UNIVERSITÀ
DEGLI STUDI
DI UDINE

Università degli studi di Udine

Formalizing a lazy substitution proof system for μ-calculus in the Calculus of Inductive Constructions

Original

Availability:
This version is available http://hdl.handle.net/11390/678312 since 2016-11-26T18:10:33Z

Publisher:
Springer Verlag

Published
DOI:10.1007/3-540-48523-6_52

Terms of use:
The institutional repository of the University of Udine (http://air.uniud.it) is provided by ARIC services. The aim is to enable open access to all the world.

Publisher copyright

(Article begins on next page)

18 April 2024

Formalizing a Lazy Substitution Proof System for μ-Calculus in the Calculus of Inductive Constructions

Marino Miculan*

Dipartimento di Matematica e Informatica, Università di Udine
Via delle Scienze, 206, I-33100, Udine, Italy. miculan@dimi.uniud.it

Abstract. We present a Natural Deduction proof system for the propositional modal μ-calculus, and its formalization in the Calculus of Inductive Constructions. We address several problematic issues, such as the use of *higher-order abstract syntax* in inductive sets in presence of recursive constructors, the encoding of modal (sequent-style) rules and of context sensitive grammars. The formalization can be used in the system Coq, providing an experimental computer-aided proof environment for the interactive development of error-free proofs in the μ-calculus. The techniques we adopt can be readily ported to other languages and proof systems featuring similar problematic issues.

Introduction

The μ-calculus, often referred to as μK, is a temporal logic which subsumes many modal and temporal logics, such as PDL, CTL, CTL^*, $ECTL$. Despite its expressive power, μK enjoys nice properties such as decidability, axiomatizability and the finite model property. Therefore, the μ-calculus is an ideal candidate as a logic for the verification of processes. Nevertheless, like any formal systems, its applicability to non trivial cases is limited by long, difficult, error-prone proofs.

This drawback can be (partially) overcome by supplying the user with a *computer-aided proof environment*, that is, a system in which he can represent (*encode, formalize*) the formal system, more or less abstractly: its syntax, axioms, rules and inference mechanisms. After having supplied the proof environment with a representation of the formal system, the user should be able to correctly manipulate (the representations of) the proofs.

Clearly, the implementation of a proof environment for a specific formal system is a complex, time-consuming, and daunting task. An alternative, and promising solution is to develop a general theory of logical systems, that is, a *Logical Framework*. A Logical Framework is a metalogical formalism for the specification of both the *syntactic* and the *deductive* notions of a wide range of formal systems. Logical Frameworks provide suitable means for representing and deal with, in the metalogical formalism, the *proofs* and *derivations* of the object system. Much of the implementation effort can be expended once and for all; hence, the implementation of a Logical Framework yields a *logic-independent proof development environment*. Such an environment is able to check validity of deductions in any formal system, after it has been provided by the specification of the system in the formalism of the Logical Framework.

In recent years, several different frameworks have been proposed, implemented and applied to many formal systems. *Type theories* have emerged as leading

* Work partially supported by Italian MURST-97 grant *Tecniche formali...*

J. Wiedermann et al. (Eds.): ICALP'99, LNCS 1644, pp. 554–564, 1999.
© Springer-Verlag Berlin Heidelberg 1999

candidates for Logical Frameworks. Simple typed λ-calculus and minimal intuitionistic propositional logic are connected by the well-known *proposition-as-types* paradigm [3]. Stronger type theories, such as the *Edinburgh Logical Framework*, the *Calculus of Inductive Constructions* and *Martin-Löf's type theory*, were especially designed, or can be fruitfully used, as a logical framework [7,2,15]. In these frameworks, we can represent faithfully and uniformly all the relevant concepts of the inference process in a logical system: syntactic categories, terms, assertions, axiom schemata, rule schemata, tactics, etc. via the *judgments-as-types, proofs-as-λ-terms* paradigm [7]. The key concept is that of *hypothetico-general* judgment [11], which is rendered as a type of the dependent typed λ-calculus of the Logical Framework. With this interpretation, a judgment is viewed as a type whose inhabitants correspond to proof of this judgment.

It is worthwhile noticing that Logical Frameworks based on type theory directly give rise to proof systems in *Natural Deduction style* [6]. This follows from the fact that the typing systems of the underlying λ-calculi are in Natural Deduction style, and rules and proofs are represented by λ-terms. As it is well-known, Natural Deduction style systems are more suited to the practical usage, since they allow for developing proofs the way mathematicians normally reason.

These type theories have been implemented in logic-independent systems such as Coq, LEGO and ALF [2,9,10]. These systems can be readily turned into interactive proof development environments for a specific logic: we need only to provide the specification of the formal system (the *signature*), i.e. a declaration of typed constants corresponding to the syntactic categories, term constructors, judgments, and rule schemata of the logic. It is possible to prove, informally but rigorously, that a formal system is *adequately* represented by its specification. This proof usually exhibit bijective maps between objects of the formal system (terms, formulæ, proofs) and the corresponding λ-terms of the encoding.

In this paper, we investigate the applicability of this approach to the propositional μ-calculus. Due to its expressive power, we adopt the Calculus of Inductive Constructions (CIC), implemented in the system Coq. Beside its expressive power and importance in the theory and verification of processes, the μ-calculus is interesting also for its syntactic and proof theoretic peculiarities. These idiosyncrasies are mainly due to the negative arity of "μ" (i.e., the bound variable x ranges over the same syntactic class of $\mu x\varphi$); a context-sensitive grammar due the condition on $\mu x\varphi$; rules with complex side conditions (sequent-style "proof" rules). These anomalies escape the "standard" representation paradigm of CIC; hence, we need to accommodate special techniques for enforcing these peculiarities. Moreover, since generated editors allow the user to reason "under assumptions", the designer of a proof editor for a given logic is urged to look for a Natural Deduction formulation of the system. Hence, we introduce a new proof system in Natural Deduction style for μK, the *lazy substitution* system $\mathbf{N}^{ls}_{\mu K}$. This system should more natural to use than traditional Hilbert-style systems; moreover, it takes best advantage of the possibility of manipulating assumptions offered by CIC in order to implement the problematic substitution of formulæ for variables. In fact, substitutions are delayed as much as possible, and are kept in the derivation context by means of assumptions. This mechanism fits perfec-

tly the stack discipline of assumptions of Natural Deduction, and it is neatly formalized in CIC.

Due to lack of space, full proofs and the complete Coq signature will be omitted; see [12] for an extended version of this paper.

1 Syntax, Semantics and Consequence Relation of μK

The language of μK is an extension of the syntax of propositional dynamic logic. Let Act be a set of *actions* (ranged over by a, b, c), and Var a set of propositional variables (ranged over by x, y, z); then, the syntax of the μ-calculus on Act is:

$$\Phi: \quad \varphi, \psi ::= ff \mid \neg\varphi \mid \varphi \supset \psi \mid [a]\,\varphi \mid x \mid \mu x\varphi$$

where the formation of $\mu x\varphi$ is subject to the *positivity condition:* every occurrence of x in φ has to appear inside an even number of negations (In the following we will spell out this condition more in detail). We call *preformulæ* the language obtained by dropping the positivity condition. The variable x is *bound* in $\mu x\varphi$; the usual conventions about α-equivalence apply. Given a set $X \subseteq Var$ of variables, we denote by $\Phi_X \stackrel{\text{def}}{=} \{\varphi \in \Phi \mid \mathrm{FV}(\varphi) \subseteq X\}$ the set of formulæ with free variables in X. Capture-avoiding substitutions are the usual maps $\Phi \to \Phi$, written as lists of the form $\{\varphi_1/x_1, \ldots, \varphi_n/x_n\}$; they are ranged over by σ, τ. We denote by $\varphi\sigma$ the formula obtained by applying the substitution σ to φ.

The interpretation of μ-calculus comes from Modal Logic. A model for the μ-calculus is a transition system, that is, a pair $\mathcal{M} = \langle S, [\![\cdot]\!]\rangle$ where S is a (generic) nonempty set of *(abstract) states*, ranged over by s, t, r, and $[\![\cdot]\!]$ is the interpretation of command symbols: for all a, we have $[\![a]\!] : S \to \mathcal{P}(S)$.

Formulæ of μ-calculus may have free propositional variables; therefore, we need to introduce *environments*, which are functions assigning sets of states to propositional variables: $Env \stackrel{\text{def}}{=} Var \to \mathcal{P}(S)$. Given a model $\mathcal{M} = \langle S, [\![\cdot]\!]\rangle$ and an environment ρ, the semantics of a formula is the set of states in which it holds, and it is defined by extending $[\![\cdot]\!]$ compositionally:

$$[\![ff]\!]\rho \stackrel{\text{def}}{=} \emptyset \qquad [\![\varphi \supset \psi]\!]\rho \stackrel{\text{def}}{=} (S \setminus [\![\varphi]\!]\rho) \cup [\![\psi]\!]\rho$$
$$[\![x]\!]\rho \stackrel{\text{def}}{=} \rho(x) \qquad [\![[a]\,\varphi]\!]\rho \stackrel{\text{def}}{=} \{s \in S \mid \forall r \in [\![a]\!]s : r \in [\![\varphi]\!]\rho\}$$
$$[\![\neg\varphi]\!]\rho \stackrel{\text{def}}{=} S \setminus [\![\varphi]\!]\rho \quad [\![\mu x\varphi]\!]\rho \stackrel{\text{def}}{=} \bigcap\{T \subseteq S \mid [\![\varphi]\!]\rho[x \mapsto T] \subseteq T\}$$

It is customary to view a formula φ with a free variable x as defining a function $\varphi^\rho_x : \mathcal{P}(S) \to \mathcal{P}(S)$, such that for all $U \subseteq S$: $\varphi^\rho_x(U) = [\![\varphi]\!]\rho[x \mapsto U]$. The intuitive interpretation of $\mu x\varphi$ is then the *least fixed point* of φ^ρ_x. The syntactic condition on the formation of $\mu x\varphi$ ensures the monotonicity of φ^ρ_x, and hence, by Knaster-Tarski's theorem, the existence of the lfp as well [8]. This does not hold if we drop the condition on the formation of $\mu x\varphi$; e.g., the formula $\neg x$ identifies the function $(\neg x)^\rho_x(T) = S \setminus T$, which is not monotone and has no (least) fixed point.

In order to have a semantical counterpart of the syntactic notion of "deduction", we introduce a consequence relation for the μ-calculus:

Definition 1. *Let $\mathcal{M} = \langle S, [\![\cdot]\!]\rangle$ be a model for μK. The* consequence relation *for μK with respect to $\mathcal{M}$ is a relation $\models_{\mathcal{M}} \subseteq \mathcal{P}_{<\omega}(\Phi) \times \Phi$, defined as follows (where $[\![\Gamma]\!]\rho \stackrel{\text{def}}{=} \bigcap_{\varphi\in\Gamma}[\![\varphi]\!]\rho$):* $\quad \Gamma \models_{\mathcal{M}} \varphi \iff \forall\rho.[\![\Gamma]\!]\rho \subseteq [\![\varphi]\!]\rho\,.$

The (absolute) consequence relation *for μK is: $\Gamma \models \varphi \iff \forall\mathcal{M}.\Gamma \models_{\mathcal{M}} \varphi$.*

$$\frac{}{posin(x, ff)} \quad \frac{y \in Var}{posin(x, y)} \quad \frac{negin(x, \varphi)}{posin(x, \neg\varphi)} \quad \frac{posin(x, \varphi) \quad posin(x, \psi)}{posin(x, \varphi \supset \psi)} \quad \frac{posin(x, \varphi)}{posin(x, [a]\,\varphi)} \quad \frac{\text{for } z \neq x : posin(x, \varphi[z/y])}{posin(x, \mu y \varphi)}$$

$$\frac{}{negin(x, ff)} \quad \frac{y \neq x}{negin(x, y)} \quad \frac{posin(x, \varphi)}{negin(x, \neg\varphi)} \quad \frac{negin(x, \varphi) \quad negin(x, \psi)}{negin(x, \varphi \supset \psi)} \quad \frac{negin(x, \varphi)}{negin(x, [a]\,\varphi)} \quad \frac{\text{for } z \neq x : negin(x, \varphi[z/y])}{negin(x, \mu y \varphi)}$$

Fig. 1. The positivity proof system.

2 A Proof System for the Positivity Condition

Since we aim to encode the μ-calculus in some logical framework, we need to enforce the context-sensitive condition on the formation of formulæ of the form $\mu x\varphi$. That is, we ought to specify in detail the condition of "occurring positive in a formula" for a variable. This notion can be represented by two new judgments on formulæ and variables, *posin* and *negin*, which are derived by means of the rules in Figure 1. Roughly, $posin(x, \varphi)$ holds iff all occurrences of x in φ are positively; dually, $negin(x, \varphi)$ holds iff all occurrences of x in φ are negative. Notice that if x does not occur in φ, then it occurs both positively and negatively. More formally, the notions these auxiliary judgments capture are the following:

Definition 2 ((Anti)Monotonicity). *For $\varphi \in \Phi$, $x \in Var$, we say that φ is* monotone *on x (written $Mon_x(\varphi)$) iff $\forall \mathcal{M}, \forall \rho, \forall U, V \subseteq S$: $U \subseteq V \Longrightarrow \varphi^\rho_x(U) \subseteq \varphi^\rho_x(V)$. We say that φ is* antimonotone *on x (written $AntiMon_x(\varphi)$) iff $\forall \mathcal{M}, \forall \rho, \forall U, V \subseteq S$: $U \subseteq V \Longrightarrow \varphi^\rho_x(U) \supseteq \varphi^\rho_x(V)$.*

These notions refer directly to the semantic structures in which formulæ take meaning. In fact, the syntactic conditions of positivity/negativity are sound wrt the semantic condition of monotonicity/antimonotonicity:

Proposition 1. *$\vdash posin(x, \varphi) \Rightarrow Mon_x(\varphi)$ and $\vdash negin(x, \varphi) \Rightarrow AntiMon_x(\varphi)$.*

The converse of Proposition 1 does not hold. Consider e.g. $\varphi \stackrel{\text{def}}{=} (x \supset x)$: clearly, $[\![\varphi]\!]\rho = S$ always, and hence $(x \supset x)^\rho_x$ is both monotone and antimonotone. However, x does not occur only positively nor only negatively in φ. Hence, we cannot derive $\vdash posin(x, (x \supset x))$ nor $\vdash negin(x, (x \supset x))$. This is generalized in the following result, which can be proved by induction on the syntax of φ:

Proposition 2. *If $x \in \mathrm{FV}(\varphi)$ occurs both positively and negatively in φ then neither $posin(x, \varphi)$ nor $negin(x, \varphi)$ are derivable.*

We can restrict ourselves to only positive formulæ without loss of generality: by Lyndon Theorem [4], every monotone formula is equivalent to a positive one.

3 The Proof System $\mathbf{N}^{ls}_{\mu K}$

Usually, systems for μ-calculus are given in Hilbert style [8]. Here we present $\mathbf{N}^{ls}_{\mu K}$ (Figure 2), a *lazy substitution* proof system in Natural Deduction style for μK. This system is called "lazy" after that substitutions of formulæ for variables are delayed as much as possible—and may be not performed at all.

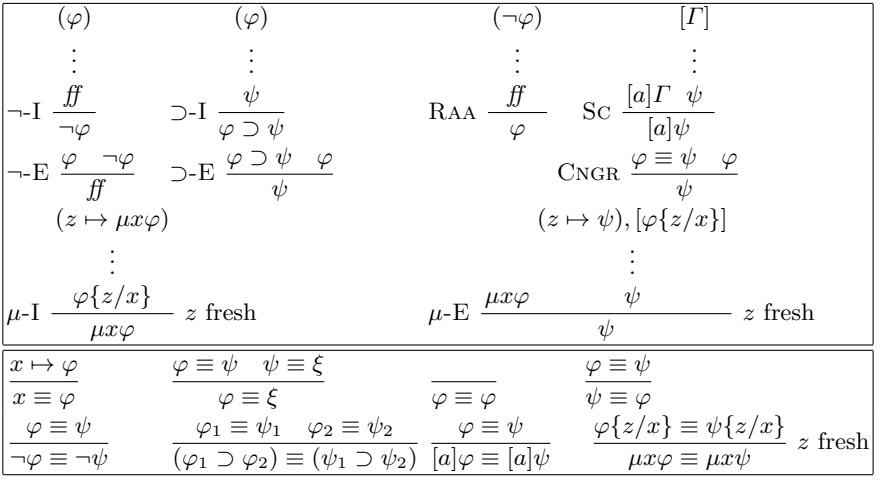

Fig. 2. The lazy substitution, Natural Deduction-style proof system $\mathbf{N}^{ls}_{\mu K}$ for μ-calculus: logical system (top), and congruence system (bottom).

$\mathbf{N}^{ls}_{\mu K}$ is composed by two derivation systems, the *logical* one and the *congruence* one. Roughly, the logical system allows for deriving formulæ from formulæ (*assumptions*) and *bindings*, which are judgments of the form $x \mapsto \varphi$, where $x \in \mathit{Var}$ and $\varphi \in \Phi$. The congruence system allows for deriving judgments of the form $\varphi \equiv \psi$, from a list of bindings. More precisely, we introduce the following

Definition 3. *A* set of assumptions *(denoted by Γ) is any finite set of formulæ; a* binding list *(denoted by Δ) is a list $\langle x_1 \mapsto \varphi_1, \ldots, x_n \mapsto \varphi_n \rangle$ such that for all $i \neq j$: $x_i \neq x_j$, and for all $i \leq j$: $x_i \notin \mathrm{FV}(\varphi_j)$.*

A derivation of φ from assumptions Γ and bindings Δ is denoted by $\Delta; \Gamma \vdash \varphi$; a derivation of $\varphi \equiv \psi$ from Δ is denoted by $\Delta \vdash \varphi \equiv \psi$.

The logical system is composed by a standard set of rules for classical propositional logic, extended by Scott's rule SC for minimal modal logic, the congruence rule CNGR, and the intro/elimination rules μ-I, μ-E. The rules for μ have a direct semantic interpretation: the introduction rule states that (the meaning of) $\mu x\varphi$ is a prefixed point of φ^{ρ}_{x}; the elimination rule states that (the meaning of) $\mu x\varphi$ implies, and then "is less than", any prefixed point of φ^{ρ}_{x}. Therefore, these rules state that (the meaning of) $\mu x\varphi$ is the least fixed point, of φ^{ρ}_{x}.

In rule SC, the square brackets surrounding Γ mean that ψ may depend only on the discharged assumption Γ. Similarly, in rule μ-E, the formula $\varphi\{z/x\}$ is the only assumption that the subderivation of ψ may depend on. These "modal" side conditions can be explicated clearly by a Gentzen-like presentation:

$$\text{SC}\ \frac{\Delta; \Gamma \vdash \psi}{\Delta; [a]\Gamma \vdash [a]\psi} \qquad \mu\text{-E}\ \frac{\Delta; \Gamma \vdash \mu x\varphi \quad \Delta, z \mapsto \psi; \varphi\{z/x\} \vdash \psi}{\Delta; \Gamma \vdash \psi}\ z \text{ fresh}$$

No logical rule requires a binding as a premise; bindings are only discharged, in rules requiring a substitution (i.e., rules μ-I, μ-E). In these rules, variables are

not textually replaced by the corresponding formula, but only by an α-equivalent ("fresh") variable. The discharged hypothesis keeps in the derivation context the binding between the substituted variable and the corresponding formula. These hypotheses form a binding list which is used by the congruence system: roughly, we can prove $\Delta \vdash \varphi \equiv \psi$ iff φ and ψ are the same formula, "up to Δ". More precisely, a binding list Δ corresponds to a particular form of substitution, which can be defined by induction on Δ as $\sigma_{\langle\rangle} \stackrel{\text{def}}{=} \{\}$, $\sigma_{\Delta,x\mapsto\varphi} \stackrel{\text{def}}{=} \sigma_\Delta \circ \{\varphi/x\}$. Then, $\equiv$ is the smallest congruence which contains σ_Δ:

Proposition 3. *For all Δ, for all $\varphi, \psi \in \Phi$: $\Delta \vdash \varphi \equiv \psi \iff \varphi\sigma_\Delta = \psi\sigma_\Delta$*

The resulting system is then sound and complete:

Theorem 1. *For all Δ, for all Γ finite and $\varphi \in \Phi$: $\Delta; \Gamma \vdash \varphi \iff \Gamma\sigma_\Delta \models \varphi\sigma_\Delta$*

Proof. (Sketch) Soundness ($\Rightarrow$) is proved by showing that each rule is sound. Completeness ($\Leftarrow$) can be proved by proving that axioms and rules of a complete Hilbert-style system (e.g., Kozen's one [17]) are derivable in $\mathbf{N}^{ls}_{\mu K}$. □

Corollary 1. *For Γ finite set of formulæ, φ formula: $\emptyset; \Gamma \vdash \varphi \iff \Gamma \models \varphi$.*

4 Encoding the Language of μ-Calculus

The encoding of the language of μ-calculus is quite elaborate. The customary approach, is to define an inductive type, `o:Set`, whose constructors correspond to those of the language of μK. In order to take full advantage of α-conversion and substitution machinery provided by the metalanguage, we adopt the *higher order abstract syntax* [5,7]. In this approach, binding constructors (like μ) are rendered by higher-order term constructors; that is, they take a *function*. The naïve representation of μ would be `mu:(o->o)->o`; however, this solution does not work inside an inductive definition of CIC, because it leads to a non-well-founded definition [2,5].

The second problem is the presence of a context-sensitive condition on the applicability of μ: in order to construct a formula of the form $\mu x\varphi$, we have to make sure that x occurs positively in φ. Inductive types do not support this kind of restriction, since they define only context-free languages [13].

In order to overcome the first problems, we adopt the *bookkeeping* technique [13]. We introduce a separate type, `var`, for the identifiers. These variables act as "placeholders" for formulæ: they will be bound to formulæ in the application of μ-I and μ-E rules, by means of an auxiliary judgment. There are no constructors for type `var`: we only assume that there are infinitely many variables.

```
Parameter var : Set.
Axiom var_nat : (Ex [srj:var->nat](n:nat)(Ex [x:var](srj x)=n)).
```

Then, we define the set of preformulæ of μ-calculus, also those not well formed:

```
Parameter Act : Set.
Inductive o   : Set := ff : o | Not : o -> o | Imp : o -> o -> o
    | Box : Act -> o -> o | Var : var -> o | mu : (var->o) -> o.
```

Notice that, the argument of `mu` is a function of type `var->o`. In general, this may arise *exotic terms*, i.e. terms which do not correspond to any preformula of the μ-calculus [5,13]. These terms are built by using the `Case` term constructor

of inductive type theory, over the type of variables. This cannot be achieved in our approach since `var` is not declared as an inductive set. Of course, the price we pay is that equality between variables is not decidable [13, Section 11.2].

Now, we have to rule out all the non-well-formed formulæ. At the moment, the only way for enforcing in CIC context-sensitive conditions over languages is to define a subtype by means of Σ-types. As a first step, we formalize the system for positivity/negativity presented in Figure 1, introducing two judgments `posin`, `negin` of type `var->o->Prop`. A careful analysis of the proof system (Figure 1) points out that the derivation of these judgments is completely syntax driven. It is therefore natural to define these judgments as *recursively defined functions*, instead of inductively defined propositions. This is indeed feasible, but the rules for the binding operators introduce an implicit quantification over the set of variables *different from the one we are looking for*. This is rendered by assuming a new variable (`y`) and that it is different from the variable `x` (see last cases):

```
Fixpoint posin [x:var;A:o] : Prop :=
 <Prop>Cases A of  ff => True | (Not B) => (negin x B)
 | (Imp A1 A2) => (negin x A1)/\(posin x A2) | (Box a B) => (posin x B)
 | (Var y) => True | (mu F)  => (y:var)~(x=y)->(posin x (F y))
 end
with negin [x:var;A:o] : Prop :=
 <Prop>Cases A of  ff => True | (Not B) => (posin x B)
 | (Imp A1 A2) => (posin x A1)/\(negin x A2) | (Box a B) => (negin x B)
 | (Var y) => ~(x=y) | (mu F)  => (y:var)~(x=y)->(negin x (F y))
 end.
```

Therefore, in general a goal `(posin x A)` can be `Simp`lified (i.e., by applying the `Simpl` tactic, in Coq) to a conjunction of only three forms of propositions: `True`, negations of equalities or implications from negations of equalities to another conjunction of the same form. These three forms are dealt with simply in Coq, hence proving this kind of goals is a simple and straightforward task.

Similarly, a preformula is well formed when every application of μ satisfies the positivity condition:

```
Fixpoint iswf [A:o] : Prop :=
 <Prop>Cases A of  ff => True | (Var y) => True | (Not B) => (iswf B)
 | (Imp A1 A2) => (iswf A1)/\(iswf A2) | (Box a B) => (iswf B)
 | (mu F)  => (x:var)(iswf (F x))/\((notin x (mu F)) -> (posin x (F x)))
 end.
```

In the case of μ, we locally assume the fact that the `x` we introduce does not appear in the formula, i.e. it is *fresh*. Although this is automatically achieved by the metalanguage, we may need this information for proving `(posin x (F x))`. This is achieved by the hypothesis `(notin z (mu F))`. The judgment `notin` and the dual `isin` (see [12]) are auxiliary judgments for occur-checking. Roughly, `(notin x A)` holds iff `x` does not occur free in `A`; dually for `isin`.

Finally, each formula of the μ-calculus is therefore represented by a pair preformula-proof of its well-formedness:

```
Record wfo: Set := mkwfo {prp : o; cnd : (iswf prp)}.
```

In order to estabilish that our encoding is faithful, we introduce the following notation: for $X = \{x_1, \ldots, x_n\} \subset Var$, let $\Xi_X \stackrel{\text{def}}{=} \mathtt{x_1} : \mathtt{var}, \ldots, \mathtt{x_n} : \mathtt{var}$, $\mathtt{o}_X \stackrel{\text{def}}{=}$

$\{\mathtt{t} \mid \Xi_X \vdash \mathtt{t} : \mathtt{o},\ \mathtt{t}\ \text{canonical}\}$ and $\mathtt{wfo}_X \stackrel{\text{def}}{=} \{\mathtt{t} \in \mathtt{o}_X \mid \exists \mathtt{d}.\Xi_X \vdash \mathtt{d} : (\mathtt{iswf\ t})\}$. We can then define the *encoding map* $\varepsilon_X : \Phi_X \to \mathtt{o}_X$, as follows:

$$\begin{array}{ll} \varepsilon_X(x) = \mathtt{x} & \varepsilon_X(\varphi \supset \psi) = \mathtt{(Imp}\ \varepsilon_X(\varphi)\ \varepsilon_X(\psi)\mathtt{)} \\ \varepsilon_X(\neg\varphi) = \mathtt{(Not}\ \varepsilon_X(\varphi)\mathtt{)} & \varepsilon_X([a]\varphi) = \mathtt{(Box\ a}\ \varepsilon_X(\varphi)\mathtt{)} \\ \varepsilon_X(\mathit{ff}) = \mathtt{ff} & \varepsilon_X(\mu x\varphi) = \mathtt{(mu\ [x:var]}\varepsilon_{X,x}(\varphi)\mathtt{)} \end{array}$$

Theorem 2. *The map ε_X is a compositional bijection between Φ_X and $\mathtt{wfo}_X$.*
Proof. (Sketch) Long inductions. First, one proves that `posin`, `negin` adequately represent the positivity/negativity proof system. Then, a preformula φ is a formula iff each application of μ is valid, iff for each application of μ there exists a (unique) witness of `posin`, iff there exists an inhabitant of $(\mathtt{iswf}\ \varepsilon_X(\varphi))$. □

5 Encoding the Proof System $\mathbf{N}^{ls}_{\mu K}$

In the encoding paradigm of Logical Frameworks, a proof system is usually represented by introducing a *proving judgment* over the set of formulæ, like `T:o -> Prop`. A type `(T phi)` should be intended, therefore, as "φ is true;" any term which inhabits `(T phi)` is a witness (a proof) that φ is true. Each rule is then represented by a type constructor of `T`. Moreover, substitution schemata for binding operators need not to be implemented "by hand", because they are inherited from the metalanguage. This is the case, for instance, of "$\forall$" in First Order Logic; for further examples and discussion, we refer to [5,7].

However, in representing the proof system $\mathbf{N}^{ls}_{\mu K}$, two difficult issues arise: the encoding of proof rules, like Sc and μ-E, and the substitution of formulæ for variables in rules μ-I and μ-E. Moreover, Scott's rule is parametric in the number of assumptions which have to be "boxed". These issues escape the standard encoding paradigm, so we have to accommodate some special technique.

Actually, in the underlying theory of CIC there is no direct way for enforcing on a premise the condition that it is a theorem (i.e. that it depends on no assumptions) or, more generally, that a formula depends only on a given set of assumptions. This is because the typing rules of PTS's are strictly in Natural Deduction style. Therefore, in presence of sequent-style rules like Sc and μ-E, one could encode a complete sequent calculus introducing the type `olist` of lists of formulæ, the sequent judgment `Seq:olist->o->Prop`, and all the machinery of Gentzen's original system [6]. This would lead to an unusable proof system: even if our rules have a Natural Deduction flavour, all the goals would be crammed with the list of hypotheses, and we should deal with supplementary structural rules for manipulating the list of assumptions.

Instead, we represent more efficiently the assumption set by means of the proof context provided by CIC, i.e., by taking advantage of the possibility of reasoning "under assumptions" [1]. First, we represent $\mapsto$ and $\equiv$ by means of two judgments `bind:var->o->Prop` and `cngr:o->o->Prop`, respectively. The former has no constructor (it declared as a `Parameter`), while the latter is rendered as an inductive predicate, as expected. In particular, the congruence rule for μ is rendered by means of a locally quantified variable (see [12] for the whole listing):

```
Parameter bind : var -> o -> Prop.
Inductive cngr : o -> o -> Prop :=
```

```
  cngr_bind : (x:var)(A:o)(bind x A) -> (cngr (Var x) A)
  (... other rules ...)
| cngr_mu :(A,B:var->o)((x:var)(cngr (A x) (B x)))->(cngr (mu A) (mu B)).
```

Then, we introduce the basic proving judgment, `T:U->o->Prop`, where `U` a set with *no* constructors. Elements of `U` will be called *worlds* for suggestive reasons. Each pure rule (i.e., with no side condition), is parameterized over a generic world, like the following:

```
Axiom Imp_E : (w:U)(A,B:o)(T w (Imp A B)) -> (T w A) -> (T w B).
```

Therefore, in a given world all the classical rules apply as usual. It should be noticed, however, that we require a locally introduced formula to be well formed. This is the case of $\supset$-I:

```
Axiom Imp_I : (w:U)(A,B:o)(iswf A)->((T w A)->(T w B))->(T w (Imp A B)).
```

Indeed, it can be shown that if we allow for non-well formed formulæ in these "negative positions," we get easily an inconsistent derivation.

Proof rules, on the other hand, are distinguished by *local* quantifications of the world parameter, in order to make explicit the dependency between a conclusion and its premises. The rule μ-E is encoded as follows:

```
Axiom mu_E   : (F:var->o)(iswf A) ->
  ((z:var)(notin z (mu F)) -> (bind z A) -> (w':U)(T w' (F z))->(T w' A))
  -> (w:U)(T w (mu F)) -> (T w A).
```

The idea behind the use of the extra parameter is that in making an assumption, we are forced to assume the existence of a world, say `w`, and to instantiate the judgment `T` also on `w`. This judgment then appears as an hypothesis on `w`. Hence, deriving as premise a judgment, which is universally quantified with respect to `U`, amounts to establishing the judgment for the generic world `w'` on which only the given assumptions are made, i.e. on the given assumptions.

This idea can be suitably generalized to take care of an unlimited number of assumptions. In fact, a generic sequent $\varphi_1, \ldots, \varphi_n \vdash \varphi$ is faithfully represented by the type `(w:U)(T w A1)->...->(T w An)->(T w A)` where $\mathtt{A}_i = \varepsilon_X(\varphi_i)$ and $\mathtt{A} = \varepsilon_X(\varphi)$. The locally quantified world `w` forces any proof of `(T w A)` to depend only on the given assumptions. The problem is capturing the parametric flavour expressed by the "...". At this end, we introduce lists of formulæ and the auxiliary function `Sequent:U->o->olist->Prop`:

```
Inductive olist : Set := nil : olist | cons : o -> olist -> olist.
Fixpoint Sequent [w:U;B:o;l:olist] : Prop :=
   Cases l of
     nil => (T w B)   | (cons A t) => (T w A)->(Sequent w B t)
   end.
```

Therefore, the aforementioned representation of $\varphi_1, \ldots, \varphi_n \vdash \varphi$ is denoted by `(w:U)(Sequent w B G)` where `G` is the list composed by $\mathtt{A}_1, \ldots, \mathtt{A}_n$. In fact, `(Sequent w B G)` is exactly $\beta\iota\delta$-equivalent (it can be reduced) to `(T w A1)->...->(T w An)->(T w B)`. We can therefore represent Scott's rule as follows:

```
Fixpoint Boxlist [a:Act; l:olist] : olist :=
Cases l of nil => nil | (cons B t) => (cons (Box a B) (Boxlist a t)) end.
Axiom Sc  : (G:olist)(B:o)(a:Act) ((w':U)(Sequent w' B G)) ->
            (w:U)(Sequent w (Box a B) (Boxlist a G)).
```

where the map `Boxlist:Act->olist->olist` represents exactly the "$[a]\Gamma$" notation of rule Sc. Hence, we can use the conversion tactics provided by Coq for automatically converting applications of `Sequent` to the right proposition.

The encoding of μ-E (and μ-I) uses also the auxiliary judgment `bind`. Following the idea of $\mathbf{N}^{ls}_{\mu K}$, the context $\varphi(\cdot)$ of $\mu x\varphi(x)$ is filled by a fresh (i.e., locally quantified) variable `z`. The binding between `z` and the corresponding formula is kept in the derivation environment by the hypothesis `(bind z A)`. This hypothesis can be used in the derivation of congruence judgments, for replacing formulæ only when it is needed. For an example, see [12].

The discharged hypothesis `(notin z (mu F))` in rule `mu_E` reflects at the logical level, the fact that `z` is fresh. Although freshness of `z` obviously holds, it cannot be inferred *in* the system because it belongs to the metalevel of the system. Hence, we reify it by means of the discharged hypothesis, which may be needed in the rest of derivation for inferring well-formedness of discharged formulæ in rules Raa, $\supset$-I, $\neg$-I.

In order to state the adequacy of our formalization w.r.t. $\mathbf{N}^{ls}_{\mu K}$, we introduce the following notation. Let $X \subset \mathit{Var}$ be finite, and $\varphi_1, \ldots, \varphi_n, \varphi \in \Phi_X$; then, for $x_1, \ldots, x_n \in X$, we denote by $\delta_X(x_1 \mapsto \varphi_1, \ldots, x_n \mapsto \varphi_n)$ the context $\mathtt{b}_1\colon (\mathtt{bind}\ \mathtt{x}_1\ \varepsilon_X(\varphi_1)), \ldots \mathtt{b}_n\colon (\mathtt{bind}\ \mathtt{x}_n\ \varepsilon_X(\varphi_n))$, and, for `w:U`, we denote by $\gamma_{X,\mathtt{w}}(\varphi_1, \ldots, \varphi_n)$ the context $\mathtt{h}_1\colon (\mathtt{T}\ \mathtt{w}\ \varepsilon_X(\varphi_1)), \ldots, \mathtt{h}_n\colon (\mathtt{T}\ \mathtt{w}\ \varepsilon_X(\varphi_n))$.

Theorem 3. *Let $X \subset \mathit{Var}$ be finite, Δ a binding list such that* $\mathrm{FV}(\Delta) \subseteq X$, *and $\Gamma \subset \Phi_X$ finite. Then, for all $\varphi_1, \varphi_2 \in \Phi_X$:*

1. *$\Delta \vdash \varphi_1 \equiv \varphi_2$ iff there is* `t` *such that $\Xi_X, \delta_X(\Delta) \vdash \mathtt{t} : (\mathtt{cngr}\ \varepsilon_X(\varphi_1)\ \varepsilon_X(\varphi_2))$*
2. *$\Delta; \Gamma \vdash \varphi_1$ iff there is* `t` *such that $\Xi_X, \delta_X(\Delta), \mathtt{w} : \mathtt{U}, \gamma_{X,\mathtt{w}}(\Gamma) \vdash \mathtt{t} : (\mathtt{T}\ \mathtt{w}\ \varepsilon_X(\varphi_1))$.*

Proof. (Sketch) Directions $\Rightarrow$ are proved by induction on the proofs of $\Delta \vdash \varphi_1 \equiv \varphi_2$ and $\Delta; \Gamma \vdash \varphi_1$. Directions $\Leftarrow$ are proved by induction on the syntax of `t`: each constructor of `(cngr A B)` and `(T w A)` corresponds to a rule of $\mathbf{N}^{ls}_{\mu K}$. □

6 Conclusions

In this paper we have introduced an original proof system $\mathbf{N}^{ls}_{\mu K}$ for the propositional modal μ-calculus, and its formalization in the Calculus of Inductive Constructions. Beside the formalization, $\mathbf{N}^{ls}_{\mu K}$ is interesting on its own for several reasons: it is in Natural Deduction style, it has been proved complete with respect to logical consequences (while traditional Hilbert-style proof systems are complete with respect to theorems), and its usage should be easier than axiomatic proof systems. Moreover, in $\mathbf{N}^{ls}_{\mu K}$ substitutions of formulæ for variables are not always performed, but they may be delayed until actually needed.

In the encoding, we have addressed several problematic issues. First, the use of the higher order abstract syntax frees us from a tedious encoding of the mechanisms involved in the handling of α-conversion, because it is automatically inherited from the metalevel. Secondly, substitution is represented by a congruence proof system, whose proofs are syntax-driven and can be highly automatized in the Coq environment. Thirdly, we have faithfully represented the context-sensitive language of μ-calculus by formalizing the notion of "well-formed formula." Finally, the modal nature of impure rules of μ-calculus (Sc and μ-E)

has been effectively rendered, although Logical Frameworks do not support directly modal rules. The techniques we have adopted can be readily ported to other formalisms featuring similar problematic issues.

From a proof-theoretical point of view, rule Sc is not satisfactory, since it breaks the typical introduction/elimination pattern of Natural Deduction systems. Whether there is a truly Natural Deduction formulation of the system remains an open question.

The implementation of substitution by means of an environment of bindings has been previously investigated in the context of logic programming by Miller [14], and in that of model checking by Stirling and Walker [16]. This latter fact deserves further investigation. For instance, $\mathbf{N}^{ls}_{\mu K}$ could be integrated with a model checker in a simple and efficient way; moreover, the model checker could be implemented in Coq, and its correctness formally verified.

Acknowledgements. The author is grateful to Furio Honsell and an anonymous referee for many useful remarks.

References

[1] A. Avron, F. Honsell, M. Miculan, and C. Paravano. Encoding modal logics in Logical Frameworks. *Studia Logica*, 60(1):161–208, Jan. 1998.

[2] *The Coq Proof Assistant Reference Manual - Version 6.2*. INRIA, Rocquencourt, May 1998. Available at `ftp://ftp.inria.fr/INRIA/coq/V6.2/doc`.

[3] A. Church. A formulation of the simple theory of types. *JSL*, 5:56–68, 1940.

[4] G. D'Agostino, M. Hollenberg. Logical questions concerning the μ-calculus: interpolation, Lyndon, and Łoś-Tarski. To be published in the *JSL*, 1999.

[5] J. Despeyroux, A. Felty, and A. Hirschowitz. Higher-order syntax in Coq. In *Proc. of TLCA'95*. LNCS 902, Springer-Verlag, 1995.

[6] G. Gentzen. Investigations into logical deduction. In M. Szabo, ed., *The collected papers of Gerhard Gentzen*, pages 68–131. North Holland, 1969.

[7] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, Jan. 1993.

[8] D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983.

[9] Z. Luo, R. Pollack, *et al. The LEGO proof assistant*. Department of Computer Science, University of Edinburgh. `http://www.dcs.ed.ac.uk/home/lego`

[10] L. Magnusson and B. Nordström. The ALF proof editor and its proof engine. In *Proc. of TYPES'93*, LNCS 806, pages 213–237. Springer-Verlag, 1994.

[11] P. Martin-Löf. On the meaning of the logical constants and the justifications of the logic laws. TR 2, Dip. di Matematica, Università di Siena, 1985.

[12] M. Miculan. Encoding the μ-calculus in Coq. Available at `http://www.dimi.uniud.it/~miculan/mucalculus`

[13] M. Miculan. *Encoding Logical Theories of Programs*. PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy, Mar. 1997.

[14] D. Miller. Unification of Simply Typed Lambda-Terms as Logic Programming. In *Proc. of Eighth Intl. Conf. on Logic Programming*. MIT, 1991.

[15] B. Nordström, K. Petersson, and J. M. Smith. Martin-Löf's type theory. In *Handbook of Logic in Computer Science*. Oxford University Press, 1992.

[16] C. Stirling, and D. Walker. Local model checking in the modal μ-calculus. *Theoretical Computer Science*, 89:161–177, 1983.

[17] I. Walukiewicz. Completeness of Kozen's axiomatisation. In D. Kozen, editor, *Proc. 10th LICS*, pages 14–24, June 1995. IEEE.