DLP : Développement d'un langage de programmation

Master STL, Université Pierre et Marie Curie

Année 2020–2021

TME 4 – Introduction d'un nouveau trait en ILP2

Jacques Malenfant, Aurélien Moreau, Christian Queinnec & Antoine Miné

Objectif : ajouter le mot-clé `unless` à ILP2.

Buts :
- étendre le langage avec un nouveau mot-clé ;
- étendre une grammaire ANTLR 4 ;
- étendre l'AST avec un nouveau type de nœud ;
- effectuer des transformations sur l'AST ;
- comparer différentes méthodes d'extension et mesurer les conséquences sur l'implantation.

Nous considérons la construction suivante, trouvée par exemple dans le langage Perl :

body `unless` *condition*

où *body* et *condition* sont des expressions. L'effet de cette construction est d'exécuter l'expression *body* uniquement si l'expression *condition* s'évalue en faux. Cette construction peut donc être vue comme du « sucre syntaxique » équivalent à la construction suivante :

`if !`*condition* `then` *body*

Une manière simple pour ajouter `unless` à notre langage consiste donc à traduire `unless` en `if` équivalent avant l'interprétation ou la compilation proprement dite (dans le *front-end*). Ainsi, les classes `Interpreter` et `Compiler` restent inchangées. A priori, nous pouvons effectuer cette traduction à deux endroits :

1. dans le *Listener* de la grammaire ANTLR 4 ;
2. dans une passe séparée qui transforme un AST avec `unless` en AST sans `unless` (celle-ci s'implante naturellement avec un motif *visiteur*).

L'avantage de la première solution est que nous n'avons pas besoin d'ajouter de nœud AST correspondant à `unless` ; cependant, elle a l'inconvénient d'ajouter un couplage entre l'analyse syntaxique et la simplification de l'AST, alors qu'il vaut mieux garder ces étapes séparées.

3. En plus de ces deux solutions à base de transformation, une troisième solution consiste à conserver le nœud `unless` jusqu'à `Interpreter` et `Compiler`, et à les traiter dans ces classes (c'est la solution la plus générale pour ajouter un trait au langage).

Nous allons implanter les trois méthodes afin de bien évaluer les forces et les faiblesses de chacune.

Notes : Nous travaillons désormais avec ILP2, qui est une extension d'ILP1. Le code d'ILP2 est disponible sur le GitLab du cours (`https://stl.algo-prog.info`), dans le projet `ILP2` du groupe du semestre en cours (`DLP-2020oct` pour 2020–2021). Comme au TME 1, vous devrez commencer par en faire un *fork* privé, puis importer ce nouveau projet dans Eclipse. Référez-vous à nouveau aux instructions du document *Environnement des TME* (TME 0) sur la page du cours pour plus de détails. Après ces étapes, vous devrez avoir un projet sous Eclipse avec le mention `[ilp2 master]`. Ce projet sera utilisé pour les TME 4, 5 et 6, avant de passer à ILP3 puis ILP4 pour les TME suivants. Vous n'avez pas besoin d'intégrer les TME 1 à 3 d'ILP1 dans ce projet.

Travail demandé :

- vérifiez qu'ILP2 est bien installé et fonctionne ;
- dans le *package* `com.paracamplus.ilp2.ilp2tme4.parser.ilpml`, créez une nouvelle grammaire `ILPMLGammar2tme4` qui reprend `ILPMLGrammar2` mais y ajoute la construction `unless` ;
- ajoutez une classe `ASTunless` (et l'interface correspondante) pour notre nouveau nœud (il sera utilisé dans les méthode 2 et 3) ;
- implantez les trois méthodes de gestion de `unless` proposées ci-dessus ;
- créez des classes `InterpreterTest` et `CompilerTest` ;
  tester votre implantation sur les exemples ILP2 existants (tests de régression) et sur des programmes ILP avec `unless` que vous créerez pour l'occasion ;
- modifiez le fichier `.gitlab-ci.yml` pour vous assurer que ces tests sont effectués sur le serveur GitLab.

Rendu : Comme pour les TME précédents, vous effectuerez un rendu en vous assurant que tout le code développé a été envoyé sur le serveur GitLab (*push*), mais cette fois dans votre *fork* d'ILP2. Vous vous assurerez que les tests d'intégration continue développés lors de ce TME ont été configurés (fichier `.gitlab-ci.yml`) et fonctionnent sur le serveur. Vous ajouterez un tag « rendu-initial-tme4 » en fin de séance, puis un tag « rendu-final-tme4 » quand le TME est finalisé.