Éléments de correction
Examen réparti de mi-semestre d'ILP

Christian Queinnec

12 novembre 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.

1 Questions

1.1 Grammaire (3 points)

Une directive `var` ne possède qu'un seul paramètre : le nom de la variable introduite. Cette directive n'a pas de corps mais elle a pour portée le reste des instructions de la séquence où elle apparaît.

1.2 Exemples (5 points)

Les programmes doivent bien sûr être corrects vis-à-vis de la grammaire précédente. L'exemple conduisant à une erreur doit utiliser un `try catch` afin de pouvoir tourner automatiquement sans requérir un oeil humain : il fallait donc étendre ILP3 dans la grammaire et non seulement ILP2.

Coder les deux exemples de l'énoncé était un pis-aller. De nombreux autres cas devaient être imaginés. En vrac :

- `var` en tête de séquence
- `var` pas en tête de séquence
- au moins deux `var` en tête de séquence
- au moins deux `var` n'importe où dans une séquence
- deux `var` introduisant le même nom
- `var` introduisant un nom déja dans l'environnement (local ou global)

Mais de toute façon, il valait mieux dire ce que le programme était censé tester plutôt que laisser le correcteur tenter de deviner.

1.3 Stratégie d'interprétation (6 points)

Trop de transtypages sauvages ! Quand vous écrivez une expression de type A alors qu'Eclipse vous dit qu'il attend B, la solution n'est pas d'ajouter systématiquement `(A)` : il faut un peu réfléchir ! Si, par exemple, B est une entité n'existant qu'à l'exécution (runtime), la transtyper vers un programme (entité syntaxique) ou un élément DOM est insensé ! Et inversement !

Conserver à l'évaluation (`eval` ou `compile`), un pointeur vers un analyseur syntaxique (`parser`) n'a pas plus de sens.

Nommer `CEASTvar` la classe implantant la déclaration de variable ne me paraît pas approprié : la proximité linguistique avec `CEASTvariable` est trop forte et induit des erreurs. Par exemple, `CEASTvar` est une instruction alors que `CEASTvariable` n'en est

pas une. Utiliser des instructions comme clé pour rechercher une information dans un environnement de variable est pour le moins confondant.

Utilisez l'héritage ! On voit encore trop de copier-coller de classes ou de paquetages entiers noyant les quelques lignes intéressantes au sein d'une immensité de code qui gagnerait à être partagé et qui ainsi démontrerait votre maîtrise.

Enfin, lisez aussi les consignes et évitez, comme indiqué, les lignes trop longues.

Une façon simple de traiter l'interprétation était de modifier l'évaluation de la séquence afin que l'évaluation d'une déclaration enrichisse l'environnement d'interprétation de cette séquence. Ainsi les instructions qui suivent bénéficient d'un environnement enrichi. Il fallait toutefois se rendre compte que l'extension d'un environnement a pour valeur un nouvel environnement et ne modifie pas l'environnement étendu.

Une autre façon de faire (celle qui figure dans le corrigé) était d'effectuer une transformation d'AST afin de convertir

```
{ ...1                  { ...1
  var x                   { letundef x
  ...2          en          ...2
  ...(x)...                 ...(maybeUninitialized(x))...
}                         }
                        }
```

Dans ce pseudo-code, letundef est similaire à un bloc unaire sauf qu'il introduit une variable non initialisée. Il faut donc, dans toutes les références à cette variable, vérifier qu'elle est bien initialisée et pour cela, il suffit de créer une variante de `CEASTreference` qui vérifie cela.

Identifier les déclarations et les migrer en tête de bloc ne peut convenir car cela modifie la portée de ces variables.

1.4 Stratégie de compilation (6 points)

La première méthode indiquée pour l'interprétation peut aussi être utilisée pour la compilation en effectuant à la volée l'équivalent de la transformation d'AST mentionnée en seconde méthode.