Master Parisien de Recherche en Informatique
Cours 2-7-2 Assistants de preuve
2006–2007

Projet
Mini-Coq

Le projet devra être réalisé individuellement. Le projet devra être envoyé par courrier électronique à l'adresse `filliatr@lri.fr`, avec le sujet « `projet cours 2-7-2` », sous la forme d'une archive contenant les fichiers sources et un rapport synthétique.

1 Objectif

Le projet consiste en la réalisation d'un mini assistant de preuve, sous la forme d'un vérificateur de preuve pour le calcul des constructions inductives. Plus précisément, ce programme prendra en entrée un ensemble de déclarations (paramètres, définitions et inductifs) et devra signaler les erreurs de typage, le cas échéant.

Ce projet pourra être réalisé dans le langage de programmation de votre choix. Dans le cas de l'utilisation d'un langage peu courant, un exécutable Linux devra être fourni avec les sources. Votre programme devra être suffisamment commenté pour que sa lecture par les enseignants ne soit pas trop difficile.

2 Calcul des Constructions Inductives

2.1 Syntaxe abstraite

$$
\begin{array}{rcl}
s & ::= & \texttt{Prop} \mid \texttt{Type}_i \\
e & ::= & s \mid x \mid \lambda x : e.\, e \mid e\, e \mid \Pi x : e.\, e \\
 & \mid & \texttt{fix}\; x\; (x : e) \ldots (x : e) : e := e \\
 & \mid & \texttt{match}\; e\; \texttt{return}\; e\; \texttt{with}\; x \ldots x \Rightarrow e \mid \ldots \mid x \ldots x \Rightarrow e\; \texttt{end}
\end{array}
$$

où x est un identificateur, i un entier positif.

$$
\Gamma \quad ::= \quad \emptyset \mid \Gamma,\, x : e \mid \Gamma,\, x : e := e \mid \Gamma,\, \texttt{ind}\; \overline{x : \vec{e}} : e := \overline{x : \vec{e}}
$$

2.2 Conversion

$$
\frac{}{\Gamma \vdash ((\lambda x : T.\, e_1)\; e_2) =_\beta e_1[x \leftarrow e_2]}
$$

$$
\frac{x : T := e \in \Gamma}{\Gamma \vdash x =_\delta e}
$$

$$
\frac{}{\Gamma \vdash \texttt{match}\; C_i\; \vec{e}\; \texttt{return}\; P\; \texttt{of}\; C_1\; \overrightarrow{x_1} \Rightarrow e_1 \mid \ldots \mid C_n \overrightarrow{x_n} \Rightarrow e_n\; \texttt{end} =_\iota e_i[\overrightarrow{x_i} \leftarrow \vec{e}]}
$$

$$
\frac{e_n \text{ a la forme } C\; \vec{e'} \text{ où } C \text{ est un constructeur}}{\begin{array}{ll}\Gamma \vdash & (\texttt{fix}\; f\; (x_1 : T_1) \ldots (x_n : T_n) : T := e)\; e_1 \ldots e_n \;\; =_\varphi \\ & e[f \leftarrow \texttt{fix}\; f\; (x_1 : T_1) \ldots (x_n : T_n) : T := e][x_i \leftarrow e_i]\end{array}}
$$

$=$ est la clôture réflexive, symétrique, transitive et contextuelle de $=_{\beta\delta\iota\phi}$

$$\frac{\Gamma \vdash t = u}{\Gamma \vdash t \leq u}$$

$$\frac{i \leq j}{\Gamma \vdash \texttt{Type}_i \leq \texttt{Type}_j} \qquad \frac{\Gamma \vdash T = T' \quad \Gamma, x : T \vdash U \leq U'}{\Gamma \vdash \Pi x : T.\, U \leq \Pi x : T'.\, U'}$$

2.3 Règles de typage

On présente les règles du Calcul des Constructions Inductives sous une forme dirigée par la syntaxe ce qui permet l'usage de la règle de conversion avec sous-typage.

Typage des expressions

$$\frac{\Gamma(x) = (x : T := e)}{\Gamma \vdash x : T} \qquad \frac{\Gamma(x) = (x : T)}{\Gamma \vdash x : T}$$

$$\frac{\Gamma \vdash T : s \qquad \Gamma, x : T \vdash e : U}{\Gamma \vdash \lambda x : T.\, e : \Pi x : T.U} \qquad \frac{\Gamma \vdash e : \Pi x : U.V \qquad \Gamma \vdash U \leq U' \qquad \vdash e' : U'}{\Gamma \vdash e\, e' : V[x \leftarrow e']}$$

$$\frac{\Gamma \vdash T : s \qquad \Gamma, x : T \vdash U : s'}{\Gamma \vdash \Pi x : T.\, U : max(s, s')} \qquad \frac{}{\Gamma \vdash \texttt{Prop} : \texttt{Type}_1} \qquad \frac{}{\Gamma \vdash \texttt{Type}_i : \texttt{Type}_{i+1}}$$

$$\frac{\Gamma(I) = (\texttt{ind}\ I\ \overrightarrow{x : T} : v := \overrightarrow{C : U})}{\Gamma \vdash I : \Pi \overrightarrow{x : T}.\, v} \qquad \frac{\Gamma(c_j) = (\texttt{ind}\ I\ \overrightarrow{x : T} : v := \overrightarrow{C : U})}{\Gamma \vdash C_j : \Pi \overrightarrow{x : T}.\, U_j}$$

$$\frac{\begin{array}{l} \Gamma \vdash e : I\, \vec{t}\, \vec{v} \\ \Gamma(I) = (\texttt{ind}\ I\ \overrightarrow{x : T} : \Pi \overrightarrow{z : V}.\, s := C_1 : \Pi \overrightarrow{x_1 : U_1}.\, I\ \vec{x}\ \vec{v_1} \ldots C_m : \Pi \overrightarrow{x_m : U_m}.\, I\ \vec{x}\ \vec{v_m}) \\ \Gamma, \overrightarrow{z : V}, y : I\, \vec{t}\, \vec{z} \vdash U : s \\ s \text{ est une sorte d'élimination autorisée pour } I \\ \text{pour tout } j,\ \Gamma, \overrightarrow{x_j : U_j[\vec{x} \leftarrow \vec{t}]} \vdash e_j : U[\vec{z} \leftarrow \vec{v_j}[\vec{x} \leftarrow \vec{t}]][y \leftarrow C_j\ \vec{t}\ \vec{x_j}] \end{array}}{\Gamma \vdash \texttt{match}\ e\ \texttt{return}\ \lambda \overrightarrow{z : V}.\, \lambda y : I\, \vec{t}\, \vec{z}.U\ \texttt{with}\ C_1\ \vec{x_1} \Rightarrow e_1 \mid \ldots \mid C_m\ \vec{x_m} \Rightarrow e_m\ \texttt{end} : U[\vec{z} \leftarrow \vec{v}][y \leftarrow e]}$$

$$\frac{\begin{array}{l} \Gamma(I) = (\texttt{ind}\ I\ \overrightarrow{x : T} : v := \overrightarrow{c : U}) \\ \Gamma, \overrightarrow{y : T}, x : I \vdash U \\ \Gamma, \overrightarrow{y : T}, x : I, f : (\Pi \overrightarrow{y : T}.\, \Pi x : I.\, U) \vdash e : U \qquad e|_f^{\emptyset} < x \end{array}}{\Gamma \vdash (\texttt{fix}\ f\ \overrightarrow{y : T}\ (x : I) : U := e) : (\Pi \overrightarrow{y : T}.\, \Pi x : I.\, U)}$$

Typage des contextes

$$\frac{}{\emptyset \vdash} \qquad \frac{\Gamma \vdash T : s}{\Gamma, x : T \vdash} \qquad \frac{\Gamma \vdash t : T}{\Gamma, x : T := t \vdash}$$

$$
\frac{\begin{array}{l}\text{pour tout } i,\ \Gamma, x_1 : T_1, \ldots, x_{i-1} : T_{i-1} \vdash x_i : T_i \\ \Gamma, \overrightarrow{x : T} \vdash V : \Pi \vec{y : w}.\, s \\ \text{pour tout } j,\ \Gamma, \overrightarrow{x : T}, I : \Pi \overrightarrow{x : T}.\, V \vdash U_j : s \\ \text{pour tout } j,\ U_j \text{ vérifie la condition de positivité pour } I\end{array}}{\Gamma, \mathtt{ind}\ I\ \overrightarrow{x : T} : V := \overrightarrow{C : U} \vdash}
$$

Notes :

– *max* est défini par

$$
\begin{array}{lcl}
max(s, \mathtt{Prop}) & = & \mathtt{Prop} \\
max(\mathtt{Prop}, \mathtt{Type_i}) & = & \mathtt{Type_i} \\
max(\mathtt{Type_j}, \mathtt{Type_i}) & = & \mathtt{Type_{max(i,j)}}
\end{array}
$$

– $\Gamma \vdash T : s$ en prémisse signifie « il existe U tel que $\Gamma \vdash T : U$ et U se réduit en une sorte s ». Pareillement pour $\Gamma \vdash e : \Pi \overrightarrow{x : T}.\, s$ ou $\Gamma \vdash e : \Pi x : T.\, u$.

– Pour x identificateur, $\Gamma(x)$ doit être compris comme la plus récente déclaration de Γ déclarant le nom x sachant que

  $\mathtt{ind}\ I\ \overrightarrow{x : T} : V := \overrightarrow{C : U}$ déclare les noms I et C_i
  $x : T$ déclare le nom x
  $x : T := t$ déclare le nom x

  Alternativement, on peut aussi s'arranger pour que les noms déclarés dans Γ soient tous distincts.

– Pour la définition de la condition de positivité, voir le manuel de référence de Coq, section 4.5.3.

– Pour la définition de sorte d'élimination autorisée pour un inductif, voir le manuel de référence, section 4.5.4.

– Pour la définition de $t|_f^\rho < x$ (condition de garde), on se réfèrera aux transparents du cours ou au manuel de référence, section 4.5.5.

– Les régles dirigées par la syntaxe présupposent un contexte initial bien typé. En particulier, on a $\Gamma \vdash t : T$ et $\Gamma \vdash$ ssi $\Gamma \vdash' t : T$, où $\vdash$ désigne la présentation « traditionnelle » des règles d'inférences[1]. Dans la pratique, puisque le rôle de Mini-Coq n'est que de construire incrémentalement des contextes bien formés à partir du contexte vide, l'invariant que Γ est bien formé dans les règles de typage sera préservé.

3 Syntaxe concrète

3.1 Conventions lexicales

Espaces, tabulations et retour-chariots constituent des blancs. Les commentaires sont délimités par `(*` et `*)` et peuvent être imbriqués. Les identificateurs et les constants entières sont définis par les expressions régulières ⟨*ident*⟩ et ⟨*num*⟩ suivantes :

⟨*digit*⟩	::=	`0-9`
⟨*num*⟩	::=	⟨*digit*⟩$^+$
⟨*alpha*⟩	::=	`a-z` | `A-Z`
⟨*ident*⟩	::=	⟨*alpha*⟩ (⟨*alpha*⟩ | `_` | ⟨*digit*⟩)*

Les identificateurs suivants sont des mots clés :

```
Definition    Inductive    Parameter
Prop          Type         as
end           fix          forall
fun           in           match
return        with
```

Pour les identificateurs et les mots-clés, la casse des caractères est significative.

[1]Les calculs dirigés par la syntaxe ont été étudiés par Pollack, Typechecking in Pure Type Systems.

3.2 Grammaire

La grammaire BNF des fichiers sources considérée est donnée figure 1. Le point d'entrée est le non-terminal $\langle \mathit{file} \rangle$.

$\langle \mathit{file} \rangle$	::=	$\langle \mathit{decl} \rangle^\star$
$\langle \mathit{decl} \rangle$	::=	`Parameter` $\langle \mathit{ident} \rangle$ `:` $\langle \mathit{constr} \rangle$ `.`
	|	`Definition` $\langle \mathit{ident} \rangle$ `:` $\langle \mathit{constr} \rangle$ `:=` $\langle \mathit{constr} \rangle$ `.`
	|	`Inductive` $\langle \mathit{ident} \rangle$ $\langle \mathit{closed\_binder} \rangle^\star$ `:` $\langle \mathit{constr} \rangle$ `:=` $\langle \mathit{constructor} \rangle^\star$ `.`
$\langle \mathit{constr} \rangle$	::=	$\langle \mathit{ident} \rangle$ | `Prop` | `Type` | `Type` $\langle \mathit{num} \rangle$ | `(` $\langle \mathit{constr} \rangle$ `)`
	|	$\langle \mathit{constr} \rangle$ $\langle \mathit{constr} \rangle$
	|	$\langle \mathit{constr} \rangle$ `->` $\langle \mathit{constr} \rangle$
	|	`forall` $\langle \mathit{binders} \rangle$ `,` $\langle \mathit{constr} \rangle$
	|	`fun` $\langle \mathit{binders} \rangle$ `=>` $\langle \mathit{constr} \rangle$
	|	`match` $\langle \mathit{constr} \rangle$ $\langle \mathit{as\_in} \rangle$? `return` $\langle \mathit{constr} \rangle$ `with` $\langle \mathit{branch} \rangle^\star$ `end`
	|	`fix` $\langle \mathit{ident} \rangle$ $\langle \mathit{closed\_binder} \rangle^+$ `:` $\langle \mathit{constr} \rangle$ `:=` $\langle \mathit{constr} \rangle$
$\langle \mathit{as\_in} \rangle$	::=	`as` $\langle \mathit{ident} \rangle$ `in` $\langle \mathit{constr} \rangle$
$\langle \mathit{binders} \rangle$	::=	$\langle \mathit{ident} \rangle^+$ `:` $\langle \mathit{constr} \rangle$
	|	$\langle \mathit{closed\_binder} \rangle^+$
$\langle \mathit{closed\_binder} \rangle$	::=	`(` $\langle \mathit{ident} \rangle^+$ `:` $\langle \mathit{constr} \rangle$ `)`
$\langle \mathit{branch} \rangle$	::=	`|` $\langle \mathit{ident} \rangle^+$ `=>` $\langle \mathit{constr} \rangle$
$\langle \mathit{constructor} \rangle$	::=	`|` $\langle \mathit{ident} \rangle$ `:` $\langle \mathit{constr} \rangle$

FIG. 1 – Grammaire des fichiers Mini-Coq

Les associativités et précédences des divers opérateurs sont données par la table suivante, de la plus faible à la plus forte précédence :

opérateur	**associativité**
`fun`, `forall`, `fix`	—
`->`	à droite
application	à gauche

Remarque : dans la notation

$$\texttt{match}\ t\ \texttt{return}\ u\ \texttt{with}\ x_1^1 \dots x_{n_1}^1 \Rightarrow e_1 \mid \dots \mid x_1^p \dots x_{n_p}^p \Rightarrow e_p\ \texttt{end},$$

l'expression u est supposée être un terme de la forme $\lambda x : \vec{T}.\lambda y : I\ \vec{t}\ \vec{x}.U$ et la syntaxe concrète associée sera

$$\texttt{match}\ t\ \texttt{as}\ y\ \texttt{in}\ I\ \vec{t}\ \vec{x}\ \texttt{return}\ u\ \texttt{with}\ x_1^1 \dots x_{n_1}^1 \Rightarrow e_1 \mid \dots \mid x_1^p \dots x_{n_p}^p \Rightarrow e_p\ \texttt{end}.$$

Inversement, si la clause $\langle \mathit{as\_in} \rangle$ est absente dans la syntaxe concrète, la syntaxe abstraite correspondante restera de la forme

$$\texttt{match}\ t\ \texttt{return}\ \lambda x : \vec{T}.\lambda y : I\ \vec{t}\ \vec{x}.U\ \texttt{with}\ x_1^1 \dots x_{n_1}^1 \Rightarrow e_1 \mid \dots \mid x_1^p \dots x_{n_p}^p \Rightarrow e_p\ \texttt{end},$$

mais avec la propriété que ni les $\vec{x}$, ni y n'ont d'occurrence dans U.

4 Travail demandé

Votre programme doit prendre sur sa ligne de commande un unique argument, qui est le nom d'un fichier Mini-Coq à analyser. Ce fichier peut être supposé conforme à la syntaxe concrète ci-dessus. Si le fichier est valide, votre programme doit terminer avec le code de sortie 0, sans rien afficher. Sinon, l'erreur de typage doit être signalée par un message de la forme

```
File "f.v", line 12, characters 7-15:
application mal typée
```

et le programme doit terminer avec le code de sortie 1. Le format de la première ligne doit être compatible avec la fonction `next-error` d'Emacs, mais le message de la seconde ligne peut être quelconque (en français ou en anglais, selon votre choix).

Indications

- On pourra commencer par une version qui ne vérifie pas le critère de (stricte) positivité des inductifs, ni la condition de garde des points-fixes, ni la correction des sortes d'élimination.
- Les lieurs pourront être représentés par la méthode de votre choix : variables nommées, indices de de Bruijn, etc. Si vous hésitez, nous vous conseillons d'adopter les indices de de Bruijn, en vous inspirant par exemple des notes de cours de Gérard Huet, « *Constructive Computation Theory* », disponibles ici : `http://pauillac.inria.fr/~huet/bib.html`.