Formalization of a \( \lambda \)-Calculus with Explicit Substitutions in Coq

Amokrane SAÏBI*

Projet COQ (INRIA-Rocquencourt)
B.P. 105 - 78153 Le Chesnay CEDEX

Abstract. We present a formalization of the \( \lambda \sigma \)-calculus[9] in the Coq V5.8 system[6]. The principal axiomatized result is the confluence of this calculus. Furthermore we propose a uniform encoding for many-sorted first order term rewriting systems, on which we study the local confluence by critical pairs analysis.

1 Introduction

The \( \lambda \sigma \)-calculus[2] (called also \( \lambda \)-calculus with explicit substitutions) is an extension of the \( \lambda \)-calculus where substitutions are manipulated explicitly. It is a suitable setting for studying the abstract machines since it is a bridge between \( \lambda \)-calculus and its concrete implementations.

In the \( \lambda \sigma \)-calculus, the substitutions have a syntactic representation. Thus if \( a \) is a term and \( s \) a substitution then \( a[s] \) represents the term \( a \) on which we should perform \( s \). The principal rule of the \( \lambda \sigma \)-calculus is Beta which starts the substitution operation:

\[
(\lambda x.a)b \rightarrow a[(b/x) \cdot id]
\]

\([b/x] \cdot id\) denotes the substitution that replaces \( x \) by \( b \) and does not affect any other variable (\( \cdot \) is called \textit{cons} and \textit{id} is the identity substitution). Generally, we can view a substitution as an environment (a list of pairs "term/variable"). To avoid name clashes, the \( \lambda \sigma \)-calculus uses de Bruijn's notation for the \( \lambda \)-calculus:

\[
a ::= n | aa | \lambda a
\]

The variable names are replaced by natural numbers (\( n \)) recording their binding depth, for example: \( \lambda x.\lambda y.x y \) becomes \( \lambda \lambda 10 \). In the \( \lambda \sigma \)-calculus, these indices correspond also to a position in an environment, thus \textit{Beta} becomes:

\[
(\lambda a)b \rightarrow a[b \cdot id]
\]

The substitution operation is then performed step by step by a first order term rewriting system.

* This research was partially supported by ESPRIT Basic Research Action "TYPES."

...
There are several versions of \(\lambda\)-calculus with explicit substitutions\[2, 5, 9\].
The \(\lambda\sigma\)\-calculus (initially called \(\lambda\)Env\-calculus), defined by T. Hardin and J.-J. Lévy, is a particular \(\lambda\sigma\)\-calculus that verifies the confluence property on open terms.

In this paper we describe a formalization of this calculus and the encoding of its confluence proof in the Coq V5.8 proof assistant system. This system is an implementation of the Calculus of Constructions\[3\] extended by a primitive mechanism of inductive definitions\[14\]. Through this development, we propose a uniform encoding for many-sorted first order term rewriting systems on which we study the local confluence by critical pairs analysis. Only the main steps of the proofs are described in this paper. The complete script Coq proof is available by ftp on ftp.inria.fr as a part of the current distribution.

The Coq system is not described in this paper. A complete presentation is given in \[14\], \[6\] and \[3\]. Moreover we suppose the reader familiar with basic notions in rewriting (see \[10\]) and \(\lambda\sigma\)\-calculus (see \[2, 5, 7, 9\]).

2 Relations in Coq

In this section, we state some fundamental properties of a binary relation \(R\) on a set \(A\), which we call reduction. The relation \(a \rightarrow b\), often noted \(a \xrightarrow{R} b\), is read "\(a\) reduces to \(b\)" and \(b\) is called a reduct of \(a\). We will also see the encoding of these properties in the Coq system.

2.1 Generalities

We define \(A\) as a variable of type \(\text{Set}\) and \(R\) as a characteristic function, i.e. the function that takes two elements of \(A\) and states whether the second is a reduct of the first.

Section Rels.

Variable \(A : \text{Set}\).
Variable \(R : A \rightarrow A \rightarrow \text{Prop}\).

The composition of relations \(R_1\) and \(R_2\), i.e. \(R_1 R_2 = \{(x, y) \mid \exists z x R_1 z \text{ and } z R_2 y\}\), is translated in Coq by (comp_rel \(R_1\) \(R_2\)):

```
Inductive Definition comp_rel [\(R_1, R_2 : A \rightarrow A \rightarrow \text{Prop}\)]: A -> A -> Prop
    = comp_2rel : (x~y,z:A)(l%i x y) -> (R2 y z) -> (comp_rel R1 R2 x z).
```

The inclusion of relations \(R_1 \subseteq R_2\) is encoded by (inclus \(R_1\) \(R_2\)):

```
Definition inclus = [\(R_1, R_2 : A \rightarrow A \rightarrow \text{Prop}\)](x,y:A)(R1 x y) -> (R2 x y).
```

The transitive closure \(R^+\) of \(R\), is defined in Coq by (rel_plus \(R\)):