Éléments de correction
Examen final réparti d'ILP

Christian Queinnec

10 janvier 2013

Voici quelques éléments de correction de l'examen de janvier 2013. Cet examen s'appuyait sur le TME3 (le code était dans le dépôt Mercurial) et n'avait aucun rapport avec l'examen de décembre 2008.

Copier/coller et héritage font mauvais ménage ! Copier la classe SC qui hérite de C ne crée pas une classe raffinant SC mais une classe sœur de SC et donc perdant tout ce que SC apportait. Par ailleurs, il était demandé d'étendre ilp4 et non ilp1 ou ilp6 !

Nombreux sont ceux qui ont donné un nom aux vecteurs !? Un vecteur est une structure de donnée qui n'a pas besoin de nom pour exister. Par ailleurs, cette structure de donnée est initialisable avec des expressions et pas uniquement des constantes (il suffisait de regarder l'exemple fourni pour s'en rendre compte). D'ailleurs pourquoi enchâsser profondément les expressions comme dans ce qui suit ? L'élément V1 est un conteneur suffisant qui n'a pas besoin d'un fils qui serait un conteneur.

```
element V1 {
   element V2 {
     expression *
   }
}
```

Il était demandé d'écrire des fonctions manipulant des vecteurs, étendre la grammaire pour donner des syntaxes particulières pour ces opérations était donc inutile. Une extension grammaticale n'était nécessaire que pour la graphie de création de vecteurs en question 3.

Si l'accès aux vecteurs est vérifié en Java, ce n'est pas le cas en C. Pour qu'ILP reste un langage sûr, il faut donc vérifier les accès (sinon, c'est une porte ouverte pour les pirates qui peuvent ainsi analyser tout le contenu de la mémoire).

isVector appliqué à une valeur qui n'est pas un vecteur n'est pas une erreur sinon il n'y aurait aucun moyen de se prémunir de cette erreur :)

En ILP, tout ce qui n'est pas faux est vrai. Il ne faut donc pas écrire `= Boolean.TRUE` mais plutôt `!= Boolean.FALSE`.

Prêtez attention à ce que vous dit Eclipse ou gcc. Il serait étonnant de ne pas voir la réaction d'Eclipse quand on écrit ce qui suit.

```
BigInteger bi;     ...        (int) bi
if ( o instanceof BigInteger ) { ... foo((String) o) ... }
```

Ce sont des erreurs statiques et certaines auraient pu être évitées si les programmes étaient correctement indentés. Écrire du C sans au moins vérifier qu'il n'y a pas d'anomalie ne demande que quelques secondes et évite des déboires. Écrire du RelaxNG compact et convertir en RNG ne demande que quelques secondes et évite des déboires.

Ne confondez pas héritage et surcharge. Dans ce qui suit, la methode A ne sera pas invoquée :

```
public Object foo () { ... }
```

```
public Object invoke (Object[] o) { ... } // methode A
Object o = foo();
if ( o instanceof Object[] ) { ...  this.invoke(o); ... }
```

Toujours en Java, voici quelques amusantes variantes pour tester que o est un vecteur :

```
o instanceof Object[]
o.getClass() == (new Object[0]).getClass()
try { (Object[]) o; ... } catch (ClassCastException e) {...}
(new isVectorPrimitive()).invoke(o) != Boolean.FALSE
```

Et, en C, une variante inefficace et compliquée à écrire :

```
memcpy(v[i], a, sizeof(ILP_Object))
/* au lieu de */ v[i] = a
```

Toujours en C, évitez `*(o->_content.asVector._vector+i)` qui est illisible.

Peu ont regardé le code C du TME3 et ont vu qu'une seule allocation permettait de construire un vecteur. Deux appels à malloc sont moins efficaces, consomment plus de mémoire et introduisent des zones mémoires non toutes identifiables par un entête donc plus compliquées à gérer car séparées. Et puisque l'on parle d'efficacité, le coût de cette vérification est stupéfiant :

```
ILP_GreaterThan(i, ILP_make_integer(0))
/* plutot que (i est un entier ILP) */ i->_content.asInteger > 0
```

Pour la question 1, il suffisait d'adapter le code du TME3 à ILP4 et ainsi de définir les cinq fonctions demandées. Le code du TME3 n'était pas sûr, il fallait donc ajouter les vérifications nécessaires et changer de représentation pour les vecteurs.

Pour la compilation, le code C à fournir était quasiment le même qu'au TME3 aux vérifications près. Le patron C était, en revanche, un peu plus délicat à écrire.

Pour la question 3, il fallait étendre la grammaire :

```
expression |= vecteur
vecteur = element vecteur {
    expression *
}
```

puis indiquer le schéma de compilation. Celui pouvait utiliser les fonctions précédentes `makeVector` et `vectorSet` ou allouer plus efficacement le vecteur et le remplir. Encore une fois, donner le schéma de compilation était plus payant que de s'embarquer dans des output.append() filandreux.