UFR 919 Informatique – Master Informatique

Spécialité STL – UE ILP

TME7 — Visiteurs

Christian Queinnec

Cette séance est consacrée à la mise en œuvre de visiteurs pour remplacer les différents algorithmes de parcours d'arbres présents dans ILP4.

1 État des lieux

Objectif : Identifier et comprendre les différents algorithmes de parcours d'arbres déjà utilisés d'ILP1 à ILP4.

Buts :

— déterminer les algorithmes de parcours,
— pour chacun, décrire les avantages, les inconvénients et comment ajouter de nouveaux comportements si l'on ajoute de nouveaux types d'AST ?

1.1 Travail à réaliser

Construire une matrice croisant les algorithmes, leurs modes d'emploi, les avantages, les inconvénients et leur extensibilité. Ω

2 Visiteurs

Objectif : Remplacer les différents algorithmes existants de parcours d'arbre AST à l'aide du design pattern visiteur.

Buts :

— comprendre le design pattern visiteur,
— savoir mettre en oeuvre le design pattern visiteur.

2.1 Introduction

Quelques visiteurs sont déjà présents dans les sources d'ILP4 : `GlobalCollector` et `XMLwriter`. Toute l'infrastructure pour ajouter de nouveaux visiteurs est présente dans les interfaces `IAST4*`. On souhaite remplacer les algorithmes précédents par de nouveaux visiteurs.

Pour cela créer le paquetage `fr.upmc.ilp.ilp4visitor` qui contiendra vos interfaces et classes. Dans un premier temps, vous pourrez modifier la méthode `prepare` de `Process` pour utiliser `GlobalCollector`.

2.2 Intégration de fonctions

Objectif : Implanter un visiteur pour remplacer la méthode `inline`. Votre visiteur ne doit donc plus, bien sûr, utiliser la méthode `inline` !

Travail à réaliser

Créer la classe `VisitorInlining` (dans un sous-paquetage `ast`) héritant de `AbstractExplicitVisitor` pour remplacer la méthode `inline` dans la méthode `prepare` de `Process`. On pourra utiliser l'argument `data` des méthodes `visit` pour fournir une `IAST4Factory` nécessaire pour la construction de nouveaux noeuds AST. Penser en premier avec quels types il faut instancier `AbstractExplicitVisitor`.

2.3 Calcul des fonctions invoquées

Objectif : Implanter un visiteur pour remplacer la méthode `computeInvokedFunctions`.

Travail à réaliser

Créer la classe `InvokedFunctionVisitor` (dans le sous-paquetage `ast`) héritant de `AbstractExplicitVisitor` pour remplacer la méthode `computeInvokedFunctions` dans la méthode `prepare` de `Process`. Utiliser `Set<IAST4globalFunctionVariable>` comme type de l'argument `data` des méthodes `visit` pour stocker les fonctions invoquées.

2.4 Évaluation

Objectif : Implanter un visiteur pour remplacer la méthode `eval`. On pourra écrire les méthodes au fur et à mesure des tests de non-régression.

Travail à réaliser

Créer la classe `VisitorEvaluator` pour remplacer les méthodes `eval`. Pour pouvoir passer les environnements lexical et global dans l'argument `data` des méthodes `visit`, créer une classe `Context` implémentant l'interface `IContext` :

```
1 public interface IContext {
2     ILexicalEnvironment getLexicalEnvironment();
3     ICommon getCommonEnvironment();
4 }
```

Dans cette question, toutes les classes et interfaces créées doivent être placées dans un sous-paquetage `eval`.