Projet individuel d'algorithmique-programmation IPF : groupe 2 (Enseignant : C. Dubois)

10 mars 2018

1 Informations générales

1.1 Travail à rendre

Le projet est à réaliser en OCaml **individuellement**. Il sera accompagné d'un **dossier** contenant impérativement la description des choix faits, la description des types et des fonctions. Pour chaque fonction, on donnera impérativement l'interface complète (dans le code en commentaire et dans le rapport pour les fonctions présentées).

Même si le sujet est décomposé en questions, il est possible qu'une question se résolve par l'écriture d'une ou plusieurs fonctions intermédiaires. Celles-ci doivent comporter une interface également.

Le dossier fournira également des cas de tests accompagnés des résultats attendus et retournés.

Sur le site du cours figure un petit document sur ce que l'on attend dans un rapport. Consultez-le !

Attention, le code doit être purement fonctionnel : pas de boucle, pas d'affectation, pas de tableau. La seule tolérance concerne les affichages (voir document en ligne).

1.2 Calendrier et procédure de remise

Le projet est à rendre le **mardi 3 avril 2018** minuit au plus tard sur le site de dépots `exam.ensiie.fr`. Cliquez sur **IPF_S2_2018**. Vous y déposerez une archive contenant votre rapport au format pdf (impérativement) et le code de votre projet.

2 Enoncé du projet : Termes et réécriture

Le but du projet est d'implémenter un système de réécriture i.e. un système permettant d'appliquer de façon automatique des régles de réécriture. Le problème est cependant simplifié (en particulier on restreint les termes, les règles possibles et enfin la stratégie d'application des règles).

On définit un terme de la façon suivante (voir cours de logique) : un terme est soit
— une variable,
— un symbole d'arité 0 (une constante),
— un terme de la forme $g(t)$ où g est un symbole de fonction d'arité 1 et t un terme,
— ou un terme de la forme $f(t_1, t_2)$ où f est un symbole de fonction d'arité 2 et t_1 et t_2 deux termes.
On rappelle que l'arité est le nombre d'arguments.

En OCaml, on définira le type des termes de la façon suivante :

```
type terme = Var of string | Const of string |
           | Unop of string * terme | Binop of string * terme * terme;;
```

Par exemple si f est un symbole de fonction autorisé d'arité 2, g un symbole de fonction d'arité 1, a une constante et x une variable alors le terme f(g(a),x) est représenté en Ocaml par `Binop ("f", Unop ("g", Const "a"), Var "x")`.

On définit une *signature* comme la donnée des symboles de fonction autorisés avec leur arité. Une signature est donc une liste de type `(string*int) list`.

Dans la suite, pour tester vos fonctions, vous utiliserez **au moins** les deux systèmes de réécriture suivants :

1. Arithmétique : la signature est composée de la constante zero, du symbole de fonction succ d'arité 1, du symbole de fonction plus d'arité 2 (elle est donc représentée par la liste `[(''zero'', 0); (''succ'', 1); (''plus'', 2)]`) avec les règles de réécriture suivantes :
   $plus(x, zero) \to x$
   $plus(zero, x) \to x$
   $plus(succ(x), y) \to plus(x, succ(y))$.
2. Piles : la signature est composée des constantes zero et vide (pour la pile vide), la fonction succ d'arité 1, le symbole de fonction pop d'arité 1 (désigne le dépilement d'une pile), le symbole de fonction push d'arité 2 (attention son premier argument doit être un terme construit avec zero et succ), le symbole de fonction alternate d'arité 2 (dont les deux arguments sont des iles). Les règles de réécriture sont le suivantes :
   $top(push(x, y)) \to x$
   $pop(push(x, y)) \to y$
   $alternate(vide, z) \to z$
   $alternate(push(x, y), z) \to push(x, alternate(y, z))$

1. Écrire une fonction qui teste si un terme est bien formé par rapport à une signature donnée.
2. On appelle *motif* un terme bien formé linéaire c'est-à-dire un terme qui, s'il contient des variables, ne contient qu'une seule occurrence de chaque variable. Par exemple $f(x, g(y))$ est un motif linéaire alors que $f(x, x)$ n'est pas linéaire. Écrire une fonction qui teste si un terme est un motif.
3. On veut vérifier si un terme t sans variable est un terme de la forme précisée par un motif m. On dit alors que le motif m *filtre* le terme t.
   Par exemple si la signature contient le symbole de fonction f d'arité 2, les symboles de fonction a et b d'arité 0, on peut par exemple construire le motif $f(x, a)$ (avec x une variable). Le terme sans variable $f(f(a, b), a)$ est une instance du motif $f(x, a)$. En effet on retrouve le terme t en remplaçant dans m la variable x par $f(a, b)$. En revanche ce n'est pas une instance du motif $f(a, x)$.
   Écrire un programme qui, pour un terme sans variable t et un motif m, calcule la substitution permettant d'identifier m et t si cette identification est possible. On représentera une substitution par une liste de couples de la forme (nom de variable, terme). Si m ne filtre pas t, alors la fonction échouera.
4. Une *règle de réécriture* s'écrit $tg \to td$ où tg est un terme linéaire avec variables et td un terme avec ou sans variables. Elle signifie que tout terme de la forme de tg se réécrit (se simplifie) en td. Par exemple avec une signature où $plus$ est d'arité 2 et $zero$ d'arité 0, on peut écrire la règle de réécriture $plus(x, zero) \to x$. Avec cette règle on pourra simplifier le terme $plus(plus(zero, zero), zero)$ en $plus(zero, zero)$. Si on applique une deuxième fois cette règle on pourra simplifier le terme
   $plus(zero, zero)$ en $zero$.
   On impose que les variables du terme td soient des variables qui apparaissaient déjà dans tg (la règle n'introduit pas de nouvelle variable).
   On appelle système de réécriture une liste de règles de réécriture.
   Écrire une fonction qui vérifie qu'un système de réécriture est bien formé (c'est-à-dire que les termes des règles respectent la signature, qu'ils sont linéaires et que le membre droit d'une règle n'utilise que des variables qui sont déjà dans le membre gauche de la même règle).
5. On dit que la règle $tg \to td$ s'applique à un terme t si tg filtre t. Soit s la substitution résultat. Alors, le résultat de l'application de la règle est $s.td$, i.e. le terme résultant de l'application de la substitution s sur td.

Écrire un programme qui applique les règles d'un système de réécriture sur un terme jusqu'à ce qu'aucune des règles de la liste ne s'applique plus. Le résultat est alors le terme final obtenu.

6. Reprendre le programme précédent de manière à ce qu'il imprime la suite des termes obtenus au cours de la réécriture ainsi que le numéro de la règle qui a permis de l'obtenir. Par exemple avec le système de réécriture composé des règles $plus(x, zero) \rightarrow x$, $plus(zero, x) \rightarrow x$ et $plus(succ(x), y) \rightarrow plus(x, succ(y))$, le programme devra afficher pour le terme initial $plus(zero, plus(succ(zero), succ(zero)))$:
   par r2 : $plus(succ(zero), succ(zero))$
   par r3 : $plus(zero, succ(succ(zero)))$
   par r1 : $succ(succ(zero))$
7. Application à la démonstration : pour démontrer que deux termes sont égaux on peut appliquer la réécriture sur les deux termes et vérifier que les deux termes obtenus après simplification sont identiques (syntaxiquement). Ecrire une fonction demo qui prend en paramètre deux termes et un système de règles de réécriture et retourne true si les deux termes sont égaux, false sinon.
8. Amélioration : **Partie à aborder uniquement si ce qui précède fonctionne correctement**

   Avec le système de règles précédent, le terme $T = plus(plus(zero, succ(zero)), succ(zero))$ ne peut pas se simplifier. On va donc changer de stratégie et autoriser que la réécriture se fasse aussi sur un sous-terme du terme à réécrire (c'est-à-dire si ce sous-terme est filtré par un membre gauche de règle). Par exemple dans le terme T, le sous-terme $plus(zero, succ(zero))$ filtre le membre gauche de la règle $r2$: il se réécrit en $succ(zero)$. Tout le terme se réécrit donc $plus($**succ(zero)**$, succ(zero))$ qui peut ensuite se réécrire en $plus(zero, succ(succ(zero)))$ qui se réécrit finalement en $succ(succ(zero))$.

   Mettre en place cette nouvelle stratégie de réécriture : pour chaque règle on commencera par vérifier si la règle s'applique au terme complet, si c'est le cas, on l'appliquera. sinon on vérifiera si elle peut s'appliquer sur un des sous-termes du terme à réécrire et on l'appliquera.