On the formalization of the modal μ-calculus in the Calculus of Inductive Constructions[1]

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

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 formalization 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 modal μ-calculus. The techniques we adopt can be readily ported to other languages and proof systems featuring similar problematic issues.

Key Words: μ-calculus, modal and temporal logics, type theory, formal methods.

INTRODUCTION

We present a Natural Deduction proof system for Kozen's propositional modal μ-calculus and its formalization in the Calculus of Inductive Constructions.

The modal 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 the finite model property and decidability. Therefore, the modal μ-calculus is an ideal candidate as a logic for the verification of processes. Although Walukiewicz [31] proved the completeness of Kozen's original system [17], its applicability to non trivial cases is limited by long, difficult, error-prone proofs. Moreover, the problem of validity of a formula is EXPTIME-complete; hence, a fully-automatized theorem prover fo μK may be computationally very expensive.

These drawbacks can be (partially) overcome by supplying the user with a *computer-aided proof environment*, that is, a system in which he can represent (*encode, for-*

[1] Work partially supported by Italian MURST-97 grant *Tecniche formali....* A preliminar version has appeared in the *Proceedings of ICALP'99.*

malize) 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.

However, the implementation of a proof environment for a specific formal system is a complex, time-consuming, and daunting task. The environment should provide tools for checking previously hand-made proofs; developing interactively, step-by-step, error-free proofs from scratch; reusing previously proved properties; even, deriving properties automatically, when feasible, freeing the user from most unpleasant and error-prone steps.

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.

Several different frameworks have been proposed, implemented and applied to many formal systems. *Type theories* have emerged as leading candidates for Logical Frameworks. Simple typed λ-calculus and minimal intuitionistic propositional logic are connected by the well-known *proposition-as-types* paradigm [6]. 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 [13, 2, 5, 23]. 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 *judgements-as-types, proofs-as-λ-terms* paradigm [13]. The key concept is that of *hypothetico-general* judgement [20], which is rendered as a type of the dependent typed λ-calculus of the Logical Framework. With this interpretation, a judgement is viewed as a type whose inhabitants correspond to proofs of the judgement.

It is worthwhile noticing that Logical Frameworks based on type theory directly give rise to proof systems in *Natural Deduction style* in the sense of [1, 12]. 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 [5, 18, 19]. 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, judgements, 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 formalization.

This paper is part of an ongoing research programme at the Computer Science Department of the University of Udine on proof editors, started in 1992, based on HOAS encodings in dependent typed λ-calculus for program logics [15, 21, 16]. In this paper, we investigate the applicability of this approach to the modal μ-calculus. Due to its expressive power, we adopt the Calculus of Inductive Constructions (CIC), implemented in the system Coq. Beside its importance in the theory and verification of processes, the modal μ-calculus is interesting also for its syntactic and proof theoretic peculiarities. These idiosyncrasies are mainly due to a) the negative arity of "μ" (i.e., the bound variable x ranges over the same syntactic class of $\mu x\varphi$); b) a context-sensitive grammar due the condition on $\mu x\varphi$; c) 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 $\mathbf{N}^{\lhd}_{\mu K}$ in Natural Deduction style for μK. This system should be 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 perfectly the stack discipline of assumptions of Natural Deduction, and it is neatly formalized in CIC.

Beside these practical and theoretical motivations, this work can give insights in the expressive power of CIC and Coq. Indeed, the formalization techniques we will adopt take full advantage of pragmatic features offered by Coq, such as the automatic simplification of terms, in order to simplify as much as possible the task of proof development. Moreover, it is interesting to notice how in the formalization process, the need for an easy-to-use system leads us to some theoretical considerations on $\mathbf{N}^{\lhd}_{\mu K}$ itself.

Synopsis. In Section 1, we recall the syntax and the semantics of μK. In Section 2 we present a proof system which captures the syntactic condition on the formation of $\mu x\varphi$. The proof system $\mathbf{N}^{\lhd}_{\mu K}$ is introduced in Section 3. Section 4 gives a brief insight into the Calculus of Inductive Constructions (CIC). In Section 5 we discuss the formalization of μK in CIC and some theoretical issues on $\mathbf{N}^{\lhd}_{\mu K}$ arisen by the formalization itself. Final comments and suggestions for future work are reported in Section 6. Longer listings of Coq code are reported in Appendix.

1. SYNTAX AND SEMANTICS OF THE MODAL μ-CALCULUS

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); the syntax of the modal μ-calculus on *Act* is:

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

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 $X \subseteq \mathit{Var}$ 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 modal μ-calculus comes from Modal Logic. A model for the modal 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 modal μ-calculus may have free propositional variables; therefore, we need to introduce *environments*, which are functions assigning sets of states to propositional variables: $\mathit{Env} \stackrel{\text{def}}{=} \mathit{Var} \to \mathcal{P}(S)$. 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, defined compositionally:

$$\begin{array}{rlrl}
[\![\mathit{ff}]\!]\rho & \stackrel{\text{def}}{=} \emptyset & [\![\varphi \supset \psi]\!]\rho & \stackrel{\text{def}}{=} (S \setminus [\![\varphi]\!]\rho) \cup [\![\psi]\!]\rho \\
[\![x]\!]\rho & \stackrel{\text{def}}{=} \rho(x) & [\![[a]\,\varphi]\!]\rho & \stackrel{\text{def}}{=} \{s \in S \mid [\![a]\!]s \subseteq [\![\varphi]\!]\rho\} \\
[\![\neg\varphi]\!]\rho & \stackrel{\text{def}}{=} S \setminus [\![\varphi]\!]\rho & [\![\mu x\varphi]\!]\rho & \stackrel{\text{def}}{=} \bigcap\{T \subseteq S \mid [\![\varphi]\!]\rho[x \mapsto T] \subseteq T\}
\end{array}$$

The intuitive meaning of $[a]\varphi$ is that "φ holds in every state we reach after the execution of a." It is customary to view a formula φ with a free variable x as defining a function $\varphi_x^\rho : \mathcal{P}(S) \to \mathcal{P}(S)$, such that for all $U \subseteq S$: $\varphi_x^\rho(U) = [\![\varphi]\!]\rho[x \mapsto U]$. The intuitive interpretation of $\mu x\varphi$ is then the *least fixed point* of φ_x^ρ. The syntactic condition on the formation of $\mu x\varphi$ ensures the monotonicity of φ_x^ρ, and hence, by Knaster-Tarski's theorem, the existence of the lfp as well [17]. 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 $(\neg x)_x^\rho(T) = S \setminus T$, which is not monotone and has no (least) fixed point.

2. A PROOF SYSTEM FOR THE POSITIVITY CONDITION

Since we aim at encoding the modal 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 judgements on formulæ and variables, *posin* and *negin*, which are derived by means of the rules in Figure 1. Roughly, $\mathit{posin}(x, \varphi)$ holds iff all occurrences of x in φ are positive; dually, $\mathit{negin}(x, \varphi)$ 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 judgements refer to are the following:

Definition 2.1. [(Anti)Monotonicity] For $\varphi \in \Phi$, $x \in \mathit{Var}$, we say that

1. φ is *monotone* on x (written $\mathit{Mon}_x(\varphi)$) if and only if $\forall\mathcal{M}, \forall\rho, \forall U, V \subseteq S$: $U \subseteq V \Longrightarrow \varphi_x^\rho(U) \subseteq \varphi_x^\rho(V)$;

$$\begin{array}{|c|c|}
\hline
\dfrac{}{\mathit{posin}(x,\mathit{ff})} \quad \dfrac{y \in \mathit{Var}}{\mathit{posin}(x,y)} & \dfrac{}{\mathit{negin}(x,\mathit{ff})} \quad \dfrac{y \neq x}{\mathit{negin}(x,y)} \\
\dfrac{\mathit{posin}(x,\varphi)}{\mathit{posin}(x,[a]\,\varphi)} \quad \dfrac{\mathit{negin}(x,\varphi)}{\mathit{posin}(x,\neg\varphi)} & \dfrac{\mathit{negin}(x,\varphi)}{\mathit{negin}(x,[a]\,\varphi)} \quad \dfrac{\mathit{posin}(x,\varphi)}{\mathit{negin}(x,\neg\varphi)} \\
\dfrac{\mathit{negin}(x,\varphi) \quad \mathit{posin}(x,\psi)}{\mathit{posin}(x,\varphi \supset \psi)} & \dfrac{\mathit{posin}(x,\varphi) \quad \mathit{negin}(x,\psi)}{\mathit{negin}(x,\varphi \supset \psi)} \\
\dfrac{\text{for } z \neq x : \mathit{posin}(x,\varphi\{z/y\})}{\mathit{posin}(x,\mu y\varphi)} & \dfrac{\text{for } z \neq x : \mathit{negin}(x,\varphi\{z/y\})}{\mathit{negin}(x,\mu y\varphi)} \\
\hline
\end{array}$$

FIG. 1. The positivity proof system.

2. φ is *antimonotone* on x (written $\mathit{AntiMon}_x(\varphi)$) if and only if $\forall \mathcal{M}, \forall \rho, \forall U, V \subseteq S$: $U \subseteq V \Longrightarrow \varphi_x^\rho(U) \supseteq \varphi_x^\rho(V)$.

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

Proposition 2.1. *For all φ formula and x variable: $\vdash \mathit{posin}(x,\varphi) \Rightarrow \mathit{Mon}_x(\varphi)$ and $\vdash \mathit{negin}(x,\varphi) \Rightarrow \mathit{AntiMon}_x(\varphi)$.*

The converse of Proposition 2.1 does not hold. Consider e.g. $\varphi \stackrel{\text{def}}{=} (x \supset x)$: clearly, $[\![\varphi]\!]\rho = S$ always, and hence $(x \supset x)_x^\rho$ is both monotone and antimonotone. However, x does not occur only positively nor only negatively in φ. Hence, we cannot derive $\vdash \mathit{posin}(x,(x \supset x))$ nor $\vdash \mathit{negin}(x,(x \supset x))$. This result can be generalized as follows:

Proposition 2.2. *If $x \in \mathrm{FV}(\varphi)$ occurs both positively and negatively in φ then neither $\mathit{posin}(x,\varphi)$ nor $\mathit{negin}(x,\varphi)$ are derivable.*

Proof. By induction on the syntax of φ. ■

However, we can restrict ourselves to only positive formulæ without loss of generality: by Lyndon Theorem [7], every monotone formula is equivalent to a positive one.

3. THE PROOF SYSTEM $\mathbf{N}_{\mu K}^{\lhd}$

Usually, systems for modal μ-calculus are given in Hilbert style [17, 27]. In this section we present $\mathbf{N}_{\mu K}^{\lhd}$, a *lazy substitution* proof system in Natural Deduction style for μK. As we will see in Section 5, in order to deal with the negative constructor "μ" we have to change the standard encoding paradigm of Logical Frameworks. As a consequence, we can no longer delegate substitution to the machinery of the logical framework; hence, it has to be implemented "manually", at some extent. The system we present in this section aims at minimizing these explicit substitutions,

$$\supset\text{-I}\ \frac{\begin{array}{c}(\varphi)\\ \vdots\\ \psi\end{array}}{\varphi\supset\psi}\qquad \supset\text{-E}\ \frac{\varphi\supset\psi\quad\varphi}{\psi}\qquad \neg\text{-I}\ \frac{\begin{array}{c}(\varphi)\\ \vdots\\ f\!f\end{array}}{\neg\varphi}\qquad \neg\text{-E}\ \frac{\varphi\quad\neg\varphi}{f\!f}$$

$$\textsc{Raa}\ \frac{\begin{array}{c}(\neg\varphi)\\ \vdots\\ f\!f\end{array}}{\varphi}\qquad \textsc{Sc}\ \frac{\begin{array}{cc} & [\Gamma]\\ & \vdots\\ [a]\Gamma & \psi\end{array}}{[a]\psi}\qquad \trianglelefteq\text{-R}\ \frac{\varphi\quad\varphi\trianglelefteq\psi}{\psi}\qquad \trianglelefteq\text{-L}\ \frac{\psi\quad\varphi\trianglelefteq\psi}{\varphi}$$

$$\mu\text{-I}\ \frac{\begin{array}{c}(z\mapsto\mu x\varphi)\\ \vdots\\ \varphi\{z/x\}\end{array}}{\mu x\varphi}\ z\text{ fresh}\qquad \mu\text{-E}\ \frac{\mu x\varphi\qquad\begin{array}{c}(z\mapsto\psi),[\varphi\{z/x\}]\\ \vdots\\ \psi\end{array}}{\psi}\ z\text{ fresh}$$

$$\frac{}{\varphi\trianglelefteq\varphi}\qquad \frac{x\mapsto\psi\quad\psi\trianglelefteq\xi}{x\trianglelefteq\xi}\qquad \frac{\varphi\trianglelefteq\psi}{\neg\varphi\trianglelefteq\neg\psi}$$

$$\frac{\varphi\trianglelefteq\psi}{[a]\varphi\trianglelefteq[a]\psi}\qquad \frac{\varphi_1\trianglelefteq\psi_1\quad\varphi_2\trianglelefteq\psi_2}{(\varphi_1\supset\varphi_2)\trianglelefteq(\psi_1\supset\psi_2)}\qquad \frac{\varphi\{z/x\}\trianglelefteq\psi\{z/x\}}{\mu x\varphi\trianglelefteq\mu x\psi}\ z\text{ fresh}$$

FIG. 2. The Natural Deduction-style proof system $\mathbf{N}^{\trianglelefteq}_{\mu K}$ for modal μ-calculus: logical system (top), and expansion system (bottom).

by taking full advantage of the possibility of reasoning "under assumptions". 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.

$\mathbf{N}^{\trianglelefteq}_{\mu K}$ is composed by two derivation systems, the *logical* one and the *expansion* one (Figure 2). Roughly, the logical system allows for deriving formulæ from formulæ (*assumptions*) and *bindings*, which are judgements of the form $x \mapsto \varphi$, where $x \in \mathit{Var}$ and $\varphi \in \Phi$. The expansion system allows for deriving judgements of the form $\varphi \trianglelefteq \psi$ (reads "φ expands to ψ", or "ψ is an expansion of φ"), from a list of bindings.

Definition 3.1. 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 $i \leq j$: $x_j \notin \mathrm{FV}(\varphi_i)$.

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

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 rules $\trianglelefteq$-R and $\trianglelefteq$-L, and the intro/elimination rules μ-I, μ-E.

Rules $\trianglelefteq$-R, $\trianglelefteq$-L state that a formula is logically equivalent to any of its expansions. In a sequent-style formulation like the following, these rules can be seen as the "right" and "left" expansions, respectively—hence the names:

$$\trianglelefteq\text{-R}\ \frac{\Delta;\Gamma\vdash\varphi}{\Delta;\Gamma\vdash\psi}\,\varphi\trianglelefteq\psi \qquad \trianglelefteq\text{-L}\ \frac{\Delta;\Gamma,\varphi\vdash\xi}{\Delta;\Gamma,\psi\vdash\xi}\,\varphi\trianglelefteq\psi$$

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 minimum prefixed point, i.e. 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 in 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 sequent-style presentation:

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

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 expansion system: roughly, we can prove $\Delta\vdash\varphi\trianglelefteq\psi$ iff φ and ψ are the same formula, "up to Δ". This is made precise by the following proposition:

Proposition 3.1. *Let Δ be a binding list; then, the relation $R=\{\langle\varphi,\psi\rangle\mid\Delta\vdash\varphi\trianglelefteq\psi\}$ is the smallest precongruence which contains Δ.*

Proof. Clearly, R is reflexive and compositional on the syntax of formulæ. For all $x\mapsto\varphi\in\Delta$, it is $R(x,\varphi)$ because of the "variable replacement rule" $\frac{x\mapsto\varphi \quad \varphi\trianglelefteq\varphi}{x\trianglelefteq\varphi}$. It remains to prove that R is transitive. More precisely, we prove that given two derivations $\Pi_1:\Gamma\vdash\varphi\trianglelefteq\psi$, $\Pi_2:\Gamma\vdash\psi\trianglelefteq\xi$, we can build a derivation $\Pi:\Gamma\vdash\varphi\trianglelefteq\xi$. The proof is by induction on the depth of Π_1 and Π_2.
Base case: let Π_1 (respectively, Π_2) be an application of the reflexivity rule. Then $\varphi=\psi$ (respectively, $\psi=\xi$), hence $\Pi=\Pi_2$ (respectively, $\Pi=\Pi_1$).
Inductive case: by cases on the last rule applied in Π_1.

If it is a congruence rule, say the one for negation, then $\varphi=\neg\varphi_1$, $\psi=\neg\psi_1$ for some φ_1,ψ_1, and there is a subderivation $\Pi_{11}:\Delta\vdash\varphi_1\trianglelefteq\psi_1$. By inspection, also Π_2 must end with the same congruence rule, hence $\xi=\neg\xi_1$ for some ξ_1 and there is a subderivation $\Pi_{21}:\Delta\vdash\psi_1\trianglelefteq\xi_1$. By inductive hypothesis, there exists a derivation $\Pi':\Delta\vdash\varphi_1\trianglelefteq\xi_1$, and then by applying the congruence rule for negation, we obtain $\Pi:\Delta\vdash\neg\varphi_1\trianglelefteq\neg\xi_1$.

Otherwise, the last rule applied is $\frac{x\mapsto\varphi_1 \quad \varphi_1\trianglelefteq\psi}{x\trianglelefteq\psi}$, for some x and subderivation $\Pi_{11}:\Delta\vdash\varphi_1\trianglelefteq\psi$. By inductive hypothesis there exists $\Pi':\Delta\vdash\varphi_1\trianglelefteq\xi$, and then by applying the above mentioned rule, we obtain the thesis. ■

Before proving soundness and completeness of $\mathbf{N}^{\trianglelefteq}_{\mu K}$, we need some technical definitions and results.

Definition 3.2. Let $\mathcal{M} = \langle S, [\![\cdot]\!]\rangle$ be a model for μK, and let Δ be a binding list. An environment ρ in $\mathcal{M}$ *agrees with* Δ (written $\rho \models \Delta$) if for all bindings $x \mapsto \varphi$ in Δ: $\rho(x) = [\![\varphi]\!]\rho$.

For any environment ρ, we denote by ρ_Δ the environment defined as follows:

$$\rho_{\langle\rangle} \stackrel{\text{def}}{=} \rho \qquad\qquad \rho_{\Delta,x\mapsto\varphi} \stackrel{\text{def}}{=} \rho_\Delta[x \mapsto [\![\varphi]\!]\rho_\Delta]$$

The "agreement" operation $(\cdot)_\Delta$ enjoys the following property:

Proposition 3.2. *For all ρ environment and for all Δ binding list:*

1.$\rho_\Delta \models \Delta$*;*
2.*if $\rho \models \Delta$ then $\rho_\Delta = \rho$.*
3.$(\cdot)_\Delta$ *is idempotent, that is $(\rho_\Delta)_\Delta = \rho_\Delta$;*

Proof. Points (1.) and (2.) are trivially proved by induction on the length of Δ. Point (3.) follows from (1.) and (2.). ■

The following result can be seen as our counterpart of the classical "substitution lemma".

Lemma 3.1 (Expansion Lemma). *For Δ binding list and φ formula, there exists a formula φ_Δ such that $\Delta \vdash \varphi \trianglelefteq \psi_\Delta$ and for all $\mathcal{M}$, and ρ environment in $\mathcal{M}$: $[\![\varphi]\!]\rho_\Delta = [\![\varphi_\Delta]\!]\rho$.*

Proof. Let us define $\varphi_\emptyset \stackrel{\text{def}}{=} \varphi$, and $\varphi_{\Delta,x\mapsto\psi} \stackrel{\text{def}}{=} (\varphi\{\psi/x\})_\Delta$. Clearly, $\mathrm{FV}(\varphi_\Delta) \cap \mathrm{dom}(\Delta) = \emptyset$ by definition of binding list. The derivation $\Delta \vdash \varphi \trianglelefteq \varphi_\Delta$ can be built by induction on the syntax of φ and on the length of Δ: we apply the suitable structural rule until we reach a variable x such that there is a binding $(x \mapsto \psi) \in \Delta$. In this case, we have to prove that $\Delta \vdash x \trianglelefteq x_\Delta$, i.e., $\Delta \vdash x \trianglelefteq \psi_\Delta$. In fact, Δ can be decomposed in $\Delta', x \mapsto \psi, \Delta''$, and $\mathrm{FV}(\psi) \cap \mathrm{dom}(\Delta) = \mathrm{FV}(\psi) \cap \mathrm{dom}(\Delta')$ by definition of binding list. Therefore, by inductive hypothesis there is a derivation $\Delta' \vdash \psi \trianglelefteq \psi_{\Delta'}$, which is also $\Delta \vdash \psi \trianglelefteq \psi_\Delta$ because $\mathrm{FV}(\psi) \cap \mathrm{dom}(\Delta'') = \emptyset$. Hence, $\Delta \vdash x \trianglelefteq \psi_\Delta$ by applying the rule $\frac{x\mapsto\psi \quad \psi\trianglelefteq\psi_\Delta}{x\trianglelefteq\psi_\Delta}$.

It remains to prove that $\forall\mathcal{M}, \forall\rho. [\![\varphi]\!]\rho_\Delta = [\![\varphi_\Delta]\!]\rho$, by induction on the length of Δ. The base case is trivial. Suppose that it holds for Δ; we prove it for $\Delta, x \mapsto \psi$. If $x \notin \mathrm{FV}(\varphi)$, it is immediate. Otherwise,

$$\begin{aligned}
[\![\varphi_{\Delta,x\mapsto\psi}]\!]\rho &= [\![(\varphi\{\psi/x\})_\Delta]\!]\rho && \text{by definition of } (\cdot)_\Delta \\
&= [\![\varphi\{\psi/x\}]\!]\rho_\Delta && \text{by inductive hypothesis} \\
&= [\![\varphi]\!]\rho_\Delta[x \mapsto [\![\psi]\!]\rho_\Delta] && \text{classical substitution lemma} \\
&= [\![\varphi]\!]\rho_{\Delta,x\mapsto\psi} && \text{by definition of } \rho_{\Delta,x\mapsto\psi}
\end{aligned}$$

■

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

DEFINITION 3.3. Let Γ be a finite set of formulæ, Δ a binding list, and φ a formula. We say that

1. φ *is a consequence of* Γ *under* Δ *with respect to* $\mathcal{M}$ (written $\Delta; \Gamma \models_{\mathcal{M}} \varphi$) iff for all ρ which agrees with Δ: $[\![\Gamma]\!]\rho \subseteq [\![\varphi]\!]\rho$ (where $[\![\Gamma]\!]\rho \stackrel{\text{def}}{=} \bigcap_{\varphi \in \Gamma} [\![\varphi]\!]\rho$);

2. φ *is a consequence of* Γ *under* Δ (written $\Delta; \Gamma \models \varphi$) iff for all models $\mathcal{M}$: $\Delta; \Gamma \models_{\mathcal{M}} \rho$;

3. φ *is a consequence of* Γ (written $\Gamma \models \varphi$) iff $\emptyset; \Gamma \models \rho$.

We finally come to the soundness and completeness of the $\mathbf{N}^{\trianglelefteq}_{\mu K}$ system:

THEOREM 3.1. *For all* Δ, Γ *and* φ*:* $\Delta; \Gamma \vdash \varphi \iff \Delta; \Gamma \models \varphi$.

Proof. Soundness ($\Rightarrow$) is proved by showing that each rule is sound. We show the most complex case, namely μ-E, the others being similar.

Let $\Delta, z \mapsto \varphi; \varphi\{z/x\} \models \psi$, where z does not appear in Δ, φ, ψ; we have to prove $\Delta; \mu x\varphi \models \psi$. This is equivalent to prove that $[\![\mu x\varphi]\!]\rho \subseteq [\![\psi]\!]\rho$ for any model $\mathcal{M}$ and environment ρ which agrees with Δ. By definition of $[\![\mu x\varphi]\!]$, it is sufficient to prove that $[\![\psi]\!]\rho$ is a prefixed point of φ^ρ_x, as follows:

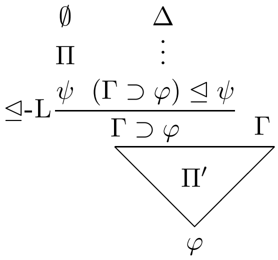
$$\begin{array}{rll} \varphi^\rho_x([\![\psi]\!]\rho) & = [\![\varphi]\!]\rho[x \mapsto [\![\psi]\!]\rho] & \text{by definition of } \varphi^\rho_x \\ & = [\![\varphi\{z/x\}]\!]\rho[z \mapsto [\![\psi]\!]\rho] & \text{for } z \text{ fresh} \\ & = [\![\varphi\{z/x\}]\!]\rho_{z \mapsto \psi} & \text{by definition of } \rho_{z \to \psi} \\ & \subseteq [\![\psi]\!]\rho_{z \mapsto \psi} & \text{because } \rho_{z \mapsto \psi} \models \Delta, z \mapsto \psi \text{ by} \\ & & \text{Proposition 3.2, and by hypothesis} \\ & = [\![\psi]\!]\rho & \text{since } z \notin \mathrm{FV}(\psi). \end{array}$$

Completeness ($\Leftarrow$) can be proved as follows. Since Γ is finite, let Γ be $\{\varphi_1, \ldots, \varphi_n\}$. Then $\Delta; \Gamma \models \varphi$ iff $\Delta; \emptyset \models \Gamma \supset \varphi$, where "$\Gamma \supset \varphi$" is a shorthand for $\varphi_1 \supset \ldots \supset \varphi_n \supset \varphi$. This means that

$$\forall \mathcal{M} = \langle S, [\![\cdot]\!] \rangle, \forall \rho : \rho \models \Delta \Rightarrow [\![\Gamma \supset \varphi]\!]\rho = S$$

which, by Proposition 3.2, is equivalent to $\forall \mathcal{M} = \langle S, [\![\cdot]\!] \rangle, \forall \rho : [\![\Gamma \supset \varphi]\!]\rho_\Delta = S$. By Lemma 3.1, there exists a formula ψ such that $\Delta \vdash (\Gamma \supset \varphi) \trianglelefteq \psi$ and $\forall \mathcal{M} = \langle S, [\![\cdot]\!] \rangle, \forall \rho : [\![\psi]\!]\rho = S$. This latter fact amounts to the validity of ψ, and hence, by completeness of Hilbert-style axiomatization [31], there is an Hilbert-style derivation of ψ. In order to prove the completeness of $\mathbf{N}^{\trianglelefteq}_{\mu K}$, therefore, it is sufficient to prove that all theorems provable by Hilbert-style system are theorems of $\mathbf{N}^{\trianglelefteq}_{\mu K}$, that is, they are derivable without using any assumptions nor bindings. In fact, if

Π is a proof of $\emptyset;\emptyset \vdash \psi$ in $\mathbf{N}^{\trianglelefteq}_{\mu K}$, then the following is a proof of $\Delta;\Gamma \vdash \varphi$:



where the subderivation $\Pi' : \Gamma, (\Gamma \supset \varphi) \vdash \varphi$ is made of n applications of $\supset$-E.

In order to prove that theorems of Kozen's system are theorems of $\mathbf{N}^{\trianglelefteq}_{\mu K}$, it is sufficient to prove that Kozen's axioms and rules (e.g. those in [27]) are derivable. We see only the case of axiom $\varphi\{\mu x\varphi/x\} \supset \mu x\varphi$:

$$\supset\text{-I}\;\cfrac{\mu\text{-I}\;\cfrac{\trianglelefteq\text{-L}\;\cfrac{(\varphi\{\mu x\varphi/x\})_1 \quad (\varphi\{z/x\})_3 \quad \begin{array}{c}(z \mapsto \mu x\varphi)_2\\ \Pi\\ \varphi\{z/x\} \trianglelefteq \varphi\{\mu x\varphi/x\}\end{array}}{\varphi\{z/x\}}\;(3)}{\mu x\varphi}\;(2)}{\varphi\{\mu x\varphi/x\} \supset \mu x\varphi}\;(1)$$

where the subderivation $\Pi : z \mapsto \mu x\varphi \vdash \varphi\{z/x\} \trianglelefteq \varphi\{\mu x\varphi/x\}$ is easily built by induction on the syntax of φ. ■

As an immediate corollary of Theorem 3.1, we recover the customary form of the adequacy of $\mathbf{N}^{\trianglelefteq}_{\mu K}$:

Corollary 3.1. *For all Γ finite and φ formula: $\emptyset;\Gamma \vdash \varphi \iff \Gamma \models \varphi$.*

As a final note on the proof system $\mathbf{N}^{\trianglelefteq}_{\mu K}$, we point out that we could consider a *strict expansion* $\triangleleft$, in place of $\trianglelefteq$. The judgement "$\varphi \triangleleft \psi$" would mean "$\varphi$ expands to ψ by applying at least one variable replacement;" hence, φ and ψ would always be different. However, although we could drop the reflexivity rule. we should add the following three rules:

$$\frac{x \mapsto \varphi}{x \triangleleft \varphi} \qquad \frac{\varphi_1 \triangleleft \varphi_2}{(\varphi_1 \supset \psi) \triangleleft (\varphi_2 \supset \psi)} \qquad \frac{\psi_1 \triangleleft \psi_2}{(\varphi \supset \psi_1) \triangleleft (\varphi \supset \psi_2)}$$

Hence, we have adopted the relation whose axiomatization involves the minor number of rules.

4. THE CALCULUS OF INDUCTIVE CONSTRUCTIONS

In this section we give a brief introduction to the *Calculus of Inductive Constructions* (CIC for short). For more details about CIC and logical frameworks, we refer the interested reader to [5, 13, 24, 25].

The Calculus of Inductive Constructions is an extension of Coquand and Huet's *Calculus of Constructions (CC)*, which can be defined as the PTS λC of Barendregt's λ-cube, with two sorts, `Prop` and `Set`. Under the *proposition-as-types, proofs-as-terms* paradigm, there is an isomorphism between propositions of intuitionistic higher-order logic and types of sort `Prop`. If A has type `Prop` then it represents a logical proposition; the fact that A is inhabited by a term M represents the fact that A holds. Each term M inhabiting A represents a *proof* of A. On the other hand, the sort `Set` is supposed to be the type of datatypes, such as naturals, lists, trees, booleans, etc. These types differ from those inhabiting `Prop` for their constructive contents.

Like any system extending the second-order PTS $\lambda 2$ (Girard's system F), CC allows to define *inductive types* by means of higher-order (impredicative) quantifications, but these representations are not always satisfactory (see [24] for a discussion). An alternative way for representing inductive types has been introduced in the *Calculus of Inductive Constructions (CIC)* by Thierry Coquand and Christine Paulin-Mohring and implemented in the Coq system [24, 5]. The idea is to extend the language of typed λ-terms by adding some special constants which represent the definition, introduction and elimination of inductive types. For instance, the following definition of natural numbers

```
Inductive nat : Set :=  O : nat | S : nat -> nat
```

allows to define terms by "case analysis", like the following function:

```
Definition pred := [n:nat]Cases n of   O => O
                                     | (S u) => u  end.
```

Using these elimination schemata, Coq automatically states and proves the induction principle for each inductively defined type. For instance, the above definition yields the Peano induction principle "for free":

```
nat_ind : (P:nat->Prop)(P O) ->
                        ((n:nat)(P n)->(P (S n))) -> (n:nat)(P n)
```

This feature has been extensively used in the definition of logical connectives: we need only to specify the introduction rules, and we can prove the elimination rules from the elimination principle the system automatically provides us.

However, allowing for *any* inductive definition in CIC would yield non-normalizing terms, thus invalidating the standard proof of consistency of the system. Hence, inductive definitions are subject to the *positivity condition*, which (roughly) requires that the type we are defining does not occur in negative position in the type of any constructor. This condition ensures the soundness of the system, but it rules out also many sound inductive definitions. For instance, the following definition of λ-terms

```
Inductive L : Set := Lam : (L->L) -> L | App : L -> L -> L.
```

is not well-formed, due to the negative occurrence of `L` in the type of `Lam`. This kind of representation is called *higher-order abstract syntax*, because one of the constructors takes a higher-order term (i.e., a function) as an argument. The problem of combining higher-order abstract syntax with inductive definitions is a very active area of work; for further details, we refer to [8, 9, 10, 11, 14, 16].

This problem may arise also at the propositional level, in particular when predicates are defined by Natural Deduction rules with discharged assumptions. For instance, the following elementary definition of minimal propositional logic

```
Inductive T : o -> Prop :=
      Imp_I : (A,B:o)((T A)->(T B)) -> (T (Imp A B))
    | Imp_E : (A,B:o)(T (Imp A B)) -> (T A) -> (T B).
```

is not well-formed in CIC. In this case, we can stick to the classical LF paradigm [13], i.e., we can represent the type and its constructors by means of constants of the right type. Thus, the minimal propositional logic can be encoded as follows:

```
Parameter T : o -> Prop.
Axiom Imp_I : (A,B:o)((T A)->(T B)) -> (T (Imp A B)).
Axiom Imp_E : (A,B:o)(T (Imp A B)) -> (T A) -> (T B).
```

Of course, defining a predicate by means of `Axioms` does not give rise automatically to any induction principle; hence, we need to introduce all eliminations rules explicitly. The induction principle over the type of proofs may be useful if we aim at proving *meta*theoretical results, such as cut-elimination of the proof system; it is not needed for doing proofs *in* the system, provided that we provide the right elimination rules. Usually this is not a problem for proof systems in Natural Deduction style, since they specify both introduction and elimination rules for each propositional constructor. Moreover, if the formalization is faithful, then its soundness and completeness correspond to the soundness and completeness of the proof system “on the paper”, by the isomorphism between types and proposition. Hence, given a sound and complete proof system (like $\mathbf{N}^{\lhd}_{\mu K}$), its formalizations are sound and complete if and only if they are faithful.

Another problem arising from the use of higher order abstract syntax together with inductive types is that of *exotic terms*. These are λ-terms which do not correspond to any object “on the paper”, despite their types correspond to some syntactic category. Exotic term are generated when a type has a higher-order constructor over an inductive type. A simple example is the following fragment of first-order logic:

```
Inductive i : Set := zero : i | one : i.
Inductive o : Set := ff : o | eq : i->i->o | forall : (i->o)->o.
Definition weird : o := (forall [x:i](Cases x of
                                          zero => ff
                                        | one  => (eq zero zero)
                                       end)).
```

The term `weird` does not correspond to any proposition of first order logic: there is no formula $\forall x \varphi$ such that $\varphi\{0/x\}$ and $\varphi\{1/x\}$ are syntactically equal to “*ff*” and “$0 = 0$”, respectively.

Exotic terms are problematic in establishing the faithfulness of the formalization; usually, they have to be ruled out by means of auxiliary “validity” judgements. Another approach (which will be used in Section 5.1) is to have the higher order constructors to range over types which are not defined as inductive, so that there is no Cases to use as above. See [8, 14] and [21, Section 11.2] for further details.

5. THE FORMALIZATION OF MODAL μ-CALCULUS

In this section we present the formalization of the modal μ-calculus in the Calculus of Inductive Constructions. We will then present both the formalization of the language and of the proof system $\mathbf{N}^{\lhd}_{\mu K}$ given in Section 3. For the complete vernacular code, see the Appendix.

5.1. Formalizing the language

The formalization of the language of modal 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* [8, 13]. 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, but this solution does not work inside an inductive definition of CIC (see Section 4).

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 [21].

In order to overcome the first problems, we adopt the *bookkeeping* technique [21]. We introduce a separate type, var, for the identifiers. The rôle played by variables is that of "placeholders" for formulæ: they will be bound to formulæ in the application of μ-I and μ-E rules, by means of an auxiliary judgement.

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

Notice that we do not define var as an inductive set. Indeed, the definition of the syntax of μ-calculus does not require the set of variables to be inductive (Section 1, and [17, 31]). Hence, we do not committ ourselves with unnecessary (and not harmless) assumptions; we only assume that there are infinitely many variables, by means of the var_nat axiom.

Then, we define the set of preformulæ of modal 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. This cannot give rise to exotic terms, since var is not declared as an inductive set.[2] Of course, the price we pay is that equality between variables is not decidable.

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 judgements posin, negin of type var->o->Prop. A careful analysis of the proof system (Figure 1) points out that the derivation of these judgements is completely syntax driven. It is therefore natural to define these judgements as *recursively defined functions*, instead of inductively defined propositions. This is indeed possible, 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 quantification is rendered by assuming a locally 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 Simplified (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.

[2] One could object that if we "instantiate" var on an inductive set, e.g. nat, we can effectively build exotic terms. The point is that one should be aware that such instantiation would automatically add a whole bunch of extra assumptions (induction and recursion principles, reduction rules, . . .) to the encoding. Clearly, adding arbitrary assumptions to a given theory may lead to inconsistencies; hence, it is not surprising if we get troubles in instantiating var to a specific inductive set. In fact, the definition of the syntax of μ-calculus does not require var to be inductive, so we can consistently leave it as an "open" (i.e., non-inductive) set.

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
 | (Not B) => (iswf B)
 | (Imp A1 A2) => (iswf A1)/\(iswf A2)
 | (Box a B) => (iswf B)
 | (Var y) => True
 | (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 judgement `notin` and the dual `isin` (see Section A.1) are auxiliary judgements for occur-checking. Roughly, `(notin x A)` holds iff `x` does not occur free in `A`; dually for `isin`.

Finally, each formula of the modal 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 establish that our formalization is faithful, we introduce the following notation: for $X = \{x_1, \ldots, x_n\} \subset Var$, let

$$\begin{aligned}
\Xi_X &\overset{\text{def}}{=} \mathtt{x_1} : \mathtt{var}, \ldots, \mathtt{x_n} : \mathtt{var}, \mathtt{e_{12}} : \tilde{}(\mathtt{x_1} = \mathtt{x_2}), \ldots, \mathtt{e_{n-1,n}} : \tilde{}(\mathtt{x_{n-1}} = \mathtt{x_n}) \\
\mathtt{o}_X &\overset{\text{def}}{=} \{\mathtt{t} \mid \Xi_X \vdash \mathtt{t} : \mathtt{o}\ \mathtt{t}\ \text{canonical}\} \\
\mathtt{wfo}_X &\overset{\text{def}}{=} \{\mathtt{t} \in \mathtt{o}_X \mid \exists \mathtt{d}.\Xi_X \vdash \mathtt{d} : (\mathtt{iswf\ t})\}.
\end{aligned}$$

We can then define the *encoding map* $\varepsilon_X : \Phi_X \to \mathtt{o}_X$, as follows:

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

LEMMA 5.1. *For $X \subset Var$ finite, for $\varphi \in \Phi_X$ and $x \in X$:*

1.$\vdash$ *posin*$(x, \varphi) \iff \exists \texttt{t}.\Xi_X \vdash \texttt{t} : (\texttt{posin x } \varepsilon_X(\varphi))$

2.$\vdash$ *negin*$(x, \varphi) \iff \exists \texttt{t}.\Xi_X \vdash \texttt{t} : (\texttt{negin x } \varepsilon_X(\varphi))$

Proof. By simultaneous induction on the syntax of φ.
Base case. The only interesting case is when $\varphi = y$. If $\vdash$ *negin*(x, y) holds, then y is different from x. By definition of Ξ_X, there is the assumption $\texttt{e}_{\texttt{xy}} : \tilde{}(\texttt{x} = \texttt{y})$, hence $\texttt{t} = \texttt{e}_{\texttt{xy}}$. On the other hand, suppose that $\texttt{t}$ is such that $\Xi_X \vdash \texttt{t} : (\texttt{negin x (Var y)})$. Since $(\texttt{negin x (Var y)})$ reduces to $\tilde{}(\texttt{x} = \texttt{y})$, $\texttt{t}$ has to be an assumption $\texttt{e}_{\texttt{xy}}$. Hence, x is different from y, so $\vdash$ *negin*(x, y) holds.
Inductive case. The only interesting case is $\varphi = \mu y \psi$. Then, $\vdash$ *posin*$(x, \mu y \psi)$

$\iff \vdash$ *posin*(x, ψ) (definition of *posin*) ■
$\iff \exists \texttt{t}.\Xi_{X,y} \vdash \texttt{t} : (\texttt{posin x } \varepsilon_{X,y}(\psi))$ (ind. hypothesis)
$\iff \exists \texttt{t}'.\Xi_X \vdash \texttt{t}' : (\texttt{y : var})\tilde{}(\texttt{x} = \texttt{y})\texttt{->}(\texttt{posin x } \varepsilon_{X,y}(\psi))$
$\iff \exists \texttt{t}'.\Xi_X \vdash \texttt{t}' : (\texttt{posin x (mu [y : var]}\varepsilon_{X,y}(\psi))$ (type conversion)
$\iff \exists \texttt{t}'.\Xi_X \vdash \texttt{t}' : (\texttt{posin x } \varepsilon_X(\mu y \psi))$ (definition of ε)

LEMMA 5.2. *For $X \subset Var$ finite, for φ preformula:*

$$\varphi \in \Phi_X \iff \exists \texttt{t}.\Xi_X \vdash \texttt{t} : (\texttt{iswf } \varepsilon_X(\varphi))$$

Proof. By induction on the syntax of φ. The only interesting case is $\varphi = \mu x \psi$. Then, $\varphi \in \Phi_X$ iff $\psi \in \Phi_{X,x}$ and $\vdash$ *posin*(x, ψ) (by definition of *iswf*). By inductive hypothesis and Lemma 5.1, this holds iff there exist $\texttt{t}_1, \texttt{t}_2$ such that $\Xi_X, \texttt{x} : \texttt{var} \vdash \texttt{t}_1 : (\texttt{iswf } \varepsilon_{X,x}(\psi)$ and $\Xi_{X,x} \vdash \texttt{t}_2 : (\texttt{posin x } \varepsilon_{X,x}(\psi))$. Now, the only assumptions of the form $\texttt{e} : \tilde{}(\texttt{y} = \texttt{z})$ which can be really needed in $\texttt{t}_2$, have $\texttt{x}$ in place of $\texttt{y}$ and $\texttt{z} \in \mathrm{FV}(\psi) \setminus \{x\}$ (see the proof of Lemma 5.1), so we can drop all the others. Moreover, all these assumptions can be derived from $(\texttt{notin x (mu [x : var]}\varepsilon_{X,x}(\psi)))$ (easily proved by induction on ψ). Hence,

$$\Xi_X, \texttt{x} : \texttt{var}, \texttt{n} : (\texttt{notin x (mu [x : var]}\varepsilon_{X,x}(\psi))) \vdash \texttt{t}_2 : (\texttt{posin x } \varepsilon_{X,x}(\psi)).$$

By abstracting $\texttt{t}_1$ over $\texttt{n}$, and combining it with $\texttt{t}_2$ we obtain a term $\texttt{t}'$ such that

$$\begin{aligned}\Xi_X \vdash \texttt{t}' : (\texttt{x : var})(\texttt{iswf } \varepsilon_{X,x}(\psi))\texttt{/\textbackslash}\\ ((\texttt{notin x (mu [x : var]}\varepsilon_{X,x}(\psi)))\ \texttt{->}\ (\texttt{posin x } \varepsilon_{X,x}(\psi)))\end{aligned}$$

The type of this term is convertible to $(\texttt{iswf } \varepsilon_X(\mu x \psi))$, hence the thesis. ■

The faithfulness of our formalization is therefore stated in the following theorem:

THEOREM 5.1. *For $X \subset Var$ finite, the map ε_X is a compositional bijection between Φ_X and $\texttt{wfo}_X$.*

Proof. Clearly, ε_X is injective and compositional. For $\varphi \in \Phi_X$, by Lemma 5.2 the type $(\texttt{iswf}\ \varepsilon_X(\varphi))$ is inhabited, hence $\varepsilon_X(\varphi) \in \texttt{wfo}_X$. Surjectivity can be proved by defining the inverse map $\delta_X : \texttt{wfo}_X \to \Phi_X$, as follows:

$$\begin{array}{ll} \delta_X((\texttt{Var x})) = x & \delta_X(\texttt{(Imp } A\ B\texttt{))} = \delta_X(A) \supset \delta_X(B) \\ \delta_X(\texttt{(Not } A\texttt{))} = \neg\delta_X(A) & \delta_X(\texttt{(Box a } A\texttt{))} = [a]\delta_X(A) \\ \delta_X(\texttt{ff}) = \mathit{ff} & \delta_X(\texttt{(mu [x:var]}A\texttt{))} = \mu x \delta_{X,x}(\varphi) \end{array}$$

This map is well defined: each term in $\texttt{wfo}_X$ has a proof of its well-formedness, and hence it correspond to a well-formed formula. It is easy to prove that for all $\texttt{t} \in \texttt{wfo}_X : \varepsilon_X(\delta_X(\texttt{t})) = \texttt{t}$ (by induction on the syntax of t). ■

5.2. Formalizing the proof system $\mathbf{N}^{\lhd}_{\mu K}$

In the formalization paradigm of Logical Frameworks, a proof system is usually represented by introducing a *proving judgement* 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 [2, 8, 13].

However, in representing the proof system $\mathbf{N}^{\lhd}_{\mu K}$, several difficult issues arise. These issues escape the standard formalization paradigm, so we have to accommodate some special technique. In the following subsections, we will describe in detail these problems and the solutions we adopted.

5.2.1. *Formalization of classical, least-fixed point and modal rules*

The formalization of modal proof rules, like Sc and μ-E, requires special care. Moreover, Scott's rule is parametric in the number of assumptions which have to be "boxed". 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 judgement `Seq:olist->o->Prop`, and all the machinery of Gentzen's original system [12]. 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". The solution we adopt exploits again the possibility provided by Logical Frameworks of considering locally quantified premises, i.e. general judgements in the terminology of Martin-Löf. For a in-depth discussion on this technique for the representation of modal logics, we refer to [3].

First, we represent $\mapsto$ and $\trianglelefteq$ by means of two judgements `bind:var->o->Prop` and `expto:o->o->Prop`, respectively. The former has no constructor (it is 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 (new) variable (see Appendix A.2 for the whole listing):

```
Parameter bind : var -> o -> Prop.
(* structural closure of bind *)
Inductive expto : o->o->Prop :=
   expto_rflx : (phi:o)(expto phi phi)
 | expto_btrns: (x:var)(phi,psi:o)
                (bind x phi)->(expto phi psi)->(expto (Var x) psi)
 (... other rules ...)
 | expto_mu   : (phi,psi:var->o)((x:var)(expto (phi x) (psi x)))->
                (expto (mu phi) (mu psi)).
```

Then, we introduce the basic proving judgement, `T:U->o->Prop`:

```
Parameter U : Set.
Parameter T : U -> o -> Prop.
```

where the type `U` (the *universe*) is an open (i.e., non-inductive) set with no constructor. Hence, the only terms which can inhabit `U` are variables, which will be called *worlds* for suggestive reasons.[3] Each pure rule (i.e., with no side condition), is parameterized over a generic world, like the following:

```
Axiom Imp_I : (w:U)(phi,psi:o)
              ((T w phi) -> (T w psi)) -> (T w (Imp phi psi)).
```

Therefore, in a given world all the classical rules apply as usual. It should be noticed that these rule are schematic on generic preformulæ, and not only on well formed formulæ. The reason of this will be discussed in Section 5.2.3; for the moment, we just point out that we require the `phi` in `Imp_E` and `Not_E`, and the `(mu F)` in `mu_E`, to be well-formed, e.g. as follows:

```
Axiom Imp_E : (phi,psi:o)(w:U)(iswf phi) ->
              (T w (Imp phi psi)) -> (T w phi) -> (T w psi).
```

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 represented as follows:

```
Axiom mu_E   : (F:var->o)(iswf (mu F)) ->
               ((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).
```

[3] The reader aware of Kripke models can see here a clear connection with the semantics of modal logics, as pointed out in [4]. However, we would stress that these variables can be seen just as a syntactic device on their own, without bearing in mind any particular semantics. In fact, the technique can be applied to any logic with similar rules, even if these logics do not have a Kripke-like model.

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 judgement T also on w. This judgement then appears as an hypothesis on w. Hence, deriving as premise a judgement, which is universally quantified with respect to U, amounts to establishing the judgement for the generic world w' on which only the given assumptions are made, i.e. on the given assumptions.

The formalization of μ-E (and μ-I) uses also the auxiliary judgement bind. Following the idea of $\mathbf{N}^{\lhd}_{\mu K}$, the context $\varphi(\cdot)$ of $\mu x \varphi(x)$ is filled with 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). Since the typing rules of CIC automatically ensure that z is different from any other free variable, the set of bindings in the environment still form a binding list. These bindings can be used in the derivation of expansion judgements, for replacing formulæ only when it is needed. For an example, see Appendix A.3.

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.

5.2.2. *Formalization of sequent-style rules*

The idea presented in the previous section can be suitably generalized to take care of an unlimited number of assumptions. A generic sequent $\varphi_1, \ldots, \varphi_n \vdash \varphi$ is faithfully represented by the type (w:U)(T w phi$_1$)->...->(T w phi$_n$)->(T w phi) where phi$_i = \varepsilon_X(\varphi_i)$ and phi $= \varepsilon_X(\varphi)$. The locally quantified world w forces any proof of (T w phi) to depend only on the given assumptions. The problem is to capture the parametricity expressed by the "...". At this end, we introduce the auxiliary set of *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; phi:o; l:olist] : Prop :=
        Cases l of
            nil => (T w phi)
          | (cons phii l') => (T w phii) -> (Sequent w phi l')
        end.
```

Therefore, the aforementioned representation of $\varphi_1, \ldots, \varphi_n \vdash \varphi$ is denoted by (w:U)(Sequent w phi G) where G is the list composed by phi$_1$,...,phi$_n$. In fact, (Sequent w phi G) is exactly $\beta\iota\delta$-equivalent (that is, it reduces) to (T w phi$_1$)->...->(T w phi$_n$)->(T w phi). Hence, Scott's rule is represented as follows:

```
Fixpoint Boxlist [a:Act; l:olist] : olist :=
        Cases l of
            nil => nil
          | (cons psi l') => (cons (Box a psi) (Boxlist a l'))
        end.
Axiom Sc : (phi:o)(w:U)(G:olist)(a:Act)
```

```
((w':U)(Sequent w' phi G))
-> (Sequent w (Box a phi) (Boxlist a G)).
```

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

5.2.3. *Dealing with non-well-formed formulæ*

Although it allows for a faithful representation of the syntax, the use of Σ-types is not harmless when we come to the representation and usage of the proof system. In fact, a strict formalization of the rules should consider only terms of type `wfo`, as follows

```
Axiom w_Imp_E : (phi,psi:wfo)(w:U)
      (T w (Imp (prp phi) (prp psi))) -> (T w (prp phi)) ->
      (T w (prp psi)).
```

where `prp:wfo->o` is the first projection defined on the Σ-type `wfo`. This differs from the rules we have presented in the previous sections, where we allow for dealing with generic preformulæ. Actually, the usage of a "well formed formulæ only" formalization would be quite cumbersome and awkward, because at every rule application we would be required to provide the proof of well-formedness of the involved formulæ. Hence we need a "light" version of $\mathbf{N}^{\trianglelefteq}_{\mu K}$, where the well-formedness constraints are reduced to a minimum.

Let us call *prederivations* the derivations of $\mathbf{N}^{\trianglelefteq}_{\mu K}$ where also preformulæ are allowed. Of course, every derivation is also a prederivation, but there are many prederivations of meaningless sequents, like the following $\Pi : \emptyset; \emptyset \vdash \mu x \neg x$

$$\text{R{\small AA}}\,\dfrac{\neg\text{-E}\,\dfrac{\begin{array}{c}(\neg\mu x\neg x)_1\\ \Pi'\\ \mu x\neg x\end{array}\qquad (\neg\mu x\neg x)_1}{f\!f}}{\mu x\neg x}\,(1)$$

where the derivation $\Pi' : \emptyset; \neg\mu x \neg x \vdash \mu x \neg x$ is an instance of a subderivation of $\varphi\{\mu x\varphi/x\} \supset \mu x\varphi$ which appears in the proof of Theorem 3.1. Hence we need to add some well-formedness constraint, in order to restrict prederivations to only derivations.

As far as the expansion system is concerned, we have the following result:

Proposition 5.1. *Let Δ be a binding list containing only well-formed formulæ, and φ, ψ preformulæ such that $\Delta \vdash \varphi \trianglelefteq \psi$. Then, φ is well-formed iff ψ is well-formed.*

Proof. By induction on the derivation of $\Delta \vdash \varphi \trianglelefteq \psi$. The case of reflexivity rule is trivial. Congruence rules also are easy, the most complex case being that of μ. Suppose that $\Delta \vdash \mu x\varphi \trianglelefteq \mu x\psi$ ends with an application of the congruence rule w.r.t. μ. Then, $\mu x\varphi$ is well-formed iff $\varphi\{z/x\}$ is well-formed and z appears only positively in $\varphi\{z/x\}$, iff $\psi\{z/x\}$ is well-formed (by inductive hypothesis) and z appears only positively in $\psi\{z/x\}$ (easy to prove), iff $\mu x\varphi$ is well-formed.

Finally, suppose that $\Delta \vdash x \trianglelefteq \psi$ ends with an application of the "variable replacement rule." Then, there is a binding $x \mapsto \varphi$ in Δ. By assumption, φ is well-formed, and hence by inductive hypothesis, also ψ is. ■

On the other hand, for derivations of the logical system we introduce the following notion of well-formedness:

Definition 5.1. The preformula φ in rules $\supset$-E, $\neg$-E and the formula $\mu x\varphi$ in rule μ-E are called *cut preformulæ.*

A prederivation $\Pi : \Delta; \Gamma \vdash \varphi$ is *well-formed* (*wf-prederivation*) if all preformulæ in Δ, Γ, φ and all cut preformulæ are well-formed.

Clearly, every derivation is a wf-prederivation. Also the converse holds:

Proposition 5.2. *Every wf-prederivation is a derivation.*

Proof. We have to prove that if $\Pi : \Delta; \Gamma \vdash \varphi$ is a wf-prederivation, then every preformula in it is well-formed. We proceed by induction on the depth of Π.

If Π is an assumption, then $\varphi \in \Gamma$ and hence it is well-formed.

If the last rule of $\Pi : \Delta; \Gamma \vdash \varphi \supset \psi$ is $\supset$-I, then there is a prederivation $\Pi : \Delta; \Gamma, \varphi \vdash \psi$, which is well-formed because φ and ψ are well-formed. By inductive hypothesis, Π' is a derivation, and hence also Π is. Similarly if the last rule of Π is $\neg$-I, Raa, Sc.

If the last rule of $\Pi : \Delta; \Gamma \vdash \psi$ is $\supset$-E, then there are two prederivations $\Pi' : \Delta; \Gamma \vdash \varphi$ and $\Pi'' : \Delta; \Gamma \vdash \varphi \supset \psi$. Since φ is a cut formula, then it is well-formed by the hypothesis that Π is well-formed. Thus, Π' and Π'' are wf-derivations and hence, for inductive hypothesis, they are derivations. So also Π is. Similarly if the last rule of Π is $\neg$-E.

If the last rule is $\trianglelefteq$-L or $\trianglelefteq$-R, then the thesis follows from Proposition 5.1.

If the last rule of $\Pi : \Delta; \Gamma \vdash \psi$ is μ-E, then there are two prederivations $\Pi' : \Delta; \Gamma \vdash \mu x\varphi$ and $\Pi'' : \Delta, z \mapsto \psi; \Gamma, \varphi\{z/x\} \vdash \psi$. Since $\mu x\varphi$ is a cut formula, then it and $\varphi\{z/x\}$ are well-formed by the hypothesis that Π is well-formed. Thus, Π' and Π'' are wf-derivations and hence, for inductive hypothesis, they are derivations. So also Π is. Similarly if the last rule of Π is μ-I. ■

This result allows us to adopt the "relaxed" formalization of $\mathbf{N}^{\trianglelefteq}_{\mu K}$ we have discussed in the previous subsections: all the well-formedness checks but those on cut formulæ can be dropped, provided that the results we have to prove are stated only on well-formed formulæ. For an example, see Appendix A.3.

5.2.4. Adequacy of the formalization

In order to state the adequacy of our formalization with respect to $\mathbf{N}^{\trianglelefteq}_{\mu K}$, we introduce the following notation. Let $X \subset \mathit{Var}$ be finite, and $\varphi_1, \ldots, \varphi_n, \varphi \in \Phi_X$;

then, for $x_1, \ldots, x_n \in X$ and for $\mathtt{w:U}$, we define

$$\delta_X(x_1 \mapsto \varphi_1, \ldots, x_n \mapsto \varphi_n) \stackrel{\text{def}}{=} \mathtt{b}_1\texttt{: (bind x}_1\ \varepsilon_X(\varphi_1)), \ldots \mathtt{b}_n\texttt{: (bind x}_n\ \varepsilon_X(\varphi_n))$$
$$\gamma_{X,\mathtt{w}}(\varphi_1, \ldots, \varphi_n) \stackrel{\text{def}}{=} \mathtt{h}_1\texttt{: (T w } \varepsilon_X(\varphi_1)), \ldots, \mathtt{h}_n\texttt{: (T w } \varepsilon_X(\varphi_n)).$$

LEMMA 5.3. *Let $X \subset Var$ be finite, Δ a binding list such that* $\mathrm{FV}(\Delta) \subseteq X$. *Then, for all $\varphi_1, \varphi_2 \in \Phi_X$: $\Delta \vdash \varphi_1 \trianglelefteq \varphi_2$ if and only if there exists* $\mathtt{t}$ *canonical such that* $\Xi_X, \delta_X(\Delta) \vdash \mathtt{t} : (\mathtt{expto}\ \varepsilon_X(\varphi_1)\ \varepsilon_X(\varphi_2))$

Proof. ($\Rightarrow$) By induction on the proof of $\Delta \vdash \varphi_1 \trianglelefteq \varphi_2$. Most cases are trivial; let us consider the case when the last rule applied is the structural rule for μ. Then, $\varphi_1 = \mu x \psi_1$ and $\varphi_2 = \mu x \psi_2$ for some ψ_1, ψ_2, and $\Delta \vdash \psi_1\{z/x\} \trianglelefteq \psi_2\{z/x\}$ for z fresh. By inductive hypothesis, there is a term $\mathtt{t}_1$ such that

$$\Xi_{X,z}, \delta_X(\Delta) \vdash \mathtt{t} : (\mathtt{expto}\ \varepsilon_{X,z}(\psi_1\{z/x\})\ \varepsilon_{X,z}(\psi_2\{z/x\})).$$

By abstracting on z and applying `expto_mu`, we obtain a term $\mathtt{t}'$ such that

$$\begin{aligned}\Xi_X, \delta_X(\Delta) \vdash \mathtt{t}' : (&\mathtt{expto}\ (\mathtt{mu}\ [\mathtt{z} : \mathtt{var}]\varepsilon_{X,z}(\psi_1\{z/x\})) \\ &(\mathtt{mu}\ [\mathtt{z} : \mathtt{var}]\varepsilon_{X,z}(\psi_2\{z/x\})))\end{aligned}$$

which is equivalent to $\Xi_X, \delta_X(\Delta) \vdash \mathtt{t}' : (\mathtt{expto}\ \varepsilon_X(\varphi_1)\ \varepsilon_X(\varphi_2))$ by α-conversion and definition of ε_X.

($\Leftarrow$) By induction on the syntax of term `t`, by cases on the head constructor. Notice that the head constructor of `t` cannot be a `Case`. Suppose that `t` is `(Case` A `of ...)`; then, A has to be a variable since `t` is canonical. But the only variables are those in $\Xi_X, \delta_X(\Delta)$, and none of them belongs to a inductive type—absurd. Hence, each constructor of type `expto` corresponds always to a rule of the expansion system, thus the proof can be readily recovered. ■

THEOREM 5.2. *Let $X \subset Var$ be finite, Δ a binding list such that* $\mathrm{FV}(\Delta) \subseteq X$, *and $\Gamma \subset \Phi_X$ finite. Then, for all $\varphi \in \Phi_X$: $\Delta; \Gamma \vdash \varphi$ if and only if there exists* $\mathtt{t}$ *canonical such that* $\Xi_X, \delta_X(\Delta), \mathtt{w} : \mathtt{U}, \gamma_{X,\mathtt{w}}(\Gamma) \vdash \mathtt{t} : (\mathtt{T\ w}\ \varepsilon_X(\varphi))$.

Proof. ($\Rightarrow$) By induction on the proof of $\Delta; \Gamma \vdash \varphi$. Here,we show briefly the case for μ-E, the others being similar (or simpler).[4]

Let $\Delta; \Gamma \vdash \mu x \varphi$ and $\Delta, z \mapsto \psi; \varphi\{z/x\} \vdash \psi$. By inductive hypothesis, there exist two terms $\mathtt{t}_1, \mathtt{t}_2$ such that

$$\Xi_X, \delta_X(\Delta), \mathtt{w:U}, \gamma_{X,\mathtt{w}}(\Gamma) \vdash \mathtt{t}_1 : (\mathtt{T\ w}\ [x : var]\varepsilon_{X,x}(\varphi))$$
$$\Xi_{X,x}, \delta_X(\Delta), \mathtt{b:(bind\ x}\ \varepsilon_X(\psi)), \mathtt{w:U}, \mathtt{h:(T\ w}\ \varepsilon_{X,x}(\varphi) \vdash \mathtt{t}_2 : (\mathtt{T\ w}\ \varepsilon_X(\psi))$$

By abstracting $\mathtt{t}_2$ over `b`, `w` and `x`, we obtain a term $\mathtt{t}_2'$ such that

$$\Xi_X, \delta_X(\Delta) \vdash \mathtt{t}_2' : (\mathtt{x} : \mathtt{var})(\mathtt{bind\ x}\ \varepsilon_X(\psi))\texttt{->}(\mathtt{w:U})(\mathtt{T\ w}\ \varepsilon_{X,x}(\varphi))\texttt{->}(\mathtt{T\ w}\ \varepsilon_X(\psi)).$$

[4] See [3] for a full description of the adequacy of encodings based on the world technique.

Moreover, since $\mu x \varphi$ is well-formed, by Lemma 5.2 there is a term $\mathtt{t}_3$ such that $\Xi_X \vdash \mathtt{t}_3 : \mathtt{(iswf}\ \varepsilon_X(\mu x \varphi)\mathtt{)}$. We can thus build the required proof by applying the constructor `mu_E` to $\mathtt{t}_1$, $\mathtt{t}_2'$ and $\mathtt{t}_3$.

($\Leftarrow$) By induction on the syntax of `t`: each constructor of `(T w phi)` corresponds to a rule of $\mathbf{N}^{\lhd}_{\mu K}$. We show the case of `mu_I`.

Let `t` be `(mu_I F [z:var][b:(bind z (mu F))]t')` of type `(T w (mu F))`, where `(mu F)` is $\varepsilon_X(\mu x \varphi)$. We have

$$\Xi_{X,z}, \delta_X(\Delta), \mathtt{b} : (\mathtt{bind\ z}\ (\mathtt{mu\ F})), \mathtt{w} : \mathtt{U}, \gamma_{X,\mathtt{w}}(\Gamma) \vdash \mathtt{t}' : (\mathtt{T\ w\ (F\ z)}),$$

where z is a fresh variable, and `(F z)` is the encoding of $\varphi\{z/x\}$. Hence, $\delta_X(\Delta), \mathtt{b} : (\mathtt{bind\ z}\ (\mathtt{mu\ F}))$ is the encoding of the binding list $\Delta, z \mapsto \mu x \varphi$. By the inductive hypothesis, there exists a proof $\Delta, z \mapsto \mu x \varphi; \Gamma \vdash \varphi$; by applying the rule μ-I, we obtain $\Delta; \Gamma \vdash \mu x \varphi$. ■

As a consequence of this result, every proof which can be build "on the paper" using the system $\mathbf{N}^{\lhd}_{\mu K}$, can be readily mimicked step-by-step in the encoding of $\mathbf{N}^{\lhd}_{\mu K}$ in Coq, and *vice versa*. Hence, since $\mathbf{N}^{\lhd}_{\mu K}$ is sound and complete for the propositional μ-calculus, then also its encoding is sound and complete.

6. CONCLUSIONS

In this paper we have introduced a proof system $\mathbf{N}^{\lhd}_{\mu K}$ for the propositional modal μ-calculus, and its formalization in the Calculus of Inductive Constructions. Beside the formalization, $\mathbf{N}^{\lhd}_{\mu K}$ 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 $\mathbf{N}^{\lhd}_{\mu K}$ substitutions of formulæ for variables are not always performed, but they may be delayed until actually needed.

In the formalization, 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 an *expansion* 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 modal μ-calculus by formalizing the notion of "well-formed formula." Finally, the modal nature of impure rules of modal 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, such as the λ-calculus, higher-order process calculi, languages defined by context-sensitive grammars...

Moreover, our experience confirmed also in dealing with the modal μ-calculus, is that Logical Frameworks allow to encode faithfully the formal systems under consideration, without imposing on the user of the proof editor the burden of cumbersome translations into, say, first-order logic or monadic second-order logic. However, nowadays proof editors and Logical Frameworks are still under development;

hence, they will benefit from extensive case studies and applications, like the one presented here, which can enlighten weak points and suggest further improvements.

Related and future work. The representation of object-logic variables by means of meta-level variables of a generic set `var` without constructors has been used before in [15] and exploited in [16] in the full formalization of the π-calculus and its metatheory. The implementation of substitution by means of an environment of bindings has been previously investigated in the context of logic programming by Miller [22], and in that of model checking by Stirling and Walker [28]. This fact deserves further investigation. For instance, $\mathbf{N}^{\lhd}_{\mu K}$ could be integrated with a model checker in a simple and efficient way; the model checker could be implemented in Coq, and its correctness formally verified. Previously, Yu and Luo formalized a model checker for CCS and the μ-calculus in LEGO [30]. Although in their encoding, binding operators are represented by means of de Bruijn indexes, their approach should be feasible also in the HOAS-based formalization presented in this paper.

At the moment, a great effort is spent on the problem of combining higher-order abstract syntax with inductive definitions. The aim is to allow for higher-order constructors inside inductive definitions, together with adequate elimination principles. Eventually, these improvements should lead to new Logical Frameworks, adequately supporting both HOAS and recursion/induction. For further details on recents developments in this area, we refer to [8, 9, 10, 11, 14].

From a proof-theoretical point of view, rule Sc is not satisfactory, since it breaks the typical introduction/elimination pattern of Natural Deduction systems. A pure Natural Deduction-style proof system for the propositional μ-calculus (i.e., without the modal constructor "$[a]\varphi$") has been investigated in [29]. Whether there is a truly Natural Deduction formulation of the modal μ-calculus remains an open question. A promising approach to the development of a system suitable for proof-theoretical results is to reflect in the system a semantic notion, like the transition relation of the underlying model. This approach has been successfully adopted by Simpson in the construction of strong normalizing Natural Deduction style proof systems for modal logics [26].

Another future work stemming from this research is the development of a user friendly, mouse oriented environment which adopts our Coq formalization as proof kernel. In such an environment, the user could carry out interactively formal verifications based on the modal μ-calculus.

APPENDIX: Coq CODE

This code is also available at `http://www.dimi.uniud.it/~miculan/mucalculus`.

A.1. CODE OF THE FORMALIZATION OF THE SYNTAX

```
(* Sets for actions, variables *)
Parameter Act, var : Set.
(* var is at least enumerable *)
Axiom var_nat : (Ex [srj:var->nat](n:nat)(Ex [x:var](srj x)=n)).

Lemma neverempty : (x:var)(Ex [y:var]~(x=y)).
```

```
(* proof omitted *)

(* the set of preformulae, also not well formed *)
Inductive o   : Set := ff : o
                     | Not : o -> o
                     | Imp : o -> o -> o
                     | Box : Act -> o -> o
                     | Var : var -> o
                     | mu  : (var->o) -> o.

Fixpoint isin [x:var;phi:o] : Prop :=
   <Prop>Cases phi of
     ff              => False
   | (Not psi)       => (isin x psi)
   | (Imp phi1 phi2) => (isin x phi1)\/(isin x phi2)
   | (Box a psi)     => (isin x psi)
   | (Var y)         => x=y
   | (mu F)          => (y:var)(isin x (F y))
   end.

Fixpoint notin [x:var;phi:o] : Prop :=
   <Prop>Cases phi of
     ff              => True
   | (Not psi)       => (notin x psi)
   | (Imp phi1 phi2) => (notin x phi1)/\(notin x phi2)
   | (Box a psi)     => (notin x psi)
   | (Var y)         => ~(x=y)
   | (mu F)          => (y:var)~(x=y)->(notin x (F y))
   end.

Fixpoint posin [x:var;phi:o] : Prop :=
   <Prop>Cases phi of
     ff              => True
   | (Not psi)       => (negin x psi)
   | (Imp phi1 phi2) => (negin x phi1)/\(posin x phi2)
   | (Box a psi)     => (posin x psi)
   | (Var y)         => True
   | (mu F)          => (y:var)~(x=y)->(posin x (F y))
   end
with negin [x:var;phi:o] : Prop :=
   <Prop>Cases phi of
     ff              => True
   | (Not psi)       => (posin x psi)
   | (Imp phi1 phi2) => (posin x phi1)/\(negin x phi2)
   | (Box a psi)     => (negin x psi)
   | (Var y)         => ~(x=y)
   | (mu F)          => (y:var)~(x=y)->(negin x (F y))
   end.

Fixpoint iswf [phi:o] : Prop :=
   <Prop>Cases phi of
     ff              => True
   | (Not psi)       => (iswf psi)
   | (Imp phi1 phi2) => (iswf phi1)/\(iswf phi2)
   | (Box a psi)     => (iswf psi)
   | (Var y)         => True
   | (mu F)          => (x:var)(iswf (F x))/\
                               ((notin x (mu F)) -> (posin x (F x)))
   end.

(* the set of well formed formuale *)
Record wfo : Set := mkwfo {prp : o;
                           cnd : (iswf prp)}.

(* separation: if x does not appear in phi and y do, then x and y
 * are not the same identifiers - proof omitted *)
Lemma separation : (x,y:var)(phi:o)
                   (notin x phi) -> (isin y phi) -> ~(x=y).
Lemma notin_posin_negin :
         (x:var)(phi:o)(notin x phi) -> (posin x phi)/\(negin x phi).
Lemma notin_posin : (x:var)(phi:o)(notin x phi) -> (posin x phi).
Lemma notin_negin : (x:var)(phi:o)(notin x phi) -> (negin x phi).
```

A.2. CODE OF THE FORMALIZATION OF THE PROOF SYSTEM

```
(* the binding judgement *)
Parameter bind : var -> o -> Prop.

(* expansion precongruence *)
Section Expansion.
Inductive expto : o->o->Prop :=
    expto_rflx : (phi:o)(expto phi phi)
  | expto_btrns: (x:var)(phi,psi:o)
                 (bind x phi)->(expto phi psi)->(expto (Var x) psi)
  | expto_Not  : (phi,psi:o)
                 (expto phi psi) -> (expto (Not phi) (Not psi))
  | expto_Imp  : (phi1,phi2,psi1,psi2:o)
                 (expto phi1 psi1)->(expto phi2 psi2)->
                 (expto (Imp phi1 phi2) (Imp psi1 psi2))
  | expto_Box  : (phi,psi:o)(expto phi psi)->
                 (a:Act)(expto (Box a phi) (Box a psi))
  | expto_mu   : (phi,psi:var->o)((x:var)(expto (phi x) (psi x))) ->
                 (expto (mu phi) (mu psi)).
Hints Immediate expto_bind expto_rflx expto_btrns
```

```
                expto_Not expto_Imp expto_Box expto_mu : core.

Lemma expto_bind : (x:var)(phi:o)(bind x phi)->(expto (Var x) phi).
Lemma expto_trns : (phi,psi:o)(expto phi psi)->
                        (xi:o)(expto psi xi) -> (expto phi xi).
(* proofs omitted *)
End Expansion.

(* lists of formulae for Scott rule *)
Inductive olist : Set :=
          nil : olist
        | cons : o -> olist -> olist.

(* Boxlist represents the [a]\Gamma notation *)
Fixpoint Boxlist [a:Act; l:olist] : olist :=
        Cases l of
            nil => nil
          | (cons psi l') => (cons (Box a psi) (Boxlist a l'))
        end.

(* the universe, for the world technique *)
Parameter U : Set.

(* the proving judgement *)
Parameter T : U -> o -> Prop.

(* (Sequent w phi (phi1...phin)) corresponds to phi1,...phin |- phi *)
Fixpoint Sequent [w:U; phi:o; l:olist] : Prop :=
        Cases l of
            nil => (T w phi)
          | (cons phii l') => (T w phii) -> (Sequent w phi l')
        end.

Section Proof_Rules.
Variable phi,psi,xi:o.
Variable w:U.

Axiom Not_I   : ((T w phi) -> (T w ff)) -> (T w (Not phi)).
Axiom Not_E   : (iswf phi) -> (T w phi)->(T w (Not phi)) -> (T w ff).
Axiom RAA     : ((T w (Not phi)) -> (T w ff)) -> (T w phi).
Axiom Imp_I   : ((T w phi) -> (T w psi)) -> (T w (Imp phi psi)).
Axiom Imp_E   : (iswf phi) ->
                (T w (Imp phi psi)) -> (T w phi) -> (T w psi).

Axiom Sc      : (G:olist)(a:Act)
                ((w':U)(Sequent w' phi G))
                -> (Sequent w (Box a phi) (Boxlist a G)).
```

```
Axiom expto_R : (expto phi psi) -> (T w phi) -> (T w psi).
Axiom expto_L : (expto phi psi) -> (T w psi) -> (T w phi).

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

(* some derived rules - proofs omitted *)
Lemma ff_E : (T w ff) -> (T w phi).
Lemma K : (a:Act)((w':U)(T w' psi) -> (T w' phi))
          -> (T w (Box a psi)) -> (T w (Box a phi)).
End Proof_Rules.
```

A.3. AN EXAMPLE SESSION IN Coq

We will show a complete Coq session, in which we prove both that $\emptyset;\varphi \vdash \mu x(\neg\varphi \supset x)$ and $\emptyset;\mu x(\neg\varphi \supset x) \vdash \varphi$. These proofs are an example of how one would proceed in working with the encoding, interleaving logical steps with proofs of well-formedness and expansions as needed. Commands entered by the user are written in *this font*.

```
miculan@coltrane:~> coqtop
Welcome to Coq V6.3.1 (December 1999)

Coq < Require mu.
[Reinterning mu ...done]
(* An example *)
Lemma ex1 : (phi:wfo)(w:U)
            (T w (prp phi)) <->
            (T w (mu [x:var](Imp (Not (prp phi)) (Var x)))).

ex1 < (Intros;Split;Intro).

(* first direction: -> *)
ex1 < Apply mu_I; Intros; Apply Imp_I; Intro.
(* now the goal is
  phi : wfo
  w : U
  H : (T w (prp phi))
  z : var
  H0 : (bind z (mu [x:var](Imp (Not (prp phi)) (Var x))))
  H1 : (T w (Not (prp phi)))
  ============================
   (T w (Var z))
  we replace z with the corresponding term:
*)

ex1 < Apply expto_L with (mu [x:var](Imp (Not (prp phi)) (Var x)));
      Auto.
(*
  phi : wfo
  w : U
  H : (T w (prp phi))
  z : var
  H0 : (bind z (mu [x:var](Imp (Not (prp phi)) (Var x))))
  H1 : (T w (Not (prp phi)))
  ============================
   (expto (Var z) (mu [x:var](Imp (Not (prp phi)) (Var x))))
*)

ex1 < Apply expto_bind; Assumption.
(*
  phi : wfo
  w : U
  H : (T w (prp phi))
  z : var
  H0 : (bind z (mu [x:var](Imp (Not (prp phi)) (Var x))))
  H1 : (T w (Not (prp phi)))
  ============================
   (T w (mu [x:var](Imp (Not (prp phi)) (Var x))))
*)
ex1 < Apply ff_E; Apply Not_E with phi:=(prp phi).
(* this generates three goals:
  phi : wfo
  w : U
  H : (T w (prp phi))
  z : var
  H0 : (bind z (mu [x:var](Imp (Not (prp phi)) (Var x))))
  H1 : (T w (Not (prp phi)))
  ============================
   (iswf (prp phi))

subgoal 2 is:
 (T w (prp phi))
subgoal 3 is:
 (T w (Not (prp phi)))

 they are immediate:
*)
ex1 < Exact (cnd phi). Assumption. Assumption.
```

```
(* second direction <- : the goal is as follows:
  phi : wfo
  w : U
  H : (T w (mu [x:var](Imp (Not (prp phi)) (Var x))))
  ============================
   (T w (prp phi))
*)
ex1 < Apply mu_E with [x:var](Imp (Not (prp phi)) (Var x));
  [Simpl; Intros; Split | Intros | Assumption].

ex1 < Split; [Exact (cnd phi) | Trivial].
(* Now we face a huge and painful term - but don't worry *)
ex1 < Intro; Split; Auto;
  Apply notin_posin; Elim (neverempty x); Intros;
  Apply proj1 with B:=(not (eq ? x x0)); Apply H0; Assumption.

(* the goal does not depend on w and assumption H any more:
  phi : wfo
  w : U
  H : (T w (mu [x:var](Imp (Not (prp phi)) (Var x))))
  z : var
  H0 : (notin z (mu [x:var](Imp (Not (prp phi)) (Var x))))
  H1 : (bind z (prp phi))
  w' : U
  H2 : (T w' (Imp (Not (prp phi)) (Var z)))
  ============================
   (T w' (prp phi))
 so we can drop them *)

ex1 < Clear H w.

(* now we have completely changed the sequent:
  phi : wfo
  z : var
  H0 : (notin z (mu [x:var](Imp (Not (prp phi)) (Var x))))
  H1 : (bind z (prp phi))
  w' : U
  H2 : (T w' (Imp (Not (prp phi)) (Var z)))
  ============================
   (T w' (prp phi))
*)

ex1 < Apply RAA; Intros; Apply Not_E with phi:=(prp phi);
      [Exact (cnd phi) | Idtac | Assumption].

(*
  phi : wfo
  z : var
  H0 : (notin z (mu [x:var](Imp (Not (prp phi)) (Var x))))
  H1 : (bind z (prp phi))
  w' : U
  H2 : (T w' (Imp (Not (prp phi)) (Var z)))
  H : (T w' (Not (prp phi)))
  ============================
   (T w' (prp phi))

  Now we replace phi with z, in order to apply H1 *)

ex1 < Apply expto_R with (Var z).

(* the first goal is an expansion *)
ex1 < Apply expto_bind; Assumption.

(* the second is
  phi : wfo
  z : var
  H0 : (notin z (mu [x:var](Imp (Not (prp phi)) (Var x))))
  H1 : (bind z (prp phi))
  w' : U
  H2 : (T w' (Imp (Not (prp phi)) (Var z)))
  H : (T w' (Not (prp phi)))
  ============================
   (T w' (Var z))

 which follows from H2
*)
ex1 < Apply Imp_E with phi:=(Not (prp phi)); Try Assumption.
ex1 < Exact (cnd phi).
ex1 < Qed.
```

ACKNOWLEDGMENT

The author is grateful to Furio Honsell for fruitful discussions and to the anonymous referees for many useful remarks.

REFERENCES

1. A. Avron. Simple Consequence Relations. *Information and Computation*, 92:105–139, 1991.
2. A. Avron, F. Honsell, I. A. Mason, and R. Pollack. Using Typed Lambda Calculus to implement formal systems on a machine. *Journal of Automated Reasoning*, 9:309–354, 1992.
3. A. Avron, F. Honsell, M. Miculan, and C. Paravano. Encoding modal logics in Logical Frameworks. *Studia Logica*, 60(1):161–208, 1998.
4. M. Benevides and T. Maibaum. A constructive presentation for the modal connective of necessity (□). *J. Logic Computat.*, 2(1):31-50, 1992.
5. *The Coq Proof Assistant Reference Manual - Version 6.2.* INRIA, May 1998. Available at ftp://ftp.inria.fr/INRIA/coq/V6.2/doc.

6. A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

7. G. D'Agostino, M. Hollenberg. Logical questions concerning the μ-calculus: interpolation, Lyndon, and Łoś-Tarski. To be published in the *JSL*, 2000.

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

9. J. Despeyroux and P. Leleu. Primitive recursion for higher-order abstract syntax with dependant types. In *Proc. of FLoC'99 IMLA workshop*, 1999.

10. M. Fiore, G. Plotkin and D. Turi. Abstract syntax and variable binding. In *Proc. of 14^{th} LICS*, pages 193–202. IEEE, 1999.

11. M. Gabbay and A. Pitts. A new approach to abstract syntax involving binders. In *Proc. of 14^{th} LICS*, pages 214–224. IEEE, 1999.

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

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

14. M. Hofmann. Semantical analysis of higher-order abstract syntax. In *Proc. of 14^{th} LICS*, pages 204–213. IEEE, 1999.

15. F. Honsell and M. Miculan. A natural deduction approach to dynamic logics. In S. Berardi and M. Coppo, editors. *Proceedings of TYPES'95*, LNCS 1158, pages 165–182. Springer-Verlag, 1996.

16. F. Honsell, M. Miculan, I. Scagnetto. π-calculus in (Co)Inductive Type Theory. To appear in *Theoretical Computer Science*, 2000. Also available at `ftp://ftp.dimi.uniud.it/pub/miculan/TCS99.ps.gz`

17. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

18. *The LEGO proof assistant.* Department of Computer Science, University of Edinburgh. `http://www.dcs.ed.ac.uk/home/lego`

19. 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.

20. 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.

21. M. Miculan. *Encoding Logical Theories of Programs.* PhD thesis TD-7/97, Dipartimento di Informatica, Università di Pisa, Italy, Mar. 1997.

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

23. B. Nordström, K. Petersson, and J. M. Smith. Programming in Martin-Löf's Type Theory: An Introduction. Oxford University Press, 1990.

24. C. Paulin-Mohring. Inductive definitions in the system Coq; rules and properties. In *Proc. TLCA'93*, LNCS 664, pages 328–345. Springer-Verlag, 1993.

25. Frank Pfenning. Logical frameworks. In *Handbook of Automated Reasoning.* Elsevier, 1999. In preparation.

26. A. Simpson. *The proof theory and semantics of intuitionistic modal logic.* PhD thesis, LFCS, Univ. of Edinburgh, 1994.

27. C. Stirling. Modal and Temporal Logics. In *Handbook of Logic in Computer Science*, vol. 2, pages 477–563. Oxford University Press, 1992.

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

29. T. Uustalu. *Natural Deduction for Intuitionistic Least and Greatest Fixedpoint Logics, with an Application to Program Construction.* PhD thesis, Dissertation TRITA-IT AVH 98:03, Dept. of Teleinformatics, Royal Inst. of Technology, Stockholm. May 1998.

30. S. Yu and Z. Luo. *Implementing a model checker for Lego.* In *Proc. of the 4th Inter Symp. of Formal Methods Europe, FME'97.* LNCS 1313, Springer-Verlag, 1997.

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