Projet de Programmation 2013-2014
Partie Ocaml - Programmation Fonctionnelle
Travail individuel

Encadrement : Pierre Bisquert, Armelle Bonenfant, Sergei Soloviev et Guillaume Verdier

13 novembre 2013

1 Introduction

Dans ce projet, il s'agit de décrire des expressions arithmétiques entières et d'effectuer des opérations sur ces expressions (calculs, simplifications, transformations...). Toute la programmation doit se faire en fonctionnel "pur" en un seul fichier. La lisibilité du code est primordiale, l'efficacité est appréciée.

2 Sujet

2.1 Représentation des données

Une expression est définie par induction. Une expression peut être :
- un entier,
- une variable (caractère),
- l'addition de deux expressions,
- la soustraction de deux expressions,
- le produit de deux expressions,
- la division entière de deux expressions,
- l'opposé d'une expression.

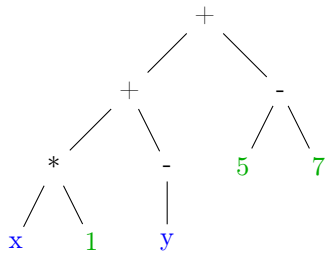
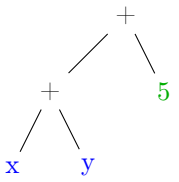


FIGURE 1 – Exemple d'expression

Définissez le(s) type(s) nécessaire(s) à la représentation d'une expression et illustrez-le(s) à l'aide d'exemples de valeurs de ce(s) type(s).

2.2 Affichage d'une expression

Seule exception au fonctionnel "pur", l'affichage est utile pour rendre plus lisible les expressions un peu complexes.



(a)

((x+y)+5)

(b)

FIGURE 2 – Affichage

Ecrivez la (les) fonction(s) nécessaire(s) pour afficher une expression telle que décrite par la figure 2a comme la figure 2b.

2.3 Evaluation d'expression

Un environnement est l'ensemble des liaisons entre variables et valeurs dans une expression. Pour un environnement donné, l'évaluation d'une expression consiste à calculer la valeur entière de l'expression.

Exemple : pour l'expression de la figure 1, $\{x, y\}$ est l'ensemble des variables et

x	y
2	4

est un environnement possible. L'évaluation de l'expression avec cet environnement a pour valeur -4.

Choisissez le(s) type(s) nécessaire(s) pour représenter un environnement (lien entre une variable et sa valeur) [1].

Ecrivez la (les) fonction(s) nécessaire(s) pour évaluer une expression pour un environnement donné.

2.4 Simplification

Afin de réduire la taille des expressions, il existe quelques simplifications simples : éléments neutres (0 pour le + et 1 pour le *), élément absorbant (0 pour le *) et doubles négations unaires. Il est possible de simplifier aussi les neutres et absorbant pour le - et le / . La simplification peut aussi détecter, pour certains cas, les divisions par zéro et générer une erreur.

Exemple :

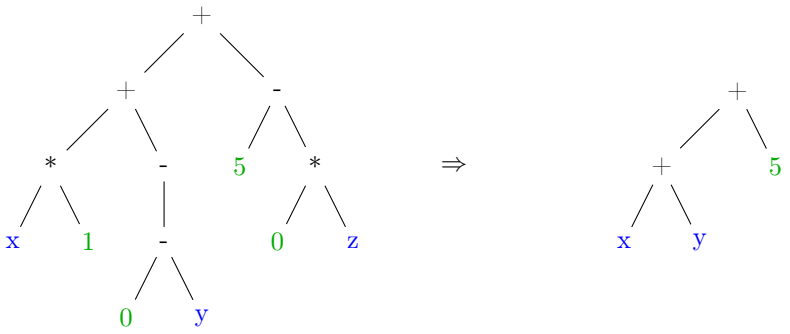


FIGURE 3 – Exemple de simplification

Ecrivez la (les) fonction(s) nécessaire(s) pour construire une expression simplifiée.

1. Selon le choix fait ici, il y a des conséquences sur la partie 2.5

2.5 Environnements

2.5.1 Maximum

Pour une variable, il existe une infinité de valeurs possible, ainsi pour une expression contenant n variables, il existe une infinité d'environnements possibles (∞^n). On va donc se limiter ici aux valeurs 0 ou 1 pour chaque variable afin de ramener le nombre d'environnements possibles à 2^n. On cherche à déterminer le maximum des évaluations d'une expression suivant ses environnements (sur $\{0, 1\}$). Pour cela, il faut donc énumérer les environnements possibles, appliquer l'évaluation (définie en 2.3) pour chaque environnement et retenir celui qui maximise l'expression.

Exemple :

FIGURE 4 – Expressions à maximiser

L'expression de la figure 4a a deux environnements possibles : $\begin{array}{|c|}\hline \text{x} \\ \hline 0 \\ \hline \end{array}$ et $\begin{array}{|c|}\hline \text{x} \\ \hline 1 \\ \hline \end{array}$, deux valeurs d'évaluation possibles 3 et 4. Le maximum est donc 4 et l'environnement maximal est $\begin{array}{|c|}\hline \text{x} \\ \hline 1 \\ \hline \end{array}$.

L'expression de la figure 4b a quatre environnements possibles : $\begin{array}{|c|c|}\hline \text{x} & \text{y} \\ \hline 0 & 0 \\ \hline \end{array}$, $\begin{array}{|c|c|}\hline \text{x} & \text{y} \\ \hline 0 & 1 \\ \hline \end{array}$, $\begin{array}{|c|c|}\hline \text{x} & \text{y} \\ \hline 1 & 0 \\ \hline \end{array}$ et $\begin{array}{|c|c|}\hline \text{x} & \text{y} \\ \hline 1 & 1 \\ \hline \end{array}$, trois valeurs d'évaluation possibles $0, 3$ et 4. Le maximum est donc 4 et l'environnement maximal est $\begin{array}{|c|c|}\hline \text{x} & \text{y} \\ \hline 1 & 1 \\ \hline \end{array}$.

Ecrivez la (les) fonction(s) nécessaire(s) qui calcule(nt) l'environnement qui maximise une expression et ce maximum.

2.5.2 Nullité

Toujours dans le cadre d'environnement limité aux valeurs 0 et 1, on cherche à savoir s'il existe un 0 pour une expression, c'est-à-dire s'il existe un environnement pour lequel l'expression s'évalue à 0.

Ecrivez la (les) fonction(s) nécessaire(s) pour savoir s'il existe un 0 pour une expression (sans générer tous les environnements).

2.6 Ordre supérieur : application en profondeur

On désire pouvoir appliquer des transformations sur les expressions. Par exemple : renommer les variables, remplacer des variables par des expressions, échanger un "+" par un "*", obtenir le miroir d'une expression...

Pour cela, il faut écrire une fonction `applique` qui va permettre de construire une nouvelle expression en appliquant récursivement une transformation donnée à une expression.

Exemple : pour la transformation "une variable est remplacée par sa valeur en code ASCII", le résultat de `applique` est représentée par la figure 5.

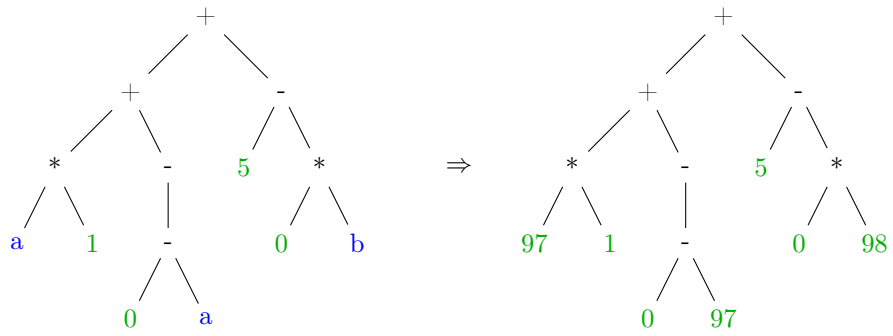
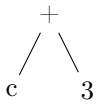


FIGURE 5 – Exemple pour applique "ASCII"

Autre exemple : pour la transformation "si la variable vaut x remplacer par l'expression exp". Le résultat de `applique` pour cette transformation de l'arbre est figure 6 pour x qui vaut "a" et `expression` qui vaut



.

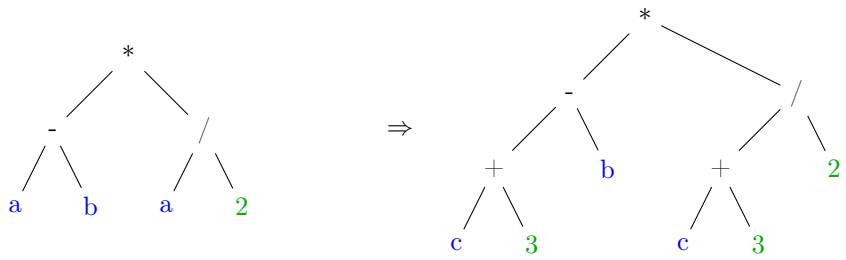


FIGURE 6 – Exemple pour applique "a" devient "c+3"

Dernier exemple : la transformation "miroir" qui permute les expressions d'opérateurs binaires. Le résultat d'applique sur une telle transformation est figure 7.

FIGURE 7 – Exemple pour applique "miroir"

Ecrivez la fonction `applique` demandée. Ecrivez les fonctions `var2code`, `x2exp` et `miroir` qui sont les transformations utilisées dans les exemples.

3 Organisation

3.1 Organisation du travail

Vous devrez déposer un unique fichier Ocaml (.ml) sous moodle à la date limite correspondant à votre groupe. Pensez à commenter votre code afin qu'il soit lisible par le correcteur. Le programme doit compiler sous peine d'une note nulle.

Lors de la validation, vous devrez :

- présenter un programme qui fonctionne,
- pouvoir modifier votre programme à la demande,
- pouvoir justifier de vos choix et de l'efficacité de vos programmes,
- pouvoir tester nos formules

3.2 Organisation dans le temps

- Distribution du sujet le 15 novembre
- Séance de questions/réponses la semaine du 25 novembre en groupe de TP
- Séance de validation la semaine du 9 décembre en groupe de TP

3.3 Organisation des ressources

Vous devez utiliser les ressources qui sont à votre disposition pour vous permettre de mener à bien ce projet (par ordre de priorité) : supports de cours et site Ocaml, forum, messagerie du responsable "projet" de votre groupe, de vos TP, de cours.

4 Evaluation

La note du module est composée : d'une note sur le travail Ocaml individuelle, d'une note sur le travail Java (fait en binôme). La note de chaque partie prend en compte non seulement la réalisation des programmes demandés, mais aussi la qualité de la programmation (commentaires, ocamldoc ou javadoc, efficacité des calculs...). Il vous sera demandé un rapport de synthèse sur le travail Java.

5 PLAGIAT/TRAVAIL COLLECTIF

Toute fraude, même partielle sera sanctionnée par des notes de groupe ET individuelle nulles (et éventuellement par un recourt au conseil de discipline). En cas de travail collectif entre différents groupes de projet, les points des portions collectives pourront être partagés ou attribués entièrement SI le travail est réellement collectif et uniquement si cela a été signalé aux encadrants du projet. Rappel : il est illégal d'utiliser des sources sans en citer les auteurs.