memory (IN,OUT,IN,IN,IN)

```
writeMemory(HI,HO,O,F,V) :- {},
    select(object(O,F,_),HI,HO1),
    writeMemory(HO1,HO,O,F,V).

writeMemory(HI,[object(O,F,V)|HI],O,F,V).
```

Nouvelle représentation du Heap (readMemory)

Read Memory (IN,IN,OUT)

readMemory(OBJECTREF,FIELD,VALUE) :-

{ },

object(OBJECTREF,FIELD,VALUE).

Nouvelle représentation du Heap (readMemory)

Read Memory (IN,IN,OUT)

```
readMemory(OBJECTREF,FIELD,VALUE) :-
    { },
    object(OBJECTREF,FIELD,VALUE).
```

Read Memory (IN,IN,IN,OUT)

```
readMemory([object(O,F,V)|HS],O,F,V) :- {}.
readMemory([object(OO,FF,_)|HS],O,F,V) :-
    {OO\=O || FF\=F},
    readMemory(HS,O,F,V).
```

Gestion du code non inclus

Gestion du code non inclus

Comment gérer les appels aux librairies externes ?

Traduire également ces librairies externes **X**

→ Problème : Le compilateur ne gère que les types numériques, or ces librairies utilisent potentiellement d'autres types.

Remplacer les appels externes par des prédicats neutres **V**

→ Idée : Garder une trace des différents appels en insérant le nom de la méthode/librairie dans le nom de prédicat.

Pas d'impact sur l'exécution CLP si appels facultatif.

Refactoring & Documentation

Préparation pour la maintenance

Refactoring Prise de recul par rapport au code produit.
Vue d'ensemble $\rightarrow$ Généralisations.

Documentation Mise au propre de la JavaDoc.
Commentaires des parties complexes.

MERCI DE VOTRE ATTENTION