Génération Automatique de Benchmark

Comment tout casser sans se fatiguer

Adrien Mathieu

Lundi 11 septembre 2023

Introduction

Definition

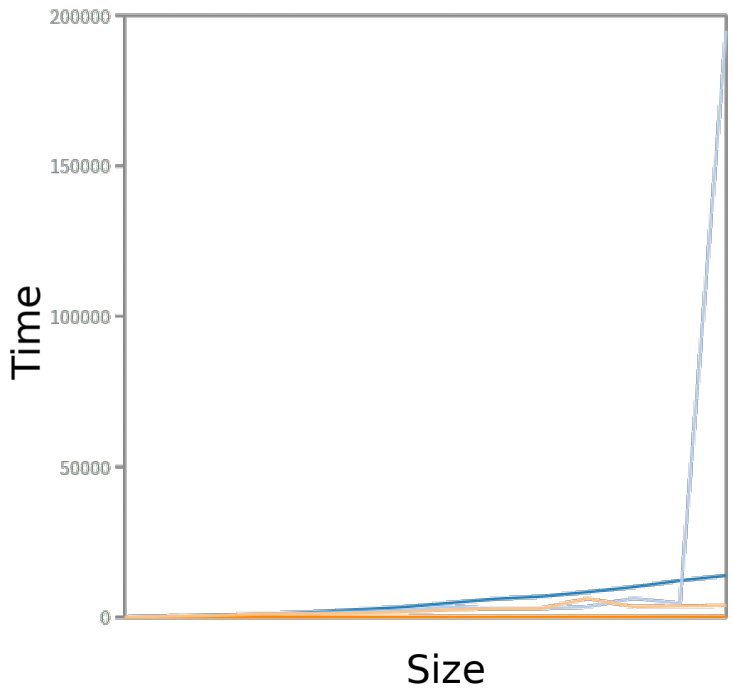
Un benchmark d'un programme est l'exécution de ce programme sur une entrée donnée dans le but d'évaluer sa performance.

Introduction

Definition

Un benchmark d'un programme est l'exécution de ce programme sur une entrée donnée dans le but d'évaluer sa performance.

Définir des benchmarks pertinents est difficile.



Introduction

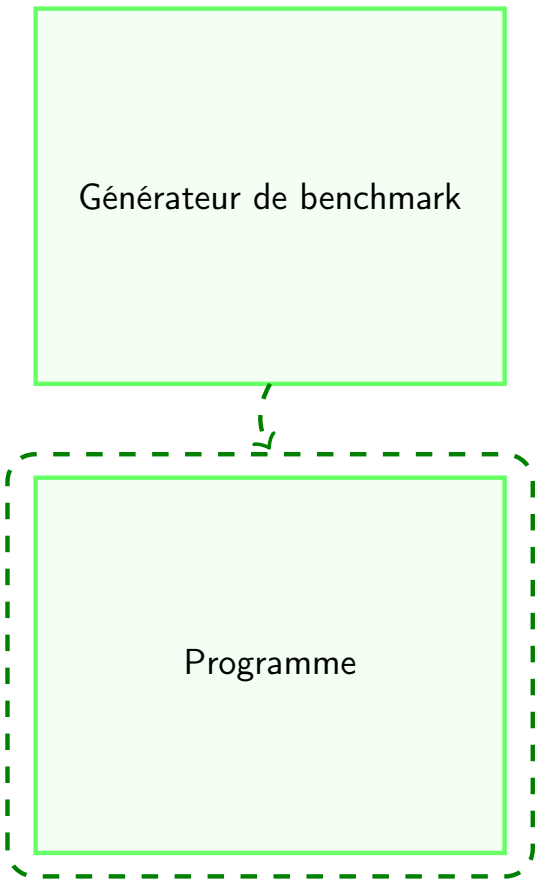
On voudrait détecter automatiquement des bugs de performance.

Introduction

On voudrait détecter automatiquement des bugs de performance.

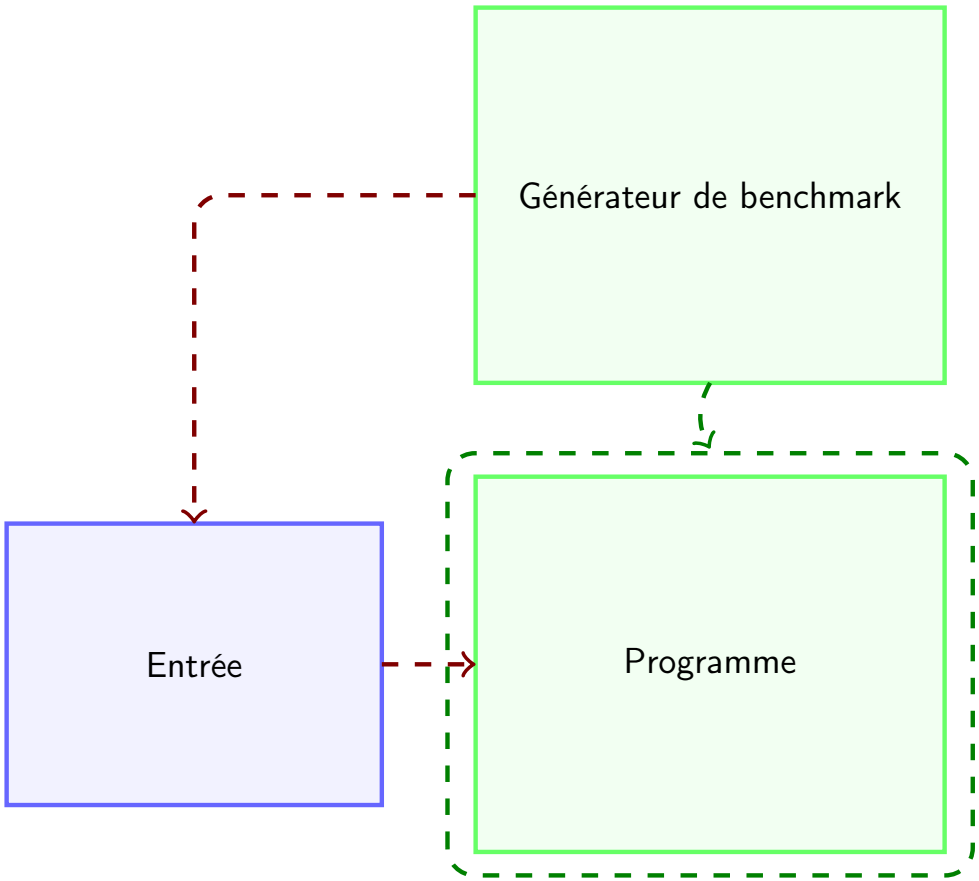
Introduction

On voudrait détecter automatiquement des bugs de performance.



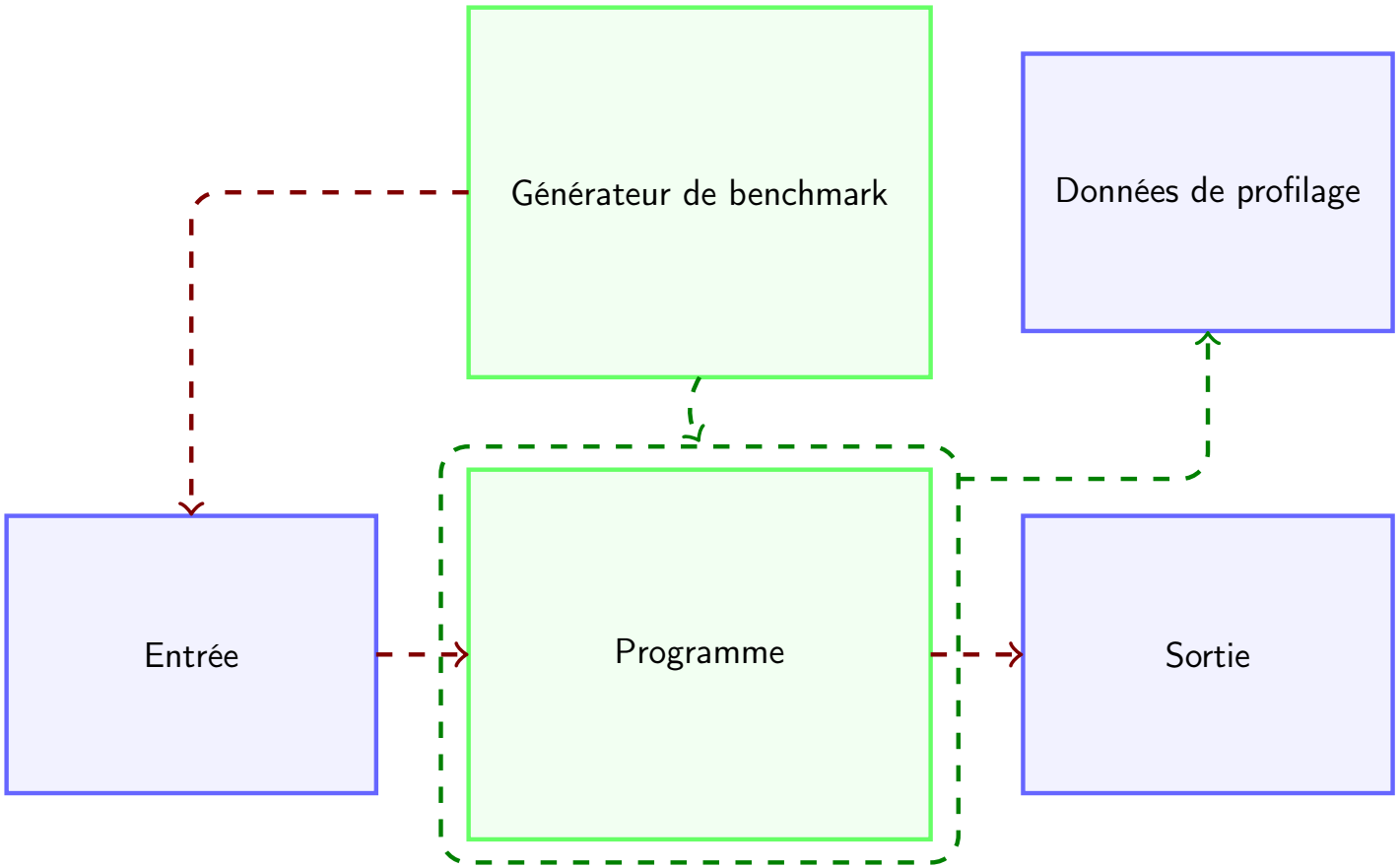
Introduction

On voudrait détecter automatiquement des bugs de performance.



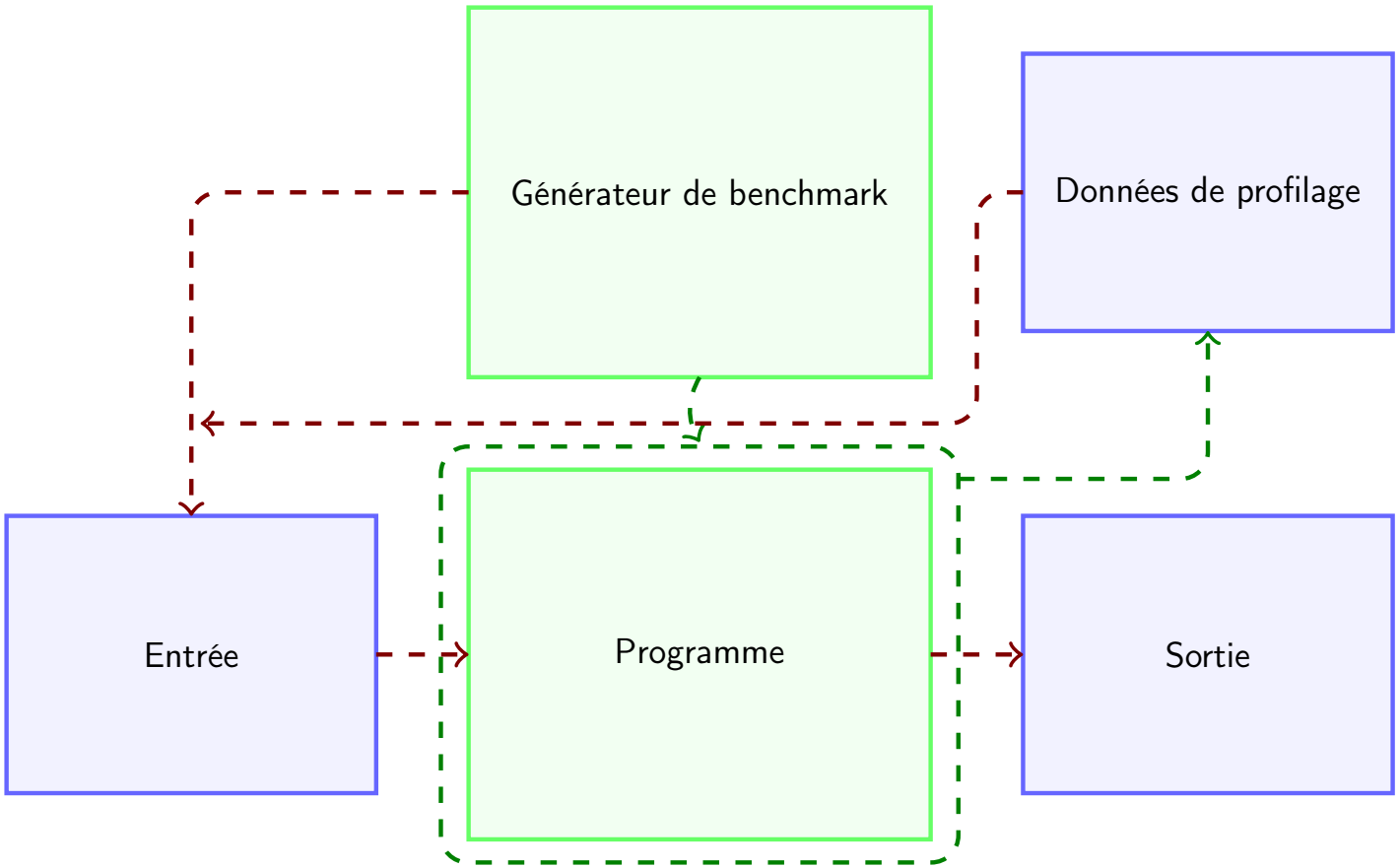
Introduction

On voudrait détecter automatiquement des bugs de performance.



Introduction

On voudrait détecter automatiquement des bugs de performance.



Définir un bug de performance

La question à laquelle on essaye de répondre de façon automatique est:

> Une trace d'exécution donnée exhibe-t-elle un bug de performance?

Définir un bug de performance

La question à laquelle on essaye de répondre de façon automatique est:

~~Une trace d'exécution donnée exhibe-t-elle un bug de performance?~~

Quelle est la performance d'une trace d'exécution donnée?

Définir un bug de performance

La question à laquelle on essaye de répondre de façon automatique est:

~~Une trace d'exécution donnée exhibe-t-elle un bug de performance?~~

Quelle est la performance d'une trace d'exécution donnée?

On se ramène donc à un problème d'optimisation:

Trouver l'entrée qui minimise la performance du programme.

Définir un bug de performance

La question à laquelle on essaye de répondre de façon automatique est:

~~Une trace d'exécution donnée exhibe-t-elle un bug de performance?~~

Quelle est la performance d'une trace d'exécution donnée?

On se ramène donc à un problème d'optimisation:

Trouver l'entrée qui minimise la performance du programme.

Reste à définir la "performance" d'un programme.

Définir la performance

Intuitivement, (l'inverse de) la performance d'un programme correspond à son temps d'exécution. Mais...

Définir la performance

Intuitivement, (l'inverse de) la performance d'un programme correspond à son temps d'exécution. Mais...

- Le temps d'exécution est très instable.

Définir la performance

Intuitivement, (l'inverse de) la performance d'un programme correspond à son temps d'exécution. Mais...

- Le temps d'exécution est très instable.
- Le temps d'exécution dépend de la machine, et n'est donc pas reproductible.

Définir la performance

Intuitivement, (l'inverse de) la performance d'un programme correspond à son temps d'exécution. Mais...

- Le temps d'exécution est très instable.
- Le temps d'exécution dépend de la machine, et n'est donc pas reproductible.

On choisit de mesurer le nombre d'envois de messages

Définir la performance

Intuitivement, (l'inverse de) la performance d'un programme correspond à son temps d'exécution. Mais...

- Le temps d'exécution est très instable.
- Le temps d'exécution dépend de la machine, et n'est donc pas reproductible.

On choisit de mesurer le nombre d'envois de messages, qui est

- fortement corrélé au temps moyen d'exécution;

Définir la performance

Intuitivement, (l'inverse de) la performance d'un programme correspond à son temps d'exécution. Mais...

- Le temps d'exécution est très instable.
- Le temps d'exécution dépend de la machine, et n'est donc pas reproductible.

On choisit de mesurer le nombre d'envois de messages, qui est

- fortement corrélé au temps moyen d'exécution;
- très stable;

Définir la performance

Intuitivement, (l'inverse de) la performance d'un programme correspond à son temps d'exécution. Mais...

- Le temps d'exécution est très instable.
- Le temps d'exécution dépend de la machine, et n'est donc pas reproductible.

On choisit de mesurer le nombre d'envois de messages, qui est

- fortement corrélé au temps moyen d'exécution;
- très stable;
- indépendant de la machine.

Définir la performance

Intuitivement, (l'inverse de) la performance d'un programme correspond à son temps d'exécution. Mais...

- Le temps d'exécution est très instable.
- Le temps d'exécution dépend de la machine, et n'est donc pas reproductible.

On choisit de mesurer le nombre d'envois de messages, qui est

- fortement corrélé au temps moyen d'exécution;
- très stable;
- indépendant de la machine.

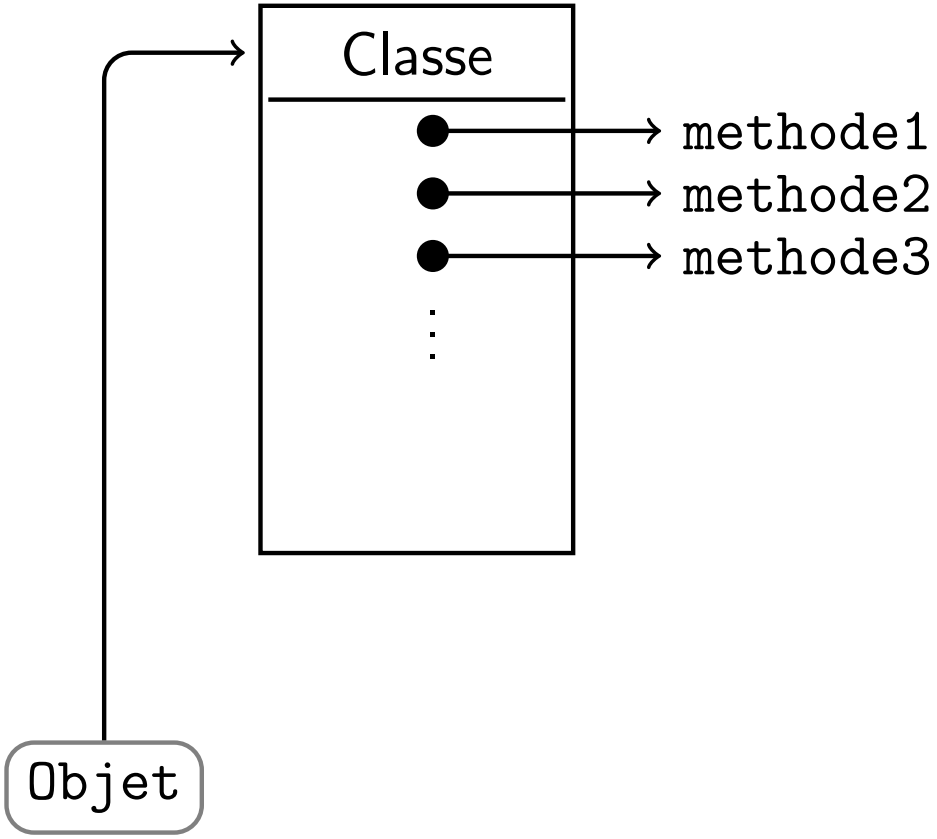
D'autres critères pourraient être considérés (temps passé dans le GC, mémoire consommée, nombre d'objets alloués, ...).

Instrumentation au niveau de l'image

Pour compter le nombre d'appels de méthodes, on choisit d'instrumenter le programme.

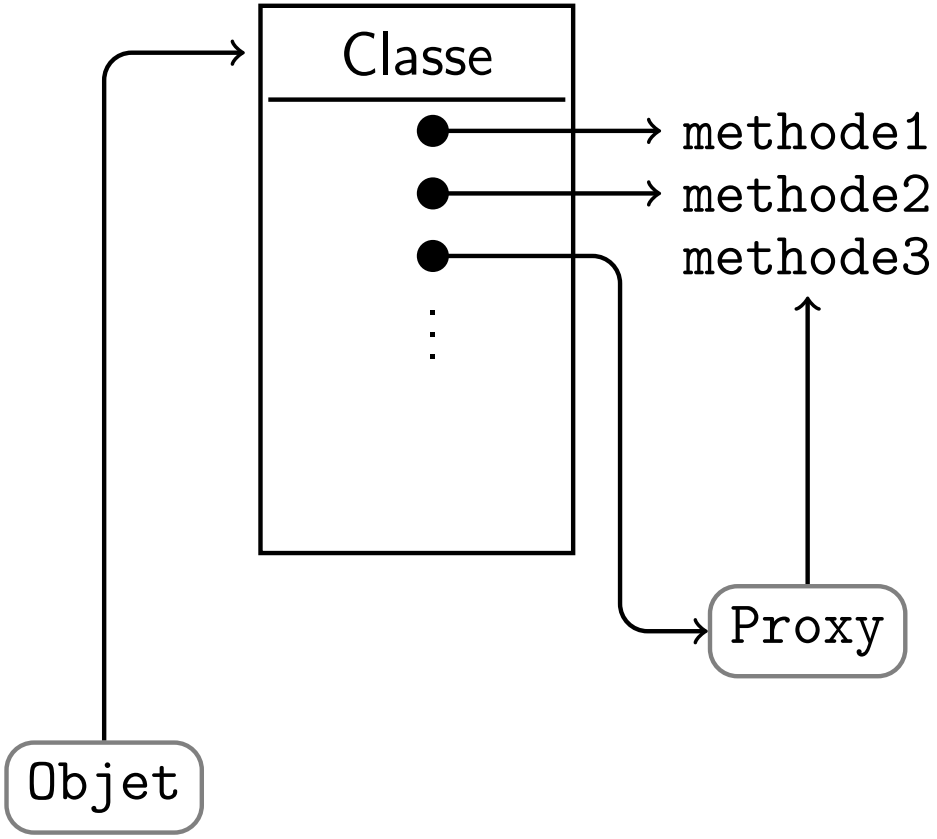
Instrumentation au niveau de l'image

Pour compter le nombre d'appels de méthodes, on choisit d'instrumenter le programme.
Supposons vouloir instrumenter `methode3`.



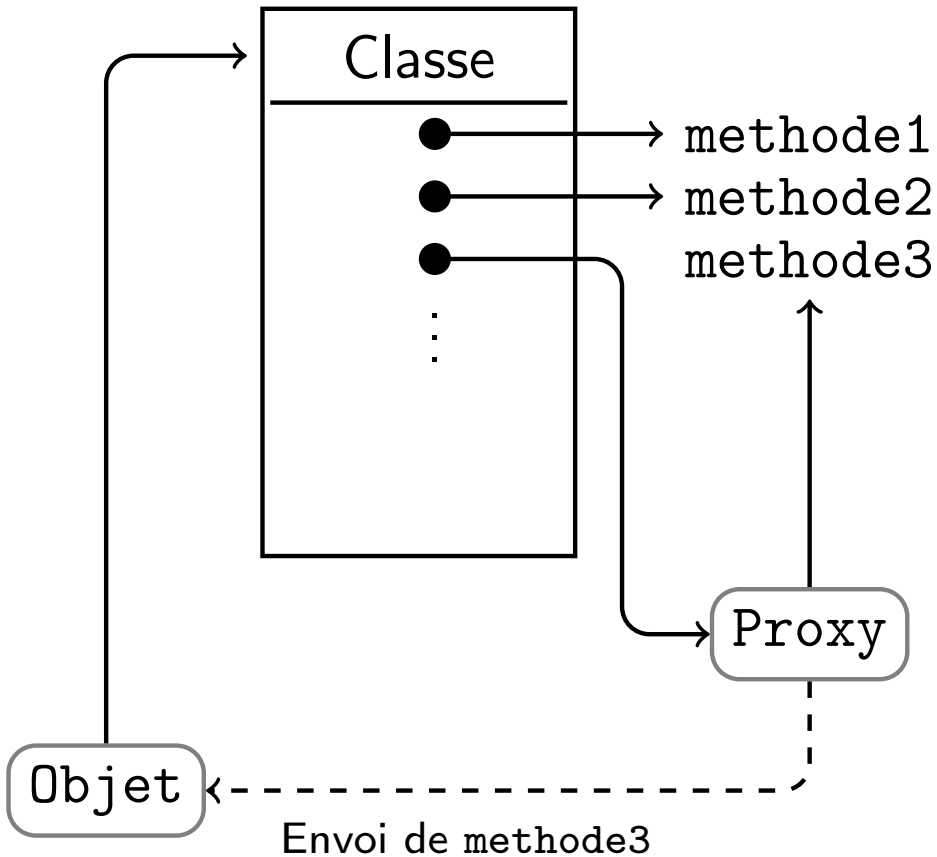
Instrumentation au niveau de l'image

Pour compter le nombre d'appels de méthodes, on choisit d'instrumenter le programme.
Supposons vouloir instrumenter `methode3`.



Instrumentation au niveau de l'image

Pour compter le nombre d'appels de méthodes, on choisit d'instrumenter le programme.
Supposons vouloir instrumenter `methode3`.



On tombe dans un problème de *méta-récurrence*.

Instrumentation au niveau de la VM

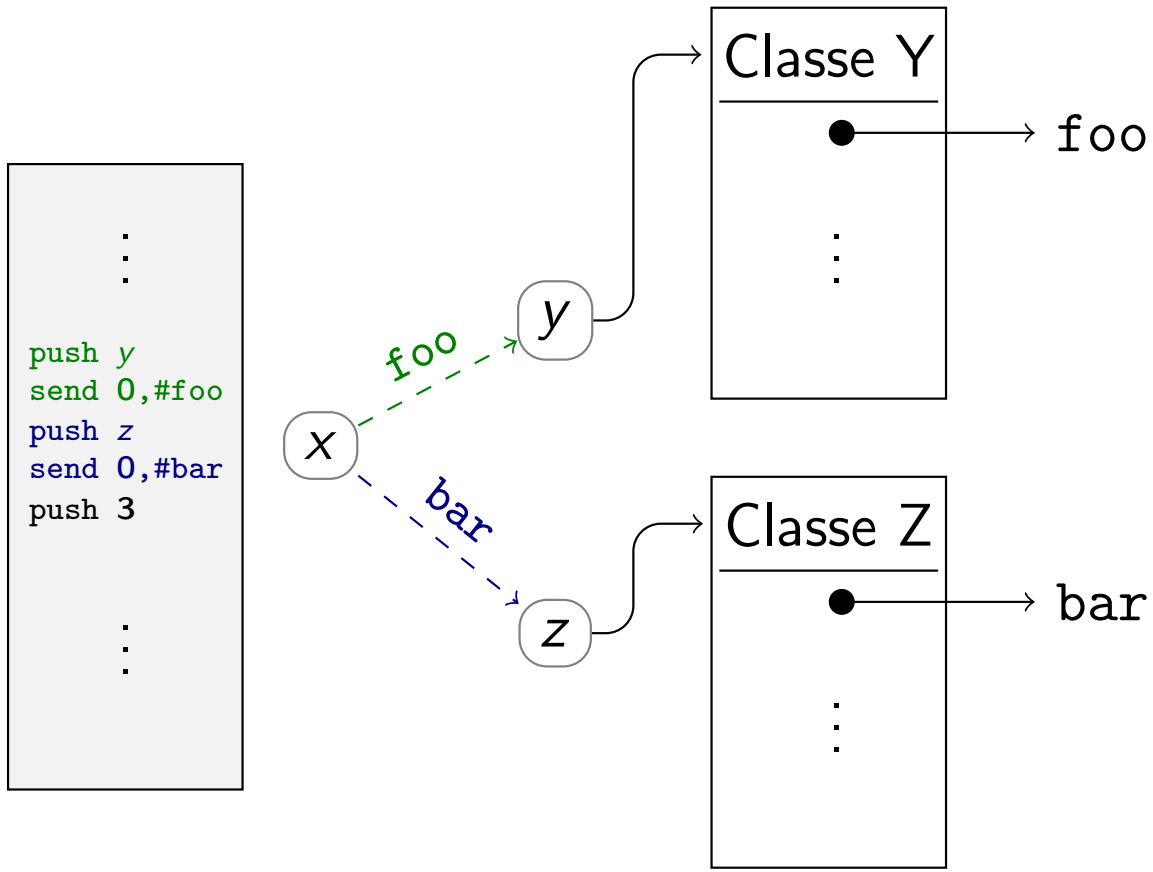
Résoudre les problèmes de méta-récurrence est possible, mais cher: le surcoût peut atteindre 1000000%!

Instrumentation au niveau de la VM

Résoudre les problèmes de méta-récurrence est possible, mais cher: le surcoût peut atteindre 1000000%!
Pour résoudre ce problème, on place nos outils au niveau de la machine virtuelle.

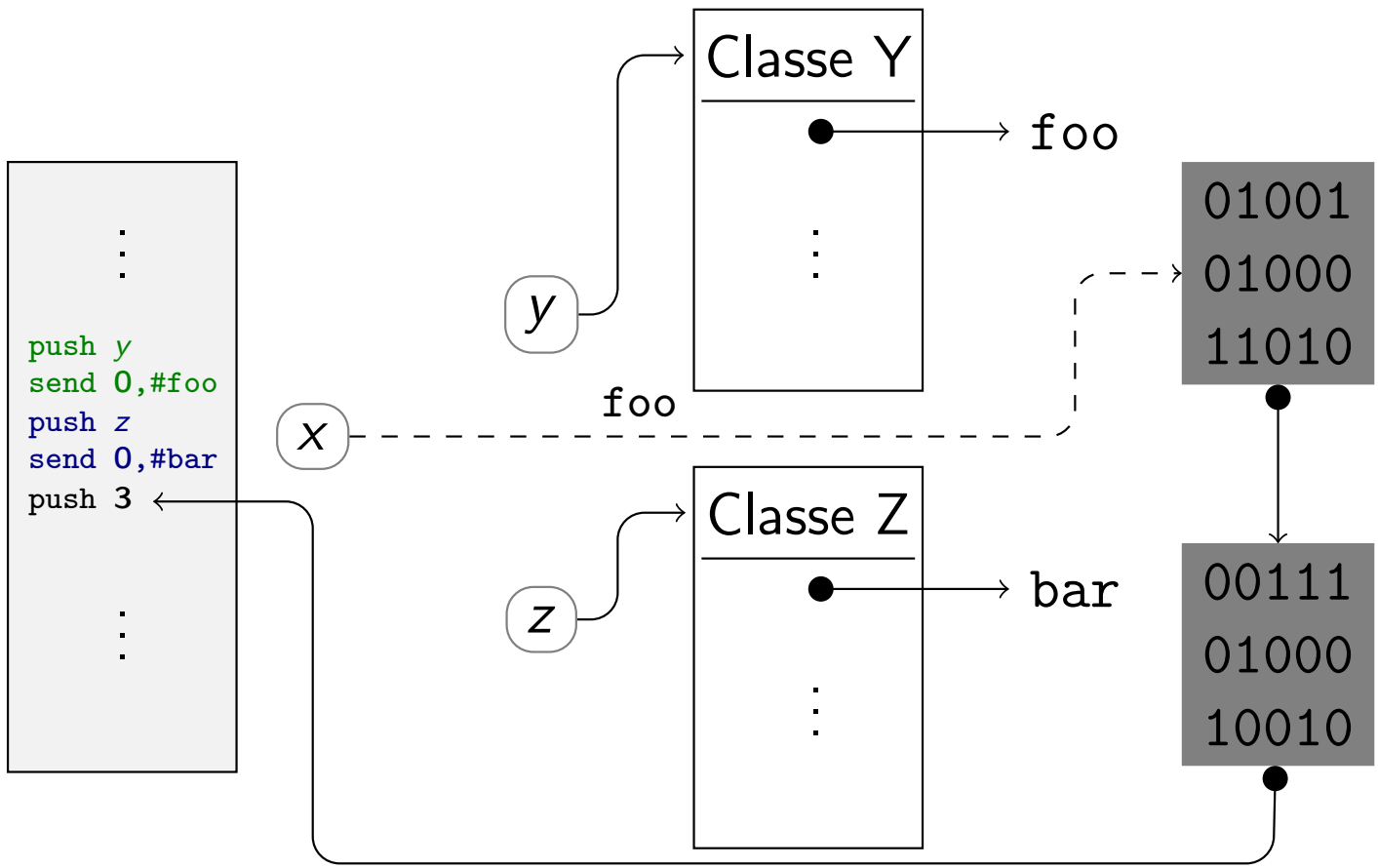
Instrumentation au niveau de la VM

Résoudre les problèmes de méta-récurrence est possible, mais cher: le surcoût peut atteindre 1000000%!
Pour résoudre ce problème, on place nos outils au niveau de la machine virtuelle.



Instrumentation au niveau de la VM

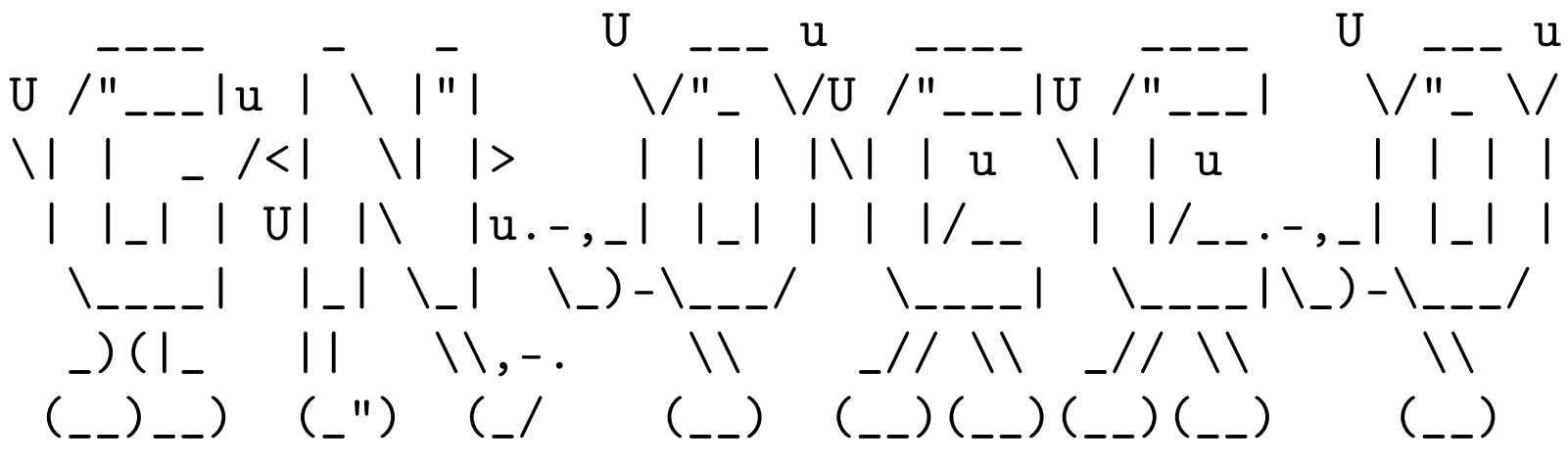
Résoudre les problèmes de méta-récurrence est possible, mais cher: le surcoût peut atteindre 1000000%!
Pour résoudre ce problème, on place nos outils au niveau de la machine virtuelle.



Comment générer des entrées valides?

Comment générer des entrées valides? On a recours à des **grammaires génératives**.

Comment générer des entrées valides? On a recours à des **grammaires génératives**.
Pour cela, faisons un détour par `Gnocco`.



Gnocco

`Gnocco` est un DSL qui permet de définir des grammaires génératives en Pharo.

Gnocco

`Gnocco` est un DSL qui permet de définir des grammaires génératives en Pharo.
Le langage s'intègre naturellement avec Pharo, mais vient avec son lot d'analyse statique.

Gnocco

`Gnocco` est un DSL qui permet de définir des grammaires génératives en Pharo.
Le langage s'intègre naturellement avec Pharo, mais vient avec son lot d'analyse statique.
`Gnocco` permet aussi d'ajouter des contraintes sur l'arbre de dérivation avant de commencer la génération.

Une grammaire

Supposons vouloir définir la grammaire suivante:

$$
\begin{aligned}
A &\rightarrow \mathtt{a}\ A\ \mathtt{b} \\
&\mid \mathtt{a} \\
&\mid B \\
B &\rightarrow \mathtt{c}\ B\ \mathtt{d} \\
&\mid \mathtt{c}
\end{aligned}
$$

Non-terminal initial: A.

Une grammaire

Supposons vouloir définir la grammaire suivante:

$$
\begin{aligned}
A &\rightarrow \mathtt{a}\ A\ \mathtt{b} \\
&\mid \mathtt{a} \\
&\mid B \\
B &\rightarrow \mathtt{c}\ B\ \mathtt{d} \\
&\mid \mathtt{c}
\end{aligned}
$$

```
1  A --> 'a', A, 'b'
2      | 'a'
3      | B.
4  B --> 'c', B, 'd'
5      | 'c'.
6  ^ A
```

Non-terminal initial: A.

Muter les grammaires

Pour pouvoir guider la génération, il faut pouvoir agir dessus.

Muter les grammaires

Pour pouvoir guider la génération, il faut pouvoir agir dessus. Pour cela, chaque règle d'une grammaire peut être annotée d'un poids:

```
1  A --> 'a', A, 'b' @ 5
2      | 'a'
3      | B @ 3.
4  B --> 'c', B, 'd' @ 5
5      | 'c'.
6  ^ A
```

Muter les grammaires

Pour pouvoir guider la génération, il faut pouvoir agir dessus. Pour cela, chaque règle d'une grammaire peut être annotée d'un poids:

```
1  A --> 'a', A, 'b' @ 5
2      | 'a'
3      | B @ 3.
4  B --> 'c', B, 'd' @ 5
5      | 'c'.
6  ^ A
```

Ces poids peuvent être changés programmatiquement.

Muter les grammaires

Pour pouvoir guider la génération, il faut pouvoir agir dessus.
Pour cela, chaque règle d'une grammaire peut être annotée d'un poids:

```
1  A --> 'a', A, 'b' @ 5
2      | 'a'
3      | B @ 3.
4  B --> 'c', B, 'd' @ 5
5      | 'c'.
6  ^ A
```

Ces poids peuvent être changés programmatiquement.
Ils permettent un contrôle fin de la direction d'exploration de l'espace des données avec peu de paramètres.

Diriger la mutation

Pour diriger la mutation, on utilise des métaheuristiques.

Diriger la mutation

Pour diriger la mutation, on utilise des métaheuristiques.

Definition

Une métaheuristique est un algorithme stochastique itératif d'optimisation, visant à trouver des approximations de maxima globaux d'un espace de recherche.

Diriger la mutation

Pour diriger la mutation, on utilise des métaheuristiques.

Definition

Une métaheuristique est un algorithme stochastique itératif d'optimisation, visant à trouver des approximations de maxima globaux d'un espace de recherche.

Deux métaheurstiques sont disponibles actuellement:

Diriger la mutation

Pour diriger la mutation, on utilise des métaheuristiques.

Definition

Une métaheuristique est un algorithme stochastique itératif d'optimisation, visant à trouver des approximations de maxima globaux d'un espace de recherche.

Deux métaheurstiques sont disponibles actuellement:

- algorithme génétique;

Diriger la mutation

Pour diriger la mutation, on utilise des métaheuristiques.

Definition

Une métaheuristique est un algorithme stochastique itératif d'optimisation, visant à trouver des approximations de maxima globaux d'un espace de recherche.

Deux métaheurstiques sont disponibles actuellement:

- algorithme génétique;
- recuit simulé.

Benchmarker les Regex

On compare l'implémentation d'expressions régulières de la librairie standard de Pharo avec une faite pour l'occastion.

Benchmarker les Regex

On compare l'implémentation d'expressions régulières de la librairie standard de Pharo avec une faite pour l'occastion.

```
Regex --> Concat | Concat, '|', Regex.
Concat --> Modifier | Modifier, Concat.
Modifier --> Atom | Atom, '?'
           | Atom, '*' | Atom, '+'.
Atom --> ($a-$z) | '.' | '(', Regex, ')'.
^ Regex
```

Benchmarker les Regex

On compare l'implémentation d'expressions régulières de la librairie standard de Pharo avec une faite pour l'occastion.

```
Regex --> Concat | Concat, '|', Regex.
Concat --> Modifier | Modifier, Concat.
Modifier --> Atom | Atom, '?'
           | Atom, '*' | Atom, '+'.
Atom --> ($a-$z) | '.' | '(', Regex, ')'.
^ Regex
```

En une dizaine de seconde, on trouve:

```
((a|(a|(c|a|a|a|a|(a|a|a|a)|a)|a)|c)+)a.
```

qui est 1000 fois plus lente avec l'implémentation de la librairie standard!