Master Parisien de Recherche en Informatique
Cours 2-7-2 Assistants de preuve
2005–2006

Projet

Tactique réflexive de dérivation formelle

Le projet devra être réalisé sur machine, le rapport final comprendra une spécification des différents points demandés (définitions, énoncés des résultats sans les preuves) accompagnée de commentaires. Ce rapport sera rendu lors du dernier cours le **16 février 2006**, et l'ensemble des fichiers Coq devra être envoyé par courrier électronique à l'adresse `filliatr@lri.fr` avec le sujet « `projet cours 2-7-2` ».

1 Objectif

L'objectif de ce projet est de réaliser une tactique réflexive de dérivation formelle. Plus précisément, il s'agira d'une tactique qui prend en argument une fonction f sur les réels et qui construira d'une part la fonction dérivée f' et d'autre part la preuve que f' est bien la dérivée de f.

On utilisera la formalisation des réels contenue dans la bibliothèque standard de Coq (sous-répertoire `theories/Reals/`) que l'on importe avec la commande

```
Require Export Reals.
```

Cette bibliothèque axiomatise les réels sous la forme d'un type `R` (voir les fichiers `Rdefinitions` et `Raxioms`). La notion de dérivée est définie dans le fichier `Ranalysis1` sous la forme d'un prédicat

```
derivable_pt_lim : (R -> R) -> R -> R -> Prop
```

tel que `derivable_pt_lim` $f\ x\ l$ signifie « la fonction f est dérivable en x de dérivée l ».

Cette bibliothèque définit d'autre part un certain nombre d'opérations et de fonctions mathématiques standards (opérations rationnelles, fonctions trigonométriques, etc.).

2 Fonctions infiniment dérivables en tout point

Dans un premier temps, on ne considère qu'un sous-ensemble de fonctions, toutes dérivables en tout point et C^∞, à savoir les fonctions obtenues à partir de :

- l'identité, les fonctions constantes, l'exponentielle et les fonctions trigonométriques sinus et cosinus ;
- l'addition, la soustraction, le produit, la composition et la puissance de fonctions déjà obtenues.

1. Définir un type inductif `fonction : Set` représentant la syntaxe abstraite de cet ensemble de fonctions.
2. Définir une fonction d'interprétation `interp : fonction -> R -> R` interprétant une expression de fonction comme une fonction de $\mathbb{R}$ dans $\mathbb{R}$.

3. Définir une fonction `deriv : fonction -> fonction` de dérivation formelle.
4. Montrer la correction de la fonction `deriv`, à savoir

$$\forall f : \texttt{fonction}.\ \forall x.\ \texttt{derivable\_pt\_lim}\ (\texttt{interp}\ f)\ x\ (\texttt{interp}\ (\texttt{deriv}\ f)\ x)$$

   On pourra utiliser tout résultat de la bibliothèque `Reals` (essayer la commande `SearchAbout derivable_pt_lim.`).
5. Définir, en utilisant la commande `Ltac`, une tactique `Derive` qui prend en argument un terme f de type `R -> R` et ajoute une hypothèse de la forme

$$\forall x.\ \texttt{derivable\_pt\_lim}\ f\ x\ (f'\ x)$$

   pour un certain terme f' de type `R -> R` (la dérivée de f). La seule difficulté consiste à transformer le terme f en une expression e de type `fonction` telle que `interp` e est convertible avec f. On pourra s'inspirer pour cela de la définition de `deriv_proof` dans `Ranalysis`.

3 Fonctions dérivables sur un sous-ensemble de $\mathbb{R}$

On souhaite maintenant pouvoir traiter le cas de fonctions non dérivables en tout point comme par exemple la division, l'inverse, la racine carrée, la valeur absolue et le logarithme.

Cela implique que le théorème de correction de la section précédente n'est pas valable pour tout x réel, mais seulement sur un certain domaine, représenté par un prédicat de type `R -> Prop`.

1. Étendre le développement avec les fonctions citées ci-dessus
2. Écrire une fonction `derivable : fonction -> R -> Prop` qui calcule le domaine de dérivabilité d'une fonction.
3. Démontrer le nouveau théorème de correction :

$$\forall f : \texttt{fonction}.\ \forall x.$$
$$\texttt{derivable}\ f\ x \Rightarrow \texttt{derivable\_pt\_lim}\ (\texttt{interp}\ f)\ x\ (\texttt{interp}\ (\texttt{deriv}\ f)\ x)$$

4 Variables de fonctions

On considère maintenant la possibilité de calculer la dérivée formelle d'expressions contenant des fonctions n'appartenant pas au language de fonctions étudié dans les précédentes sections. Pour cela, notre language d'expressions abstraites (type `fonction`) sera étendu avec un constructeur de "variables". Pour chaque sous-expressions non reconnue, la tactique engendrera une nouvelle variable. On fabriquera aussi une valuation, qui associera à chaque variable la fonction qu'elle représente.

On utilisera le type `varmap` défini dans `Quote` (dans `contrib/ring/`). Il s'agit d'un type d'arbres binaires équilibrés, chaque nœud étant étiqueté. Les variables seront représentées par des chemins dans ces arbres binaires (type `index`). La fonction `varmap_find` permet de rechercher l'étiquette du nœud atteint en suivant le chemin passé en argument. L'acces au chemin i dans un arbre v sera noté dans la suite v_i.

Ainsi, la valuation sera un objet de type `varmap (R->R)`, et la fonction d'interprétation deviendra de type `varmap (R->R) -> fonction -> R -> R`.

Si on considère une expression e contenant des occurrences d'une fonction f non reconnue, sa dérivée sera exprimée en fonction de f et de sa dérivée f', cette dernière n'ayant a priori aucune

raison d'être reconnue. En conséquence, la dérivée de e sera exprimée dans une valuation dont la taille sera le double de celle de la valuation de e : si v est la valuation de e, on considèrera une valuation v' etiquetée par les dérivées des étiquettes de v, et la dérivée de e sera exprimée dans `Node_vm` α v v' où α est une fonction quelconque puisqu'elle ne sera pas référencée.

Enfin, concernant les domaines de dérivation, on introduira une dernière valuation (de type `varmap (R->Prop)`) associant à chaque fonction non reconnue son domaine de dérivation.

1. Réécrire les fonctions d'interprétation, de dérivation et de calcul du domaine de dérivabilité.
2. Démontrer le lemme de correction suivant :

   $\forall f$: `fonction`.
   $\forall v$: `varmap` (`R` $\rightarrow$ `R`).
   $\forall v'$: `varmap` (`R` $\rightarrow$ `R`).
   $\forall d$: `varmap` (`R` $\rightarrow$ `Prop`).
   $(\forall i$: `index`. $\forall x.\ d_i\ x \Rightarrow$ `derivable_pt_lim` $v_i\ x\ (v'_i\ x)) \Rightarrow$
   $\forall x$. `derivable` $d\ f\ x \Rightarrow$
   `derivable_pt_lim` (`interp` $v\ f$) x (`interp` (`Node_vm` $\alpha\ v\ v'$) (`deriv` f) x)

3. Écrire une tactique qui, étant donné t : `R->R` construit une `varmap` équilibrée v regroupant toutes les fonctions non reconnues dans t. On évitera de faire apparaître plusieurs fois la même fonction.
4. Écrire une tactique qui, étant donnés t et une varmap v, construit une expression f telle que `interp` $v\ f$ soit convertible avec t.
5. Réécrire la tactique qui instanciera le théorème de correction (on ne cherchera pas à instancier les variables v' et d, ni à décharger les hypothèses de ce lemme), et simplifie l'énoncé de façon à ne plus faire apparaître la fonction d'interprétation.