Étude d'une machine virtuelle en héritage multiple basée sur le hachage parfait

Julien Pagès

LIRMM, MaREL

23 Octobre 2014

Le projet

Thèse commencée en 2013 : Étude d'une machine virtuelle en héritage multiple basée sur le hachage parfait

Objectifs principaux

- Preuve de faisabilité d'une telle machine virtuelle
- Expérimenter des techniques d'implémentation dans les machines virtuelles
- Étudier les protocoles d'optimisation dynamiques
- Avoir une machine virtuelle (relativement) efficace

Les machines virtuelles

Trois grandes familles de systèmes d'exécution

1. Interpréteur
2. Compilateur
3. Machine virtuelle

Caractéristiques d'une machine virtuelle

- Chargement dynamique (monde ouvert)
- Compilation à la volée

Outils

Développer une telle machine virtuelle est un projet très important

Outils

Développer une telle machine virtuelle est un projet très important

La réutilisation d'outils est obligatoire !

Outils choisis

1. Un langage : nit[2]
2. Une technique d'implémentation de l'héritage multiple : le hachage parfait[1]
3. Une aide au développement : machine virtuelle développée à partir de l'interpréteur nit

Outils

Développer une telle machine virtuelle est un projet très important

La réutilisation d'outils est obligatoire !

Outils choisis

1. Un langage : nit[2]
2. Une technique d'implémentation de l'héritage multiple : le hachage parfait[1]
3. Une aide au développement : machine virtuelle développée à partir de l'interpréteur nit

nitvm : une machine virtuelle pour nit, en nit

Fonctionnement de l'interpréteur nit

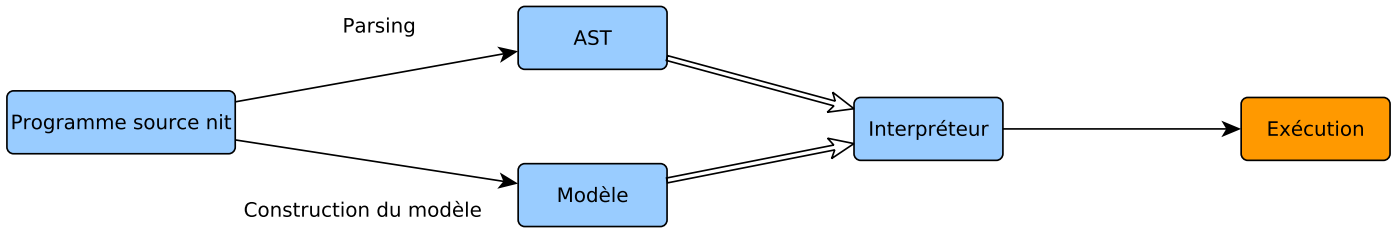


Figure : Fonctionnement de l'interpréteur nit

Fonctionnement de l'interpréteur nit

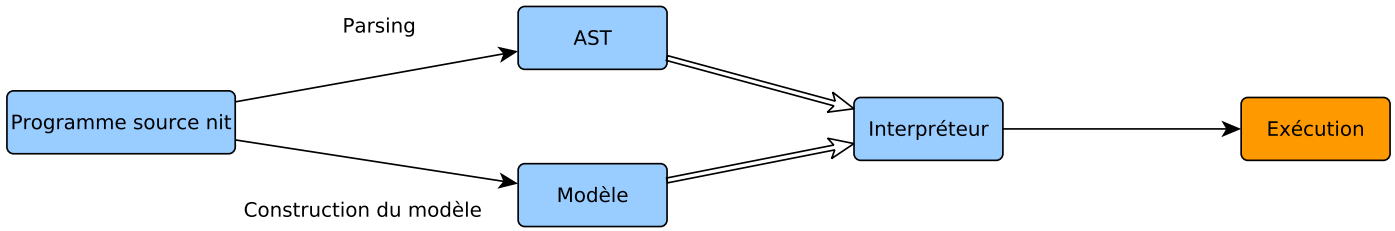


Figure : Fonctionnement de l'interpréteur nit

Éléments manquants par rapport à une machine virtuelle

- Chargement dynamique
- Implémentation efficace des mécanismes objets
- Protocoles d'optimisations
- Compilation à la volée du code nit

Comment développer une telle machine virtuelle

Théorème : *ce qui fonctionne avec l'interpréteur fonctionne avec la nitvm*

Méthodologie

- Développement par héritage de l'interpréteur
- Ce qui n'est pas remplacé fonctionne encore
- Utilisation du C pour les structures de bas-niveau depuis nit

Se rapprocher du modèle d'une machine virtuelle

1. Chargement dynamique : chargement de classe à la première instanciation
2. Compilation à la volée : construction paresseuse des structures

Comment avoir une machine virtuelle efficace ?

Implémentation efficace des mécanismes objet

- Appel de méthode
- Accès aux attributs
- Test de sous-typage

Avec des variantes :

1. Héritage simple $\rightarrow$ rapide
2. Héritage multiple $\rightarrow$ lent

Comment avoir une machine virtuelle efficace ?

Implémentation efficace des mécanismes objet

- Appel de méthode
- Accès aux attributs
- Test de sous-typage

Avec des variantes :

1. Héritage simple → rapide
2. Héritage multiple → lent

Mais aussi

- Un modèle paresseux ne compile que le nécessaire : gain "gratuit"

Comment avoir une machine virtuelle efficace ?

Implémentation efficace des mécanismes objet

- Appel de méthode
- Accès aux attributs
- Test de sous-typage

Avec des variantes :

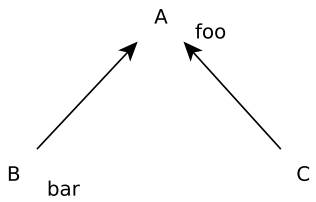
1. Héritage simple $\rightarrow$ rapide
2. Héritage multiple $\rightarrow$ lent

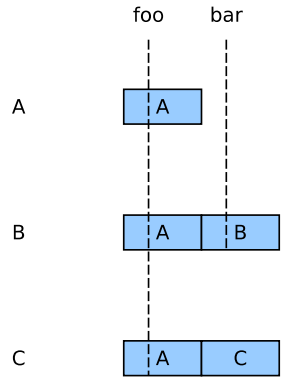
Mais aussi

- Un modèle paresseux ne compile que le nécessaire : gain "gratuit"
- Des optimisations pendant l'exécution

Protocole de compilation/recompilation : problématique

Implémentation des méthodes en héritage simple

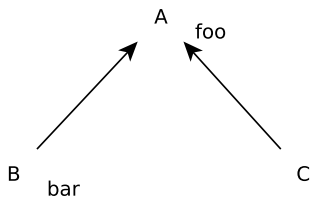


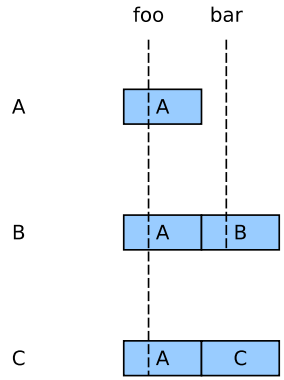



```
A x = new A()
x.foo
```

Protocole de compilation/recompilation : problématique

Implémentation des méthodes en héritage simple



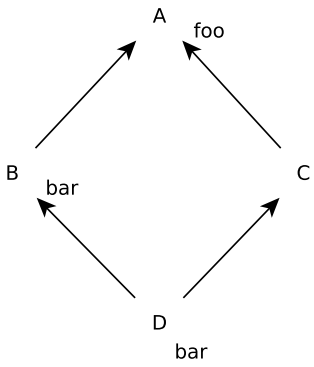


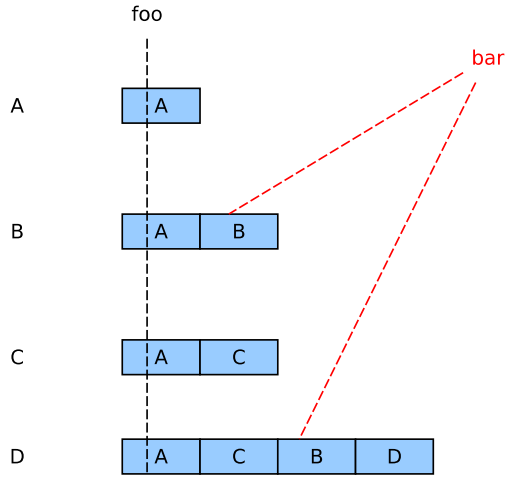

```
A x = new A()
x.foo
```

```
load [x + #tableOffset], table
load [table + #fooOffset], methAddr
call #methAddr
```

Protocole de compilation/recompilation : problématique

Implémentation des méthodes en héritage multiple

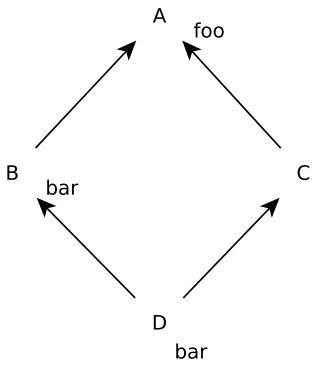


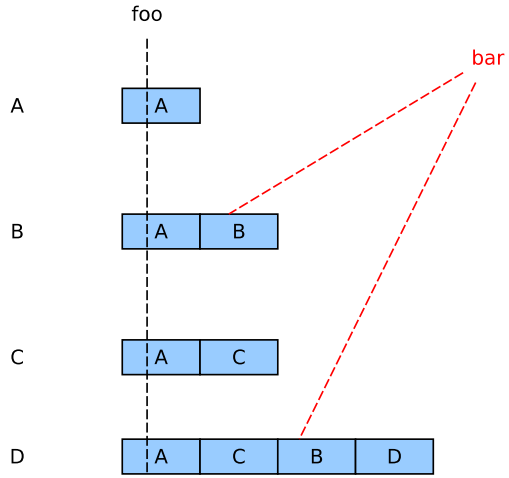



```
B x = new D()
x.bar
```

Protocole de compilation/recompilation : problématique

Implémentation des méthodes en héritage multiple

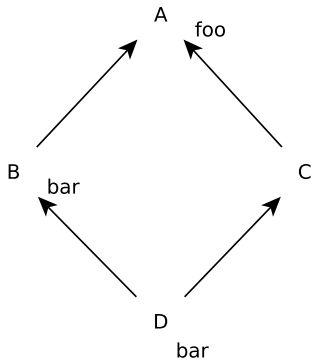


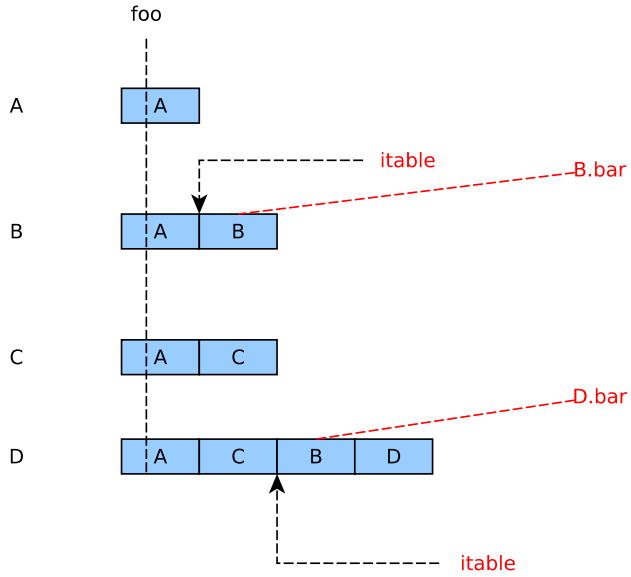



```
B x = new D()
x.bar
```

```
load [x + #tableOffset], table
load [table + #barOffset], methAddr
call #methAddr
```

Protocole de compilation/recompilation : problématique



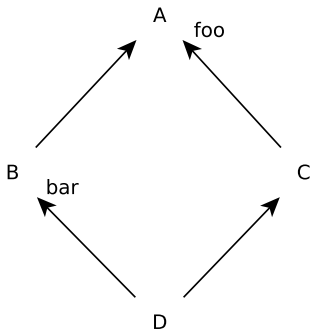


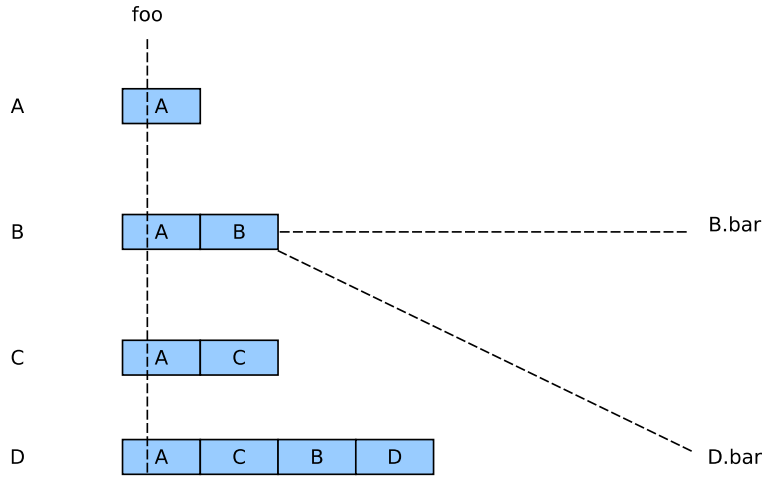

```
B x = new D()
x.bar
```

séquence de hachage parfait
itable = offset du bloc concerné

```
load [x + #tableOffset], itable
load [itable + #barOffset], methAddr
call #methAddr
```

Protocole de compilation/recompilation : exemple

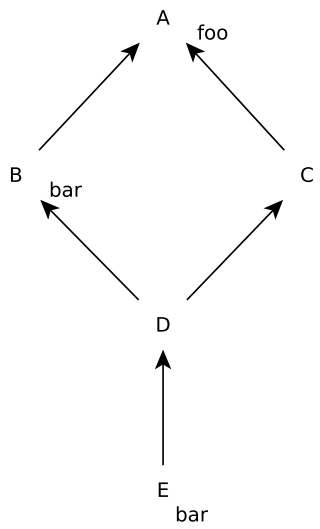


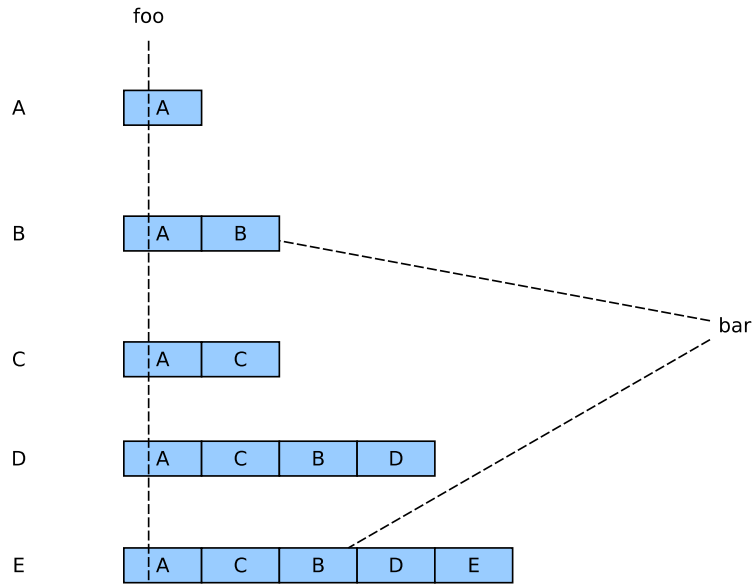


Si la méthode `bar` de `B` *n'est jamais redéfinie* :

- On compile tous les appels à `bar` par un appel direct à `bar` de `B`

Protocole de compilation/recompilation : exemple





Si la méthode `bar` de `B` *est redéfinie* dans `E` :

- On recompile certains appels à `bar` par un appel en héritage multiple

Protocole de compilation/recompilation : objectifs

Objectifs

- Maximiser les appels directs et ceux en implémentation SST...
- Quitte à devoir recompiler certains sites de manière moins efficace au chargement d'une classe

Protocole de compilation/recompilation : objectifs

Objectifs

- Maximiser les appels directs et ceux en implémentation SST...
- Quitte à devoir recompiler certains sites de manière moins efficace au chargement d'une classe

Expérimentations

- Test de différentes techniques d'optimisations et de recompilation
- Recompilation en changeant les nœuds d'AST, donc les implémentations
- Statistiques par comptage des nœuds de l'arbre syntaxique

Protocole de compilation/recompilation : objectifs

Objectifs

- Maximiser les appels directs et ceux en implémentation SST...
- Quitte à devoir recompiler certains sites de manière moins efficace au chargement d'une classe

Expérimentations

- Test de différentes techniques d'optimisations et de recompilation
- Recompilation en changeant les nœuds d'AST, donc les implémentations
- Statistiques par comptage des nœuds de l'arbre syntaxique

But : Trouver le protocole avec le meilleur compromis coût/efficacité

Bilan

État actuel du développement

- Simulation du chargement dynamique
- Meilleures structures de représentations des objets
- Ordonnancement des superclasses
- Implémentation de quelques mécanismes objets avec PH

À venir

- Terminer les différentes implémentations
- Travailler sur les protocoles de compilation/recompilation
- Compilation à la volée du code nit
- Langage intermédiaire pour nit

Il est déjà possible d'exécuter des programmes nit avec la nitvm

Références

Ducournau, R., and Morandat, F.
Perfect class hashing and numbering for object-oriented implementation.
Software: Practice and Experience 41, 6 (2011), 661–694.

Privat, J.
Nit language.
http://nitlanguage.org/.

Pour aller plus loin

`http://nitlanguage.org/`

`https://github.com/privat/nit`
`https://github.com/jpages/nit`

Pour aller plus loin

`http://nitlanguage.org/`

`https://github.com/privat/nit`
`https://github.com/jpages/nit`

Merci de votre attention