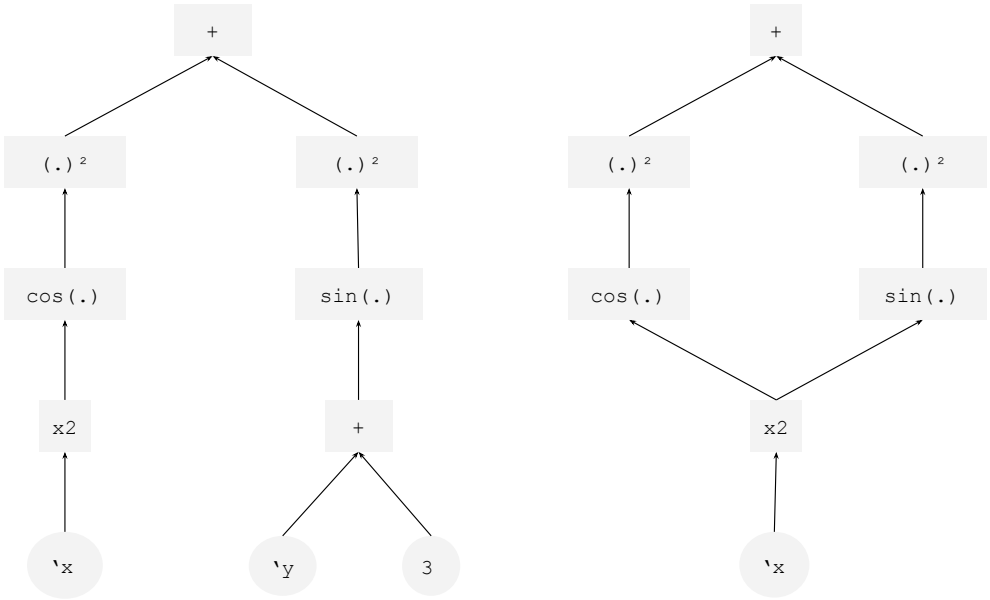
INFO0054 - Programmation fonctionnelle

Calcul symbolique

Jean-Michel Begon

2019-2020



(a) Un arbre de calculs correspondant à $f(x, y) = (cos(2x))^2 + (sin(y + 3))^2$

(b) Un graphe de calculs correspond à $f(x) = (cos(2x))^2 + (sin(2x))^2$

Figure 1 – Arbre et graphe de calculs

On peut représenter un calcul sous la forme d'un arbre (Figure 1a) ou plus généralement d'un graphe dirigé acyclique (Figure 1b). Les variables et les constantes sont appelées feuilles du graphe, alors que les autres nœuds sont qualifiés d'internes.

De telles structures sont couramment utilisées dans divers domaines de l'intelligence artificielle, notamment en *deep learning* (une branche de l'apprentissage automatique) où ces graphes permettent de calculer des gradients nécessaires à l'optimisation de réseaux de neurones. En effet, de la connaissance du graphe, il est aisé de calculer les dérivées partielles via le théorème des fonctions composées (on parle de *backpropagation* en apprentissage automatique).

Dans ce projet, nous allons nous restreindre à la manipulation de tels graphes sans aborder la différenciation.

1 Manipulation de graphe

La manipulation de graphe comprend la création, l'affichage, la résolution (partielle) et la simplification des graphes de calcul.

1.1 Définitions

Tout d'abord, voici quelques définitions qui aideront à structurer le projet.

Un NODE est une fonction qui représente un nœud du graphe, à laquelle est associé un STORE (encapsulé dans sa fermeture) et qui prend en entrée un NODE_ARGUMENT.

Un STORE est soit

— un CST_STORE, qui est un nombre ;

— un VAR_STORE, qui est un symbole représentant une variable ;

— un FN_STORE, qui est un triplet.

Un FN_STORE est une liste qui contient dans l'ordre

1. SYMBOL, un symbole correspondant à la fonction que le nœud représente. Par exemple, le symbole du nœud au sommet de l'arbre de la figure 1a est `+`.
2. CHILDREN, une liste de NODEs correspondant aux nœuds fils. Par exemple, les fils du nœud d'addition de la branche droite à la figure 1a sont les nœuds feuilles correspondant à `y` et `3`.
3. SIMPLIFY-FN, une fonction de simplification (*cf.* section 1.3).

Un NODE_ARGUMENT est soit

— une opération de graphe `f`, c'est-à-dire une fonction qui prend en entrée un STORE, auquel cas `[[(node f)]] = [[(f store)]]`, où `store` est le STORE du NODE `node` (*cf.* section 1.3) ;

— une table d'assignation `t`, c'est-à-dire une liste de paires pointées dont le premier élément est le symbole d'une variable et le second un nombre, auquel cas `[[(node t)]]` renvoie un NODE représentant le graphe de calculs où toutes les occurrences de la variable ont été remplacées par la valeur (*cf.* 1.5).

Enfin, une FACTORY est une fonction Scheme qui représente une fonction du graphe de calculs (par exemple, la fonction cos ou l'addition) et qui prend en entrée le nombre d'arguments correspondant à la fonction mathématique qu'il représente (un pour la fonction cos, un nombre arbitraire pour l'addition). La FACTORY accepte des arguments d'une des trois natures suivantes :

— des nombres ;

— des symboles ;

— des NODE.

La FACTORY renvoie le graphe (sous la forme d'un NODE) correspondant à l'opération. Dans ce graphe, les nombres et les symboles sont des feuilles et donc des NODEs également. Le graphe renvoyé est le plus simple graphe possible (*cf.* section 1.4).

1.2 Création de graphe

La création d'un graphe de calculs se fait au travers des fonctions de type FACTORY.

Supposons qu'on dispose des fonctions `Add`, `Mult`, `Pow`, `Cos`, `Sin`, chacune étant une FACTORY permettant de créer un nœud représentant une addition, une multiplication, une puissance, un cosinus et un sinus, respectivement. On peut créer l'arbre et le graphe de calculs correspondant à la figure 1 de la manière suivante :

```
> (define tree (Add (Pow (Cos (Mult 'x 2)) 2)
                    (Pow (Sin (Add 'y 3)) 2)))

> (define 2x (Mult 'x 2))
> (define graph (Add (Pow (Cos 2x) 2)
                     (Pow (Sin 2x) 2)))
```

1.3 Affichage de graphe

Afin d'afficher le graphe, vous devrez définir un NODE_ARGUMENT `repr` qui est un opérateur de graphe. L'opérateur `repr` prend en entrée un STORE et renvoie la représentation du graphe de calculs suivante :

- une variable est représentée par son symbole ;
- une constante est représentée par sa valeur ;
- une fonction est représentée par une liste de symboles. Le premier est le symbole de la fonction (*cf.* FN_STORE), les suivants sont les représentations des arguments de la fonction.

Puisque `repr` est un NODE_ARGUMENT, on peut l'utiliser comme suit :

```
> (tree repr)
'(+ (expt (cos (* 2 x)) 2) (expt (sin (+ 3 y)) 2)

> (graph repr)
'(+ (expt (cos (* 2 x)) 2) (expt (sin (* 2 x)) 2))
```

Notons que l'affichage est tel que si on lie une valeur à chaque variable, la représentation du nœud peut être évaluée directement à l'aide la forme `eval` :

```
> (define x 2)
> (eval '(+ (expt (cos (* 2 x)) 2) (expt (sin (* 2 x)) 2)))
1.0
```

Ceci impose donc le symbole utilisé pour chaque FACTORY.

1.4 Simplification de graphe

Une SIMPLIFY-FN `sf` est une fonction qui correspond à une opération `f`. Si `f` prend en entrée `n` arguments, `sf` prend en entrée une *liste* de `n` NODEs. Elle renvoie soit

- un NODE, lorsque la simplification consiste à se débarrasser du nœud lié à `f`. Par exemple, on peut remplacer un nœud qui correspond à l'addition de constantes par la constante résultante ;
- une liste de NODEs simplifiés sinon.

En effet, lors de la création d'un nœud, une série de simplifications peuvent être appliquées.

Pour l'addition, on peut

— additionner toutes les constantes pour qu'il n'y ait plus qu'un seul terme constant ;

— supplanter le nœud s'il n'y a plus de variables dans son sous-graphe. On remplace alors le nœud par le terme constant ;

— supplanter le nœud si le terme constant est nul et qu'il n'y a qu'un seul autre terme. On remplace alors le nœud par le terme restant.

— faire disparaître le terme constant s'il est nul.

Pour la multiplication, on peut

— multiplier toutes les constantes pour qu'il n'y ait plus qu'un seul facteur constant ;

— supplanter le nœud si un des facteurs est nul. On remplace alors le nœud par un nœud nul.

— faire disparaître le facteur s'il vaut 1 en supplantant le nœud si nécessaire.

De manière similaire pour la puissance, on peut simplifier les expressions pour lesquelles un des arguments vaut 0 ou 1, en supplantant le nœud quand c'est possible.

Quelque exemples :

```
> ((Add 2 -2 0 'x) repr)
'x
> ((Mult 2 (Pow 'x 0) (Pow 1 'n)) repr)
2
> ((Mult 'x 2 (Add 1 3)) repr)
'(* 8 x)
```

Remarque : par souci de lisibilité, les termes constants seront toujours les premiers fils pour les nœuds d'addition et de multiplication. L'ordre des autres nœuds devra se calquer sur celui avant simplification.

1.5 Résolution de graphe

Lorsqu'on passe une table d'assignation à un graphe, un nouveau graphe est créé où toutes les occurrences des variables présentes dans la table sont remplacées par la valeur correspondante. Le graphe est alors simplifié au maximum :

```
> ((tree '((y . 2))) repr)
'(+ 0.9195357645382262 (expt (cos (* 2 x)) 2))

> ((tree '((x . 0) (y . 2))) repr)
1.9195357645382263
```

2 Consignes

Il vous est demandé d'implémenter les fonctions suivantes dans un fichier `graph.scm` :

— les FACTORY correspondants à l'addition (`Add`), la multiplication (`Mult`), la puissance (`Pow`) ainsi que les fonctions cosinus (`Cos`), sinus (`Sin`) et exponentielle (`Exp`) ;

— l'opérateur de graphe `repr` qui retourne la représentation Scheme du graphe de calculs. Veillez à bien respecter la grammaire définie à la section 1.3, ainsi que les autres consignes relatives à la représentation. Celle-ci constituera le mode d'évaluation principal.

Veillez à bien rendre les fonctions susmentionnées accessibles depuis un autre fichier (*cf.* la forme `provide`).

Suggestion : définissez une fonction `mk-factory` qui prend en entrée un symbole, une fonction Scheme et une fonction SIMPLIFY-FN et qui renvoie la FACTORY correspondante.

Le projet est à réaliser en *binôme* pour le mardi 28 avril 23h59. Le fichier `graph.scm` est à rendre dans une archive au format `tar.gz` sur la plateforme `https://submit.montefiore.ulg.ac.be/`. Il n'est pas nécessaire de soumettre un rapport.

Vous avez jusqu'au 27 mars 23h59 pour former les groupes sur la plateforme de soumission. Passé ce délai, les groupes seront gelés. Les étudiants n'appartenant à aucun groupe ne pourront soumettre et recevrons *de facto* une cote nulle. Il n'y aura aucun aménagement pour les étudiants seuls – je vous invite vivement à vous mettre en groupe. Enfin je vous rappelle qu'il est possible de voir quels étudiants n'ont pas encore de binôme grâce à la plateforme de soumission.

Toute fonction auxiliaire doit être spécifiée !

Merci d'adresser toutes questions relatives au projet directement à jm.begon@uliege.be

Bon travail