Université Paris Diderot
M1 Informatique

Programmation Fonctionnelle Avancée
Année 2015-2016

Projet

Un logiciel de calcul formel

`Sujet version 1 (18 novembre 2015)`

1 Introduction

Vous souhaitez aider votre petite sœur ou votre petit frère à passer l'épreuve de mathématiques du bac, et en particulier l'exercice d'étude de fonction. Mais traiter à la main ce genre d'exercice, c'est galère (c'est quoi déjà la dérivée de la tangente ? ou le cosinus d'une somme ?). Vous allez réaliser en OCaml un logiciel de calcul formel, capable de simplifier une expression algébrique, de la dériver, d'intégrer certaines formules simples, etc. Vous trouverez en annexe quelques exemples de sujets de bac que l'on souhaite pouvoir traiter à l'aide de notre programme. Vous pourrez vous inspirer des logiciels existants tels que Maple ou Mathematica (logiciels propriétaires) ou encore Maxima (logiciel libre, version graphique disponible sous le nom `wxmaxima`).

2 Expressions algebriques

Nous vous fournissons une interface `expr.mli` proposant le type suivant pour les expressions algébriques:

```
type ('nums,'ops) generic_expr =
  | Num of 'nums
  | Var of string
  | Op of 'ops * ('nums,'ops) generic_expr list

type basic_expr = (int,string) generic_expr
```

Par exemple `x+sqrt(3)` se représente `Op ("+",[Var "x";Op("sqrt";[Num 3]])` . Le fichier `parsexpr.ml` fournit des opérations de parsing, et en particulier une conversion `expr_of_string : string -> basic_expr`. La syntaxe acceptée est simple:

- Entiers naturels
- Identifiers seuls donnant des variables (à part les cas particuliers des deux constantes `pi` et `e` donnant des opérateurs à 0 arguments)
- Opérateurs infixes `+ * - / ^`. Nota : les successions de `+` sont regroupées en un unique constructeur `Op` à n arguments, idem pour `*`. Et l'opérateur `-` peut également être unaire.
- Opérateurs préfixes de la forme `op(arg1,...,argN)`.

Le module `Parsexpr` accepte tout nom d'opérateurs sans vérification particulière. Votre programme devra en particulier pouvoir traiter les opérateurs suivants : `sqrt exp log sin cos tan asin acos atan`. L'opérateur `log` correspond au logarithme népérien.

Notez que vous pouvez remanier ce type `basic_expr` selon vos besoins (à condition d'adapter `Parsexpr` en conséquence), ou bien proposer un autre type d'expression qui vous arrange, et une conversion de `basic_expr` vers ce nouveau type. Notez en particulier que dans un `basic_expr`, rien ne garantit qu'un opérateur ait le bon nombre d'arguments, ou bien que tous les opérateurs sont bien supportés. Une autre limitation du type `basic_expr` est l'usage de `int` pour représenter les constantes entières. Ceci n'est pas idéal pour mener des calculs formels exacts, les opérations usuelles sur `int` peuvent conduire silencieusement à des débordements, et mener à des égalités fausses en mathématiques. Par exemple en OCaml l'expression `2*max_int+2` est égale à 0, alors que `max_int` est strictement positif ! Vous pouvez chercher à réaliser des opérations "sûrs" sur `int`, qui échouent en cas de débordement, ou mieux encore passer à une autre présentation des nombres (en taille arbitraire). Une possibilité est d'utiliser le type `Num.num` dans la bibliothèque `nums` fournie avec OCaml. Vous pouvez aussi utiliser la bibliothèque externe `zarith` qui est plus moderne.

3 Commandes souhaitées

Voici une liste de commandes qu'on souhaite pouvoir réaliser sur nos expressions formelles.

- `eval(e)` : calcule une valeur approchée de l'expression `e` en utilisant des calculs sur les nombres flottants. L'expression `e` ne doit pas contenir de variables, sinon l'évaluation échoue (cf. la commande `subst` ci dessous).

- `subst(e,x,e')` : ici `x` doit être une variable, et alors dans l'expression `e`, chaque occurrence de `x` est remplacée par l'expression `e'`.

- `simpl(e)` : produit une version simplifiée (mais toujours exacte) de l'expression `e`. Il n'y a pas de définition précise ou unique de ce qu'on entend ici par forme simplifiée, à vous d'appliquer au mieux les règles de simplifications usuelles, par exemple `x-x=0`, `log(exp(x))=x`, `sqrt(8)=2*sqrt(2)`, `sin(pi/3)=sqrt(3)/2`, etc. Plus globalement on pourra s'efforcer de développer les expressions pour faire apparaître des termes qui s'annulent ou au contraire se regroupent.

- `solve(e,x)` : ici aussi `x` doit être une variable, on tente alors de donner la ou les expressions exactes de `x` qui rendent l'expression `e` nulle, s'il y en a. On pourra s'occuper d'abord du cas des polynômes de degré 1 ou 2, puis essayer également un certain nombre de manipulations ad-hoc (par exemple mettre tout au carré s'il y a un `sqrt` d'un côté).

- `derive(e,x)` : calcule la dérivée de `e` vis-à-vis de la variable `x`. On parle ici de dérivée formelle (ou exacte). Cette commande doit pouvoir fonctionner sur toutes les expressions constituées à partir des opérateurs listés précédemment.

- `integ(e,x,a,b)` : cacule l'intégrale de l'expression `e` lorsque la variable `x` va des expressions `a` à `b`. On cherche de nouveau une réponse exacte. Cette fois-ci il n'y a pas de méthode générale comme pour la dérivée, on vous demande de savoir reconnaître et traiter un certain nombre de cas classiques, tels que ceux qu'on retrouve dans les formulaires de type Bac.

- `plot(e,x)` : dessine la courbe correspondant à l'expression `e` vue comme fonction à une variable `x`. Cette commande échoue si `e` contient d'autres variables que `x`. Par défaut, cet affichage se fait dans la zone $[-5..5; -5..5]$, mais on peut proposer des variantes comme `plot(e,x,a,b,c,d)` pour laisser à l'utilisateur le choix des bornes $[a..b; c..d]$.

La description de ces commandes (en particulier `simpl`) est volontairement succinte, et de plus certaines commandes comme `solve` ou `integ` n'ont pas forcément de solution générale. A vous de traiter le plus de situations possibles, en utilisant des techniques de niveau Bac à Bac+2.

Autres idées possibles de travail à réaliser :

- Signaler les contraintes rencontrées lors des manipulations, p.ex. dire que l'on a supposé $x > 0$ avant de simplifier x/x en 1.
- Un mode verbeux qui explique à l'utilisateur quelles règles de calculs ont été utilisées (comme sur une vraie copie de bac où on doit justifier le cheminement au lieu de juste donner le résultat final).
- Réalisation d'un tableau de variation et/ou de signe pour une fonction.
- Calculs de limites.

4 Interfaces

Plusieurs choix sont possibes pour l'interface utilisateur de votre programme. Là encore, la qualité et la difficulté du travail réalisé influera sur la notation.

- Par exemple, vous pouvez simplement réaliser un programme en ligne de commande, proposant une boucle interactive: l'utilisateur tape une ligne d'expression algébrique commençant par une commande, et votre programme affiche une version simplifiée de cette expression, dans la syntaxe d'origine. Attention à ce que cet affichage soit correct tout en étant aussi léger que possible (ni trop de parenthèses, ni trop peu).
- Lorsque qu'un tracé de fonction est demandé (commande `plot`), vous pouvez par exemple utiliser le module `Graphics` fourni avec OCaml, ou bien chercher à fabriquer une image (via la bibliothèque `camlimage`).
- Une autre idée d'interface est de proposer un joli rendu des formules. Vous pouvez par exemple chercher à produire un fichier de sortie Latex, ou bien du MathML.
- Il est aussi possible de combiner l'idée d'une boucle interactive et celle de MathML dans un programme graphique, afin de se rapprocher de programmes réalistes comme Maple, Mathematica ou WxMaxima. Pour cela, vous pouvez par exemple utiliser les bibliothèques `lablgtk2` et `lablgtkmathview`, une démo minimaliste de leur usage est fournie dans l'archive du projet, cf. le sous-répertoire `mathview-demo`.
- On peut enfin chercher à réaliser une application embarquée dans un navigateur web, grâce à la bibliothèque `js_of_ocaml`. Attention par contre, dans ce cas les bibliothèques externes mentionées précédemment ne seront probablement pas disponibles (parties internes en C).

5 Rendu de projet

Le code fourni est disponible sur le gitlab de l'UFR d'informatique:
http://moule.informatique.univ-paris-diderot.fr:8080/letouzey/pfav-2015
Vous devez effectuer un "fork" de cet embryon de projet, et nous donner accès à votre projet. Au moment du rendu de projet, vous devrez nous envoyer un mail précisant:

- La composition de votre binôme
- L'adresse de votre archive sur le gitlab de l'UFR
- Quelle version de cette archive nous devons considérer: quelle branche s'il y en a plusieurs, quel tag si vous en avez posé un, etc.

Votre projet devra pouvoir être compilé en un programme autonome via la commande `make`, où à la rigueur via un script shell `build.sh`. Un fichier README devra expliquer succinctement comment se servir de votre programme.

Un rapport de 5 pages maximum—nommé `rapport.pdf` et contenu dans l'archive—devra préciser vos choix de conceptions, les difficultés rencontrées, les solutions apportées, et faire le point sur les fonctionnalités de votre projet.

La date de rendu du projet sera en janvier 2016, et il y aura une soutenance également en janvier 2016. Les dates précises vous seront communiquées ultérieurement.

6 Autres informations

Binômes. Ce projet est à réaliser par groupe de deux maximum. Nous contacter si vous ne trouvez pas de binômes. Attention, même s'il s'agit d'un travail de groupe, chacun devra parfaitement connaître l'ensemble du code réalisé, lors des soutenances les questions et les notations pourront être individualisées.

Conseils méthodologiques. L'accent devra être mis sur la *lisibilité* et la *clarté* du code produit. En particulier:

- Indentez systématiquement et évitez les lignes de plus de 80 colonnes.
- Choisissez des noms de fonctions et de variables parlants.
- Utilisez les commentaires à bon escient : ni trop, ni trop peu. Un commentaire doit apporter quelque chose: organisation, point délicat, justification ...
- Si une fonction dépasse un écran de long, il est temps de songer à subdiviser en sous-fonctions, chacunes s'occupant d'une chose à la fois.
- OCaml permet très facilement la création de types de données personnalisés, ne vous en privez pas...
- Le copier-coller de code est à proscrire. A la place, mieux vaut prendre le temps de regrouper les choses similaires via des fonctions génériques.

Bibliothèques externes OCaml permet facilement l'utilisation de bibliothèques supplémentaires, par exemple pour le graphisme, et ce sujet en suggère un certain nombre. Si vous désirez utiliser d'autres bibliothèques non mentionnées ici, vous devez nous contacter au préalable et nous demander l'autorisation.

Annexe: Quelques exercices de bac

Voir le sous-répertoire `doc` dans l'archive du projet

Changelog

Version 1: version initiale