Devoir de programmation Java
À rendre le 10/4/2006

Le contexte

Le but de ce devoir est de manipuler un système dit ≪de réécriture≫ afin de calculer la forme normale d'un terme fonctionnel. Une explication et une motivation en sont données à la fin de ce texte.

Le sujet se décompose en trois parties constituées de modules. Chaque module devra pouvoir être testé indépendamment de l'ensemble. En d'autres termes, tout ce qui est construit ou calculé dans un module pourra être affiché si on le souhaite.

Le terme de *module* utilisé dans le texte signifie méthode ou classe au sens large.

Partie I :

Manipulations d'un terme fonctionnel

Un terme fonctionnel est composé de symboles de fonction (d'arité positive ou nulle) et de variables. On appelle signature, notée Σ, l'ensemble des symboles de fonction utilisés et on note `V` (resp. `V(t)`) l'ensemble des variables (resp. des variables d'un terme `t`).

Par convention, les variables d'un terme sont désignées par des lettres majuscules.

Exemple : Le terme `f(h(c), g(X, h(Y)), a)` est composé des symboles `f, g, h, a` et `c` d'arités respectives 3, 2, 1 0, 0 et des variables `X` et `Y`. `h(c), g(X, h(Y))` et `a` en sont les sous-termes et sont eux-mêmes des termes.

On peut voir un terme fonctionnel comme un arbre, tel que la racine de cet arbre porte pour information le symbole de tête du terme et tel que les sous-arbres correspondent aux sous-termes.

Choisir une représentation sous forme d'arbre général pour un terme fonctionnel.

Module I :

Écrire un module qui lit un terme fonctionnel et qui construit la structure arborescente qui lui correspond. Ce module se chargera d'inférer l'arité des symboles fonctionnels qui interviennent dans le terme et stockera le résultat dans une liste associée au terme.

Remarque : Vous pourrez supposer que le terme lu est correctement écrit. On pourrait en faire une analyse mais ce n'est pas l'objet de ce travail.

Exemple : Pour le terme `t = f(h(c), g(X, h(Y)), a)`, la liste construite associée à `t` est la liste [(`f`, 3):: (`h`, 1)::(`c`, 0)::(`g`, 2)::(`a`, 0)]

Remarque : L'ordre des chaînons dans la liste n'est pas important.

Module II :

Écrire un module qui affiche la représentation arborescente d'un terme fonctionnel, sous une forme graphique agréable (par exemple sous la forme d'un arbre indenté).

Exemple : Le terme `f(h(c), g(X, h(Y)), a)` sera affiché comme suit :

```
f
  h
    c
  g
    X
    h
      Y
  a
```

Désormais, la suite manipule exclusivement la représentation arborescente du terme fonctionnel.

L'usage veut que l'on attribue un numéro à chaque nœud d'un arbre correspondant à un terme `t`. Ces numéros sont attribués de la façon suivante :

- la racine de l'arbre a le numéro 1 ;
- soit un nœud de numéro `i` correspondant à un symbole de fonction `f` d'arité `k` et dont l'arbre à partir de ce nœud correspond au terme $f(t_1, t_2, \ldots, t_k)$, alors on attribue à t_1 le numéro `i1`, à t_2 le numéro `i2`, ..., à t_k le numéro `ik`.

L'ensemble des numéros attribués à un terme `t` depuis sa racine est l'ensemble des occurrences de ce terme, noté `Occ(t)`. On suppose pour des raisons de simplicité, qu'aucun symbole de fonction n'a une arité supérieure à 9.

Exemple : soit `t` le terme `f(h(c), g(X, h(Y)), a)`,

```
Occ(t) = {1, 11, 111, 12, 121, 122, 1221, 13}
```

Module III :

Écrire un module qui détermine l'ensemble des occurrences d'un terme.

On note `t(i)` le sous-terme de `t` à l'occurrence `i`.
Exemple : soit `t` le terme `f(h(c), g(X, h(Y)), a)` ; alors `t(12) = g(X, h(Y))`, `t(13) = a`, `t(1221) = Y`.

Module IV :

Écrire un module qui, étant donnés un terme `t` et un entier `i`, affiche le sous-terme de `t` à l'occurrence `i`. Vous signalerez une erreur si `i` n'est pas dans `Occ(t)`.

Les substitutions

On définit une substitution comme une application qui associe un terme à une variable. Appliquer une substitution à un terme consiste à remplacer une ou plusieurs de ses variables par des

termes précisés dans la définition de la substitution.

Exemple : Soit $\sigma = \{\texttt{X} \leftarrow \texttt{b, Y} \leftarrow \texttt{h(Z)}\}$ et soit le terme `t = f(h(c), g(X, h(Y)), a)`. Si on applique σ à `t`, on obtient le terme noté σ`t` égal à `f(h(c), g(b, h(h(Z))), a)`.
On appelle domaine de σ les variables qui apparaissent dans sa définition. Ici, $\texttt{Dom}(\sigma) = \{\texttt{X, Y}\}$

Substitution bien définie : σ est bien définie par rapport à un terme `t` si aucun des cas suivants ne se produit :

1. son domaine comporte une variable de `t` plus d'une fois (cela voudrait dire qu'elle remplace une même variable par deux termes différents) ;
2. son domaine ne comporte aucune variable de `t`. Dans ce cas, elle n'a aucun effet sur `t`.

Si dans la définition de σ apparaissent des variables du terme `t`, σ n'est pas applicable à `t` telle quelle. Il faut alors renommer ces variables dans σ avant d'effectuer la substitution. Par exemple, si $\sigma = \{\texttt{Y} \leftarrow \texttt{h(X)}\}$, on ne peut pas l'appliquer au terme `f(h(c), g(X, h(Y)), a)` puisque `X` qui apparaît dans la définition de σ est aussi dans `V(t)`. On renomme la variable `X` dans σ. σ devient $\{\texttt{Y} \leftarrow \texttt{h(Z)}\}$ et σ`t` vaut `f(h(c), g(X, h(h(Z))), a)`

Module V :

Écrire un module qui, étant donnés un terme `t` et une substitution σ définie sur un sous-ensemble des variables de `t`, applique σ à `t` et construit un nouveau terme. Si σ n'est pas bien définie par rapport à `t`, signaler pourquoi et ne pas effectuer la substitution.

Partie II :

Manipulations d'un couple de termes fonctionnels vu comme une règle de réécriture

On souhaite à présent introduire le concept de réécriture d'un terme par une règle de réécriture. Mais tout d'abord, qu'est-ce qu'une règle de réécriture ? C'est tout simplement une équation orientée.

Une équation $\texttt{t}_1 = \texttt{t}_2$ est un couple de termes fonctionnels traduisant l'égalité de ces deux termes dans une certaine théorie axiomatique. D'un point de vue pratique, cette équation est utilisée dans un seul sens du terme *le plus gros* vers le terme *le plus petit* donnant lieu à une règle de réécriture notée $\texttt{t}_1 \rightarrow \texttt{t}_2$[1]

Enfin, on définit la réécriture d'un terme par une règle de réécriture comme suit : soient $\texttt{g} \rightarrow \texttt{d}$ une règle de réécriture et `t` un terme. `t` se réécrit par $\texttt{g} \rightarrow \texttt{d}$ s'il existe une occurrence `u` dans `t` et s'il existe une substitution σ telle que `t(u)` $= \sigma$`g`.
`t` se réécrit alors en `t'` où à l'occurrence `u`, on a remplacé σ`g` par σ`d`. On note `t` $\Rightarrow$ `t'`

Exemple : Soit la règle `h(X)` $\rightarrow$ `X` et soit `t = f(h(b), Z)`, alors `t` $\Rightarrow$ `f(b, Z)` à l'occurrence `11` avec $\sigma = \{\texttt{X} \leftarrow \texttt{b}\}$

Remarque : Un terme peut être réécrit par une même règle à différentes occurrences. Par exemple, soient `f(X, X)` $\rightarrow$ `X` et `t = f(f(a, a), f(a, a))` alors

1. `t` $\Rightarrow$ `f(a, f(a, a))` à l'occurrence 11 avec $\sigma = \{\texttt{X} \leftarrow \texttt{a}\}$,
2. `t` $\Rightarrow$ `f(f(a, a), a)` à l'occurrence 12 avec $\sigma = \{\texttt{X} \leftarrow \texttt{a}\}$, et

[1] Ceci sous-entend que les termes t_1 et t_2 sont comparables avec un ordre. Vous vous contenterez de supposer qu'un tel ordre existe et que toute équation peut être orientée en règle de réécriture.

3. `t` $\Rightarrow$ `f(a, a)` à l'occurrence 1 avec $\sigma = \{$`X` $\leftarrow$ `f(a, a)`$\}$

Module VI :

Écrire un module qui prend en arguments une règle de réécriture et un terme et qui réécrit ce terme par la règle à toutes les occurrences où c'est possible. Ce module fournira les termes résultant des différentes possibilités de réécrire le terme par la règle ou l'ensemble vide le cas échéant.

Partie III :

Réduction d'un terme

On arrive enfin à l'objectif de ce travail. Il consiste à calculer la forme normale d'un terme `t` par rapport à un ensemble de règles de réécriture `R` (dit aussi système de réécriture). En d'autres termes, on calcule le *plus petit terme* `t'` qui soit égal à `t` dans la théorie axiomatisée par `R`.[2]

Module VII : Écrire un module qui prend en argument un système de réécriture `R` et un terme `t` et qui retourne la forme normale de `t` dans `R`. Ce module doit offrir la possiblité d'afficher les étapes de réduction de `t` par les règles de `R`. Si `t` se réduit en sa forme normale de plusieurs façons, vous devrez les montrer toutes si on le souhaite.

Conclusion et motivations

Tout ce travail s'inscrit dans une problématique de procédure de décision pour le *problème du mot*. Ce problème s'énonce comme suit :

$\ll$On voudrait, étant donnée une théorie `R` formulée par un système de réécriture ayant une certaine propriété de complétude[3], pouvoir décider d'une égalité de termes $\mathtt{t}_1 = \mathtt{t}_2$. $\gg$

Pour tester si l'égalité est vraie ou non dans `R`, il suffit de prendre pour point de départ chacun des termes $\mathtt{t}_1$ et $\mathtt{t}_2$ et de les transformer par les règles de `R`. Si on obtient le même terme, alors l'égalité de départ est vraie dans `R`, sinon elle est fausse. On a donc une procédure de décision ! Mais ceci est une autre histoire !

Travail demandé :

On demande de réaliser chaque partie explicitée plus haut. On demande aussi de rédiger un document de 4 pages au maximum comportant une analyse claire et détaillée du travail effectué, ainsi qu'un mode d'emploi de l'outil développé, en faisant ressortir la participation de chacun. Ce rapport devra aussi expliquer comment sont articulés vos programmes et de quelle manière on peut tester les différentes modules demandés.

Les fichiers java de vos programmes devront être suffisamment commentés et comporter obligatoirement les jeux d'essai correspondant aux exemples donnés en annexe.

Quoi rendre et sous quelle forme ? le rapport sous format pdf et les programmes java par message envoyé à votre chargé de TD avec comme sujet DEVOIRALGO4. Les adresses sont les

[2] Vous serez, on en est sûr, convaincus de l'utilité de la notion de "plus petit". Elle est utile dans bien des domaines : théorie du point fixe, inférence de type, ...

[3] Pour ne pas trop vous embrouiller, on se contente de préciser que cette propriété est essentielle sans vous l'énoncer.

suivantes : jdl@univ-orleans.fr jean-michel.Couvreur@univ-orleans.fr dao@univ-orleans.fr Wadoud.Bousdira@univ-orleans.fr

Le projet est à réaliser par groupes de deux étudiants. Si vous n'arrivez pas à programmer la totalité de ce qui est demandé, surtout pas de découragement ; un travail personnel inachevé sera largement plus apprécié qu'un travail fini mais copie conforme de celui des voisins !

Annexe

Jeu d'essai n$^{\text{ø}}$ 1 :

$$\mathtt{R} = \begin{cases} \mathtt{f(g(X,\ X))} \to \mathtt{f(X)} \\ \mathtt{g(a,\ a)} \to \mathtt{a} \\ \mathtt{f(h(b))} \to \mathtt{h(b)} \\ \mathtt{h(X)} \to \mathtt{X} \\ \mathtt{f(b)} \to \mathtt{h(b)} \end{cases}$$

Calculer la forme normale de $\mathtt{t}_1 = \mathtt{f(g(a,\ a))}$, $\mathtt{t}_2 = \mathtt{f(h(b))}$ et $\mathtt{t}_3 = \mathtt{f(h(g(a,\ a)))}$

Jeu d'essai n$^{\text{ø}}$ 2 :

$$\mathtt{R} = \begin{cases} \mathtt{f(X,\ X)} \to \mathtt{a} \\ \mathtt{f(X,\ g(X))} \to \mathtt{b} \\ \mathtt{g(c)} \to \mathtt{c} \\ \mathtt{b} \to \mathtt{a} \end{cases}$$

Calculer la forme normale de $\mathtt{t} = \mathtt{f(g(c),\ g(c))}$

Jeu d'essai n$^{\text{ø}}$ 3 :

$$\mathtt{R} = \begin{cases} \mathtt{add(X,\ e)} \to \mathtt{X} \\ \mathtt{add(add(X,\ Y),\ Z)} \to \mathtt{add(X,\ add(Y,\ Z))} \\ \mathtt{add(e,\ X)} \to \mathtt{X} \end{cases}$$

Calculer la forme normale de $\mathtt{t} = \mathtt{add(add(X,\ e),\ add(e,\ Y))}$

Jeu d'essai n$^{\text{ø}}$ 4 :

$$\mathrm{E} = \begin{cases} \mathtt{f(f(f(f(f(X)))))} \to \mathtt{X} \\ \mathtt{f(f(f(X)))} \to \mathtt{X} \\ \mathtt{f(X)} \to \mathtt{X} \\ \mathtt{g(a+0)} \to \mathtt{g(a)} \end{cases}$$

Calculer la forme normale de $\mathtt{t} = \mathtt{f(f(f(f(g(f(a)+0)))))}$

Jeu d'essai n$^{\text{ø}}$ 5 :

$$\mathtt{R} = \begin{cases} \mathtt{g(f(g(f(X))))} \to \mathtt{g(f(X))} \\ \mathtt{g(f(X))} \to \mathtt{g(X)} \\ \mathtt{g(f(g(X)))} \to \mathtt{g(X)} \\ \mathtt{g(g(X))} \to \mathtt{g(X)} \end{cases}$$

Calculer la forme normale de $\mathtt{t} = \mathtt{g(g(f(g(f(b)))))}$