Éléments de correction
Examen réparti de fin de semestre d'ILP

Christian Queinnec

15 décembre 2010

Un corrigé est déjà disponible en ligne. Ce document n'ajoute que quelques remarques plus ou moins générales, suite à la correction des copies.

Présentez vos programmes correctement, non seulement je n'aime pas les programmes mal présentés mais en plus vous vous pénalisez vous-même car les programmes mal présentés sont plus souvent faux que ceux bien présentés.

Un sur cinq parmi vous a encore des difficultés avec l'héritage ! Un copier-coller d'une classe ILP4 (héritant d'ILP3) mène à une classe qui hérite d'ILP3, pas d'ILP4 !

La façon la plus simple (à mon sens) de traiter ce problème était de construire un visiteur construisant l'AST simplifié à partir de l'AST initial. La simplification d'une affectation d'une variable à elle-même exigeait que la normalisation ait eu lieu afin de pouvoir comparer des variables. Surcharger la normalisation ou encombrer l'analyse syntaxique permettait d'assurer certaines simplifications mais pas toutes et pas nécessairement simplement au niveau du code à produire.

Un seul visiteur était préférable à de multiples visiteurs qu'il aurait fallu appliquer de nombreuses fois car une simplification peut en amener d'autres de nature différente. Bien sûr, avant d'appliquer toute simplification sur un nœud, il fallait s'assurer que les fils de ce nœud était déjà simplifiés ou que le résultat de la simplification devait lui aussi être simplifié.

Ajouter à toute expression un champ `simplified` demandait de redéfinir toutes les classes (un peu comme ILP4 vis-à-vis d'ILP3) ce qui était trop de travail. Il n'y avait cependant pas beaucoup d'intérêt à conserver la version originale non simplifiée.

N'oubliez pas qu'en ILP, tout ce qui n'est pas faux est vrai donc 0 est vrai. La condition d'une alternative, l'opérande d'une négation logique peuvent être des constantes non booléennes. Par ailleurs, ne confondez pas `IAST4boolean` et `Boolean`.

Dans la graphie `let v = e in ...`, le signe = ne dénote pas une affectation mais une initialisation.

Ce n'est pas `if (not constante) ...` que l'on simplifie c'est `not constante` qui à son tour déclenche la simplification de l'alternative.

Une boucle infinie n'est pas une anomalie : en Ada, il n'y a que la boucle infinie.

N'oubliez pas que toute expression a une valeur.

En Java : `o instanceof X` est différent de :

```
o.getClass().getName().equals("X")
```

La méthode `equals` sur les CEAST ne fait pas de comparaison structurelle de contenu.

Une remarque intéressante qu'ont faite certains est que `true * 0` ne peut être simplifié en `0` car l'expression originelle signale une erreur que ne signale pas la seconde. Une remarque plus subtile est que `1.1 * 0` vaut `0.0` et non `0`.

Une manipulation astucieuse pour évaluer des expressions était de tenter un `expr.eval(null, null)` !