Définition de la classe de réécriture à intégrer

Frédéric Blanqui

Lot 5.1

Technologie de vérification

Ajouter la réécriture au noyau de Coq

Définition de la classe de réécriture à intégrer

Description :	Nous définissons les objets qui doivent être rajoutés à Coq pour pouvoir définir des fonctions à l'aide de règles de récriture, et décrivons les conditions qui doivent être vérifiées pour que la correction de Coq soit préservée. Cela constitue une sorte de "cahier des charges" pour le future prototype.
Auteur(s) :	Frédéric Blanqui
Référence :	Averroes / Lot 5.1 / Fourniture 1 / V1.0
Date :	5 février 2004
Statut :	validé
Version :	1.0

Réseau National des Technologies Logicielles
Projet subventionné par le Ministère de la Recherche et des Nouvelles Technologies

CRIL Technology, France Télécom R&D, INRIA-Futurs, LaBRI (Univ. de Bordeaux – CNRS), LIX (École Polytechnique, CNRS) LORIA, LRI (Univ. de Paris Sud – CNRS), LSV (ENS de Cachan – CNRS)

Historique

8 octobre 2003	V 0.1	version préliminaire
5 février 2004	V 1.0	mise au format averroes

Table des matières

1 Contexte

Dans le système Coq, les fonctions et prédicats ne peuvent être définis que par induction, et deux propositions ne sont considerées équivalentes que si les définitions inductives (et la β-réduction) permettent de passer de l'une à l'autre. Cela est trop restrictif : certaines définitions inductives, bien que valides, ne sont pas acceptées par Coq ; et certaines égalités, pourtant montrées de manière inductive, ne sont pas utilisées par le système pour identifier deux propositions.

La récriture est une manière plus générale et plus souple de définir les fonctions et les prédicats qui englobent les définitions inductives, et qui permet d'identifier davantages de propositions, et donc de s'affranchir de davantage de détails purement calculatoires dans les preuves. C'est donc une extension conservatrice de Coq dans le sens où tout théorème prouvé sans utiliser de récriture est toujours valable dans Coq étendu avec de la récriture.

Depuis les premiers travaux sur la combinaison du λ-calcul et de la récriture de Breazu-Tannen et Gallier en 1988-1989, de nombreux chercheurs ont contribué à son étude. Nous ne citons que quelques travaux : [6, 3, 7, 5]. Pour le projet Averroes, nous nous appuierons sur les travaux parmi les plus récents : [4, 1, 2].

2 Discussion préliminaire

Avant de décrire les systèmes de récriture qui devront être acceptés dans la future extension de Coq, il convient de préciser quels objets de Coq seront ainsi "définis" par ces règles de récriture, car plusieurs options existent qui conditionnent le développement de cette extension.

Option 1 : considérer des règles de récriture définissant des axiomes ou paramètres. Pour prendre un exemple concret, dans cette approche, l'addition sur les entiers naturels pourrait être définie en déclarant un paramètre `plus :nat=>nat=>nat` et les règles `plus x O -> x` et `plus x (S y) -> S (plus x y)`. Cette approche est celle qui a été étudiée jusqu'à maintenant. Elle permet, pour ce qui est des fonctions, de se passer de la notion de récurseur. En fait, les récurseurs eux-mêmes sont des cas particuliers de telles définitions.

Option 2 : considérer des règles de récriture issues d'égalités prouvées. Dans cette approche, les fonctions continueraient d'être définies par induction mais, si une égalité prouvée peut être orientée de façon à préserver la terminaison et la confluence, elle pourrait être utilisée comme règle de récriture. Par exemple, une fois l'associativité de l'addition prouvée, celle-ci pourrait être utilisée comme règle de récriture. Cette approche commence à être étudiée par Nicolas Oury au LRI.

Nous avons choisi l'option 1 car c'est l'approche la mieux comprise actuellement et celle qui nous paraît la plus facile à mettre en oeuvre. Cependant, ces deux options ne sont pas exclusives l'une de l'autre et pourrait à terme être combinées.

Parallèlement à cela, il convient de préciser comment la récriture peut cohabiter avec le système de modules. Etant donné que cela fait encore l'objet de recherches, nous avons choisi de restreindre au minimum l'interaction entre récriture et modules. Ainsi, on ne change rien au système de modules. Il n'y a donc pas de types de module permettant de spécifier l'existence de règles de récriture, et toutes les règles de récriture déclarées dans un module (ou un foncteur) sont systématiquement exportées.

3 Description des conditions

Nous appellerons :

- **symbole** tout axiome ou paramètre que l'on souhaite définir à l'aide de règles de récriture.
- **constante** tout type inductif, constructeur, axiome ou paramètre non défini par récriture.
- **constante de prédicat** toute constante de type $(\vec{x}{:}\vec{T})s$ avec $s \in \{Set, Prop\}$.

– **symbole constructeur** tout symbole de type $(\vec{x} : \vec{T})C\vec{v}$ tel que $C : (\vec{z} : \vec{V})s$ est une constante de prédicat et $|\vec{v}| = |\vec{z}|$.

Rappel : la sorte d'un symbole ou d'une variable est le type de son type (qui doit justement être une sorte, c'est-à-dire, un élément de $\{Set, Prop, Type\}$).

3.1 Symboles définissables par récriture

Un symbole f doit être muni :

- d'un type $(\vec{x} : \vec{T})U$.
- de l'ensemble des arguments en lesquelles f doit être monotone (voir Section 3.8.2).
- de l'ensemble des arguments en lesquelles f doit être anti-monotone (voir Section 3.8.2).
- d'une théorie équationnelle (voir Section 3.2).
- d'une précédence.
- d'un statut décrivant comment, dans les appels récursifs des membres droits des règles, les arguments de f doivent être comparés (voir Section 3.3).

3.2 Théories équationnelles

Les théories équationnelles possibles pour un symbole f sont : théorie vide, associativité (A), commutativité (C), ou associativité et commutativité ensembles (AC). Pour que les équations soient typables :

- si f est commutatif alors le type de f doit être de la forme $T \Rightarrow T \Rightarrow U$.
- si f est A ou AC alors le type de f doit être de la forme $T \Rightarrow T \Rightarrow T$ avec T une constante.

Par ailleurs :

- f doit être de sorte Set ou $Prop$.[1]

En théorie, on peut avoir des théories équationnelles un peu plus complexes. Par exemple, f de type $(\vec{x} : \vec{T})U$, $i < j$, x_i non libre dans T_j et $f(\ldots x_i \ldots x_j \ldots) = f(\ldots x_j \ldots x_i \ldots)$.

3.3 Arguments strictement positifs et statuts

Un statut est une séquence $(m_1, \ldots, m_k)$ où chaque m_i est un ensemble non vide d'entiers positifs inférieurs ou égaux à $|\vec{x}|$. La comparaison entre deux séquences de termes $t_1, \ldots, t_n$ et $u_1, \ldots, u_n$ est faite de la manière suivante. Pour chaque $i \leq k$, on forme le multi-ensemble M_i (resp. N_i) des t_j (resp. u_j) tels que j appartient à m_i. Ensuite, on compare lexicographiquement $(M_1, \ldots, M_k)$ et $(N_1, \ldots, N_k)$. Les comparaisons lexicographique $(\{1\}, \ldots, \{n\})$ et multi-ensemble $(\{1,\ldots, n\})$ en sont des cas particuliers. Nous décrivons l'ordre utilisé sur les termes dans la Section 3.8.4.

Une constante de prédicat C est strictement positive si C est un type inductif[2] ou si, pour toute constante de prédicat $D : (\vec{z} : \vec{V})s$ équivalente à C, pour tout symbole constructeur $f : (\vec{x} : \vec{T})C\vec{v}$, pour toute constante de prédicat E équivalente à C, si E apparaît dans T_j alors $T_j = (\vec{y} : \vec{U})E\vec{w}$ et aucune constante de prédicat F équivalente à C n'apparaît dans $\vec{U}$ ou dans $\vec{w}$.

Etant donné une constante de prédicat $C : (\vec{z} : \vec{V})s$ et une séquence de termes $\vec{v}$ avec $|\vec{v}| = |\vec{z}|$, on dénote par $\vec{v}|_C$ la sous-séquence $v_{i_1} \ldots v_{i_n}$ telle que $i_1 < \ldots < i_n$ et $\{i_1, \ldots, i_n\}$ est l'ensemble des i tels que z_i est de sorte $Type$.

Soit maintenant un symbole $f : (\vec{x} : \vec{T})U$ de statut $(m_1, \ldots, m_k)$. L'ensemble des arguments strictement positifs de f est défini de la manière suivante : i est un argument strictement positif si et seulement s'il existe un terme $T_f^i = C\vec{t}$ tel que C est une constante de prédicat strictement positive et, pour tout $j \in m_i$, $T_j = C\vec{u}$ et $\vec{u}|_C = \vec{t}|_C$.

Maintenant :

[1] La récriture au niveau des univers n'a pas encore été étudiée.
[2] Les types inductifs de Coq sont tous strictement positifs.

- un symbole C ou AC doit avoir un statut multi-ensemble.
- deux symboles équivalents f et g doivent avoir le même statut, les mêmes arguments strictement positifs et, pour tout i strictement positif, on doit avoir $T_f^i = T_g^i$.

3.4 Règles de récriture

Une règle est donnée par :

- une paire de termes $l \to r$,
- un environnement de typage Γ pour les variables libres de r,
- une substitution ρ pour les variables libres de l qui ne sont pas dans Γ.

De plus :

- l doit être un terme de la forme $f\vec{l}$ avec $f : (\vec{x} : \vec{T})U$ un symbole et $\vec{l}$ des termes algébriques tels que $|\vec{l}| \leq |\vec{x}|$, un terme algébrique étant soit une variable, soit un symbole ou un constructeur d'un type inductif appliqué à des termes eux-mêmes algébriques.

La présence d'une substitution ρ peut paraître inhabituelle mais permet de linéariser certaines règles qui, si on exigeait que leur membre gauche soit bien typé, seraient non linéaires, sans pour autant remettre en cause la préservation du typage par récriture (voir Section 3.6). Par exemple, pour définir la concaténation de listes polymorphes (type $list : \star \Rightarrow \star$ avec les constructeurs $nil : (A : \star)listA$ et $cons : (A : \star)A \Rightarrow listA \Rightarrow listA$), $app : (A : \star)listA \Rightarrow listA \Rightarrow listA$, on peut prendre les règles :

$$\begin{aligned} app\ A'\ (nil\ A)\ l &\to l \\ app\ A'\ (cons\ A\ x\ l)\ l' &\to cons\ A\ x\ (app\ A\ l\ l') \end{aligned}$$

avec $\Gamma = A : \star, x : A, l : listA, l' : listA$ et $\rho = \{A' \mapsto A\}$. Si on exigeait que les membres gauches soient bien typés, il faudrait prendre $A = A'$ et donc vérifier, à chaque fois que l'on souhaiterait appliquer ces règles, que le 1er argument de *app* est identique au premier argument de *nil* ou *cons*, ce qui serait coûteux et, dans l'état actuel de nos connaissances, ne nous permettrait pas d'affirmer que le système est confluent (voir Section 3.5). Par ailleurs, si $A \neq A'$ et une instance d'un membre gauche est bien typée alors, d'après les règles de typage, A' est convertible à A. On n'a donc pas besoin de tester que A et A' sont égaux pour pouvoir appliquer la règle, et celle-ci préserve bien le typage.

3.5 Confluence

La confluence signifie que l'ordre des calculs n'a pas d'importance. Pour cela, on exige :

- Pour toute règle $l \to r$, l doit être linéaire : chaque variable doit avoir au plus une occurrence dans l.
- L'ensemble des règles doit former un système localement confluent modulo les théories équationnelles. Cela est décidable car la récriture termine (voir Section 3.8).

3.6 Préservation du typage par récriture

- Après application de ρ, le membre gauche est de type $U\gamma\rho$ avec $\gamma = \{\vec{x} \mapsto \vec{l}\}$ dans Γ : $\Gamma \vdash l\rho : U\gamma\rho$.
- Le membre droit r doit aussi être de type $U\gamma\rho$ dans Γ : $\Gamma \vdash r : U\gamma\rho$.
- Pour toute déclaration $x : T$ dans Γ, T doit être un type de x dérivé de l. Le type dérivé de t du sous-terme de t à la position p est défini de la manière suivante :
  - $\tau(f\vec{t}, ip) = \tau(t_i, p)$ si $p \neq \varepsilon$ (mot vide)
  - $\tau(f\vec{t}, i) = T_i\theta$ si $f : (\vec{x} : \vec{T})U$ et $\theta = \{\vec{x} \mapsto \vec{t}\}$

- Pour chaque variable x dans le domaine de ρ, il existe un sous-terme de l, disons t, tel que :
  - le type de t dérivé de l, noté T, est un terme algébrique contenant une occurrence de x et que des symboles constants,
  - t est de type $T\rho$ dans Γ.

3.7 Cohérence logique

La cohérence logique est assurée si chaque symbole f satisfait une des conditions suivantes :[3]

- f est un symbole constructeur.
- le type de f est de la forme $(\vec{x} : \vec{T})T_i$.

3.8 Terminaison

Afin de décrire les conditions de terminaison, nous devons d'abord introduire quelques définitions.

3.8.1 Positions positives et négatives

Tout d'abord, la notion de positivité. Les positions[4] positives et négatives d'un terme sont inductivement définies de la manière suivante :

- $Pos^{\delta}(s) = Pos^{\delta}(x) = \{\varepsilon \mid \delta = +\}$
- $Pos^{\delta}((x : U)V) = 1.Pos^{-\delta}(U) \cup 2.Pos^{\delta}(V)$
- $Pos^{\delta}([x : U]v) = 2.Pos^{\delta}(v)$
- $Pos^{\delta}(tu) = 1.Pos^{\delta}(t)$ si $t \neq f\vec{t}$
- $Pos^{\delta}(f\vec{t}) = \{1^{|\vec{t}|} \mid \delta = +\} \cup \bigcup\{1^{|\vec{t}|-i}2.Pos^{\varepsilon\delta}(t_i) \mid i \in Mon^{\varepsilon}(f), \varepsilon \in \{-,+\}\}$

où $\delta \in \{-,+\}$, $-+ = -$ et $-- = +$ (règle usuelle des signes).

3.8.2 Conditions de monotonie

Nous devons vérifier qu'une fonction dont des arguments ont été déclarés comme monotone ou anti-monotone est effectivement monotone et anti-monotone par rapport à ces arguments. Pour chaque règle $f\vec{l} \to r$:

- aucun symbol plus grand que f n'apparaît dans r,
- si i est un argument monotone (resp. anti-monotone) alors l_i est une variable apparaissant seulement positivement (resp. négativement) dans r.

Pour les symboles constants, les conditions de monotonie font partie des conditions d'accessibilité décrite en Section 3.8.3.

3.8.3 Arguments accessibles

Etant donné un symbole $f : (\vec{x} : \vec{T})U$, j est un argument accessible de f si :

- f est un symbole constructeur de $U = C\vec{v}$.
- tout symbole équivalent à C apparaît seulement positivement dans U_j.
- aucun symbole plus grand que C n'apparaît dans U_j.
- toute variable de sorte $Type$ libre dans U_j doit être égal à un v_i.
- si i est argument monotone (resp. anti-monotone) de C alors v_i doit être une variable apparaissant seulement positivement (resp. négativement) dans U_j.

[3] Il existe une autre condition, très générale, reposant sur la complétude des règles de récriture [4] mais sa mise en oeuvre nécessite davantage de recherche.

[4] Il s'agit de la notion habituelle de position : $Pos(s) = Pos(f) = Pos(x) = \varepsilon$ et $Pos(uv) = Pos((x : u)v) = Pos([x : u]v) = 1.Pos(u) \cup 2.Pos(v)$.

3.8.4 Ordre sur les termes

Etant donné une règle $(l \to r, \Gamma, \rho)$, on définit ci-après l'ordre utilisé pour comparer les appels récursifs de r avec l.

Un couple de termes (u, U) est accessible modulo une substitution ρ dans un couple de termes (t, T) si $t = f\vec{u}$, $f : (\vec{y} : \vec{U})C\vec{v}$, $|\vec{u}| = |\vec{y}|$, $u = u_j$, j est un argument accessible de f, $\gamma = \{\vec{y} \mapsto \vec{u}\}$, $T\rho = C\vec{v}\gamma\rho$, $U\rho = U_j\gamma\rho$ et aucun D équivalent à C n'apparaît dans $\vec{u}\rho$.

L'ordre utilisé est, pour les arguments non strictement positifs, la relation d'accessibilité. Pour les arguments strictement positifs, $(t, T) > (u, U)$ si :

- $t = f\vec{t}$, $f : (\vec{x} : \vec{T})C\vec{v}$, $\gamma = \{\vec{x} \mapsto \vec{t}\}$ et aucune constante de prédicat équivalente à C n'apparaît dans $\vec{v}\gamma\rho$.
- $u = x\vec{u}$ avec $x \in dom(\Gamma)$.
- (x, V) est accessible modulo ρ dans (t, T).
- $V\rho = x\Gamma = (\vec{y} : \vec{U})C\vec{w}$, $\delta = \{\vec{y} \mapsto \vec{u}\}$, $U\rho = C\vec{w}\delta$ et aucune constante de prédicat équivalente à C n'apparaît dans $\vec{U}\delta$.
- $\vec{v}\gamma\rho|_C = \vec{w}\delta|_C$.

3.8.5 Conditions de terminaison

Pour toute règle $(l \to r, \Gamma, \rho)$ avec $l = f\vec{l}$, $f : (\vec{x} : \vec{T})U$ et $\gamma = \{\vec{x} \mapsto \vec{l}\}$:[5]

- chaque variable de Γ est accessible dans $\vec{l}$.
- dans le membre droit r, tous les appels récursifs sont faits sur des arguments plus petits en utilisant le statut de f (voir Section 3.1 et 3.3) et l'ordre sur les termes décrit en Section 3.8.4.
- la règle est sûre : $\gamma\rho$ est une injection de l'ensemble des variables libres de $\vec{T}U$ de sorte $Type$ dans l'ensemble des variables de Γ de sorte $Type$.
- si f est de sorte $Type$ alors :
  - chaque variable libre de r de sorte $Type$ est égale à un l_i,
  - $l \to r$ n'a aucune paire critique avec les autres règles.

Références

[1] F. Blanqui. Inductive types in the Calculus of Algebraic Constructions. In *Proceedings of the 6th International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science 2701, 2003.

[2] F. Blanqui. Rewriting modulo in Deduction modulo. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science 2706, 2003.

[3] F. Blanqui. *Théorie des Types et Récriture*. PhD thesis, Université Paris XI, Orsay, France, 2001. Available in english as "Type Theory and Rewriting".

[4] F. Blanqui. Definitions by rewriting in the Calculus of Constructions, 2003. To appear in Mathematical Structures in Computer Science.

[5] J. Chrząszcz. *Implementation of a module system in Coq*. PhD thesis, Université d'Orsay, France and Warsaw University, Poland, 2003. In preparation.

[6] M. Fernández. *Modèles de calculs multiparadigmes fondés sur la réécriture*. PhD thesis, Université Paris XI, Orsay, France, 1993.

[7] D. Walukiewicz-Chrząszcz. *Termination of Rewriting in the Calculus of Constructions*. PhD thesis, Warsaw University, Poland and Université d'Orsay, France, 2003.

[5] En théorie, on peut également rajouter n'importe quel système de récriture du premier ordre, convergent et non-dupliquant, mais cela nécessite l'utilisation d'outils externes pour vérifier la terminaison.