A Natural Deduction Approach to Dynamic Logic*

Furio Honsell[1] and Marino Miculan[1,2]

[1] Dipartimento di Matematica e Informatica, Università di Udine
Via delle Scienze 206, I-33100 Udine, Italy. {honsell,miculan}@dimi.uniud.it
[2] Dipartimento di Informatica, Università di Pisa
Corso Italia 40, I-56100 Pisa, Italy. miculan@di.unipi.it

Abstract. Natural Deduction style presentations of program logics are useful in view of the implementation of such logics in interactive proof development environments, based on type theory, such as LEGO, Coq, etc. In fact, ND-style systems are the kind of systems which can take best advantage of the possibility of reasoning "under assumptions" offered by proof assistants generated by Logical Frameworks. In this paper we introduce and discuss sound and complete proof systems in Natural Deduction style for representing various "truth" consequence relations of Dynamic Logic. We discuss the design decisions which lead to adequate encodings of these logics in Coq. We derive in Dynamic Logic a set of rules representing a ND-style system for Hoare Logic.

Introduction

Computerized proof assistants are very useful, and probably necessary, in using logical systems for reasoning about programs. In fact, the amount of (often trivial and repetitive) routine details involved in using program logics renders error-prone the activity of a human prover.

Type Theories, such as the Edinburgh Logical Framework [9, 3] or the Calculus of Inductive Constructions [5, 27] were especially designed, or can be fruitfully used, as a general logic specification language, i.e. as a Logical Framework (LF). Thus they can streamline the process of generating interactive proof development environments tailored to the peculiarities of any given logics. In fact, any interactive proof development environment for these type theories (LEGO [16], Coq [14] and ELF [21]), can be readily turned into one for a specific logic, as soon as we fix a suitable environment corresponding to the encoding of the logic. Although these editors are not as efficient as some of those especially designed for a specific logic, nevertheless Logical Frameworks can be very useful for at least three reasons. First of all, they provide a common medium for integrating different systems. Hence LF-derived editors rival special purpose

* Work partially supported by the Esprit BRP no.6453, *Types for Proofs and Programs*, and italian MURST 40%-60% grants. Some of the results of this papers have been communicated by the second author at the TYPES Annual Meeting in Båstad, 1994.

editors when efficiency can be increased by integrating independent logical systems. Secondly, LF-generated editors are *natural.* A user of the original logic can transfer immediately to them his practical experience and "trade tricks,". They do not force upon the user the overhead of unfamiliar indirect codings, as would editors derived from FOL editors, via an encoding. On the contrary, it is a frequent experience that encodings in Logical Frameworks provide the "ultimate" or "normative" formalization of the logical system under consideration. The specification methodology of Logical Frameworks, in fact, forces the user to make precise all tacit conventions. Thirdly, Logical Frameworks are based on Type Theory presented in Natural Deduction style via the analogy "judgements as types". Therefore, they naturally allow the user of an LF-generated editor to reason "under assumptions" and go about in developing a proof the way mathematicians normally reason: using hypotheses, formulating conjectures, storing and retrieving lemmata, often in top-down, goal-directed fashion. This feature offered by Logical Frameworks urges the designer/implementor of an editor for a given object logic, to look for a presentation of the logic which can take best advantage of the possibility of manipulating assumptions.

The crucial concept involved in discussing the notion of assumption for a given logic is that of *consequence relation* (CR) [2]. CR's are abstract representations of logical dependencies between assumptions and conclusions. They play a crucial rôle in stating and proving adequacies of encodings in Logical Frameworks. Usually, a logic gives rise to more than one CR. For instance, in FOL we have the *validity* CR and the *truth* CR, according to how we understand free variables in assumptions. In *modal logics* further CR's arise, according to whether we focus on *frames* or *worlds.* Usually, CR's differ on the form of "deduction theorem" that they yield. Truth CR's are those which yield the simplest deduction theorems. Validity CR's are best suited for capturing the notion of *derivabilty* from sets of *axioms* and hence the notion of *theoremhood.* Many more different CR's can be defined for program logics if we take into account the possibility of restricting attention to interesting subclasses of models.

Before building an editor for a given logic, the designer/implementor has to clarify two equally important, apparently orthogonal, issues. Which CR is the one to focus on? Which style of presentation is best for actually "using" the logic, e.g. Hilbert, Natural Deduction (ND) or Gentzen (sequent) style? In the methodology of Logical Frameworks, answering the first question amounts to decide which *judgements* to encode. Experience shows that ND-style systems representing truth CR's, are best suited for exploiting the reasoning power of assumptions provided by Logical Frameworks.

In this paper we investigate logical systems for Dynamic Logic (DL) in view of their encoding in the interactive proof development environment Coq. Similarly to what happens for FOL, both validity and truth CR's arise for DL, and program logics in general. Sound and (relative) complete systems for representing the validity CR's restricted to *finite* sets of assumptions, can be readily derived from any Hilbert systems for DL [1, 8, 15]. But, surprisingly, little attention has been payed so far, in the literature, to "truth CR's" for program logics. Hence, in line with the remark above, in this paper we introduce and discuss Natural Deduction style systems for representing truth CR's.

In developing ND-style systems for DL, one of the most delicate and complicated issues one has to deal with, is that of free logical variables versus program identifiers. New difficulties arise when we consider derivations under assumptions, since assumptions on program variables enforce local constraints on the environments of subderivations. However, the type-theoretic metatheory provided by Logical Frameworks, which allows to express schematic (i.e. generalized) assumptions, provides interesting solutions to these difficulties. Another problematic issue arises in connection with "infinitary rules". Logical Frameworks, such as Coq, offer also in this case a remarkable metatheoretic solution. Since they embody the power of a higher order intuitionistic logic of inductive definitions, many recursive functions can be defined in them. Other difficulties arise in connection with "rules of proof" (i.e. rules which can be applied only to premises which depend on *no* assumptions), or with "proper sequent-style" rules, such as Scott's rule (i.e. rules which modify substantially the structure of *all* the assumptions). Some of these problematic side conditions in the rules can be internalized in the framework at the expense of a slight modification of the basic judgment, exploiting again the possibility of using schematic premises in rules.

In the encodings, we exploit thoroughly the higher-order features provided by Logical Frameworks.

In this paper, for the sake of simplicity we consider only the datatype of natural numbers.

The paper is organized as follows. In Section 1 we introduce sound and complete ND-style systems with respect to the truth CR for Dynamic Logic (DL) over the first order language of Peano Arithmetic. Encodings of these systems in type theory are given in Section 2. In Section 3 we describe the derivation of an impure ND-style system for the truth CR of Hoare Logic. In Section 4 we compare our work with the *KIV System* (which is a special purpose editor for DL [23]) and an implementation of Hoare Logic in the Cambridge HOL [6]. Final remarks appear in Section 5. In Appendix A we give the syntax and semantics of Peano Arithmetic (PA), Hoare Logic and DL. In Appendix B we give the notion of *Consequence Relation* and related basic logical notions. Throughout the paper we use standard proof theoretic notions and notations (see e.g. [22]). Terminology and notations concerning Logical Frameworks are as in [9].

The Coq code of these implementations and examples is available at the URL `http://www.dimi.uniud.it/~miculan/DL`. The authors are grateful to the referees for their useful remarks on an earlier version of the paper.

1 ND-style Proof Systems for Dynamic Logic

Dynamic Logic (see App.A for definitions) has been thoroughly investigated from the model theoretic point of view. Not as much attention, however, has been payed to its proof theory or to the possibility of representing consequence relations different from that of *validity*. The relevant concept being that of *theoremhood*, the proof systems considered have been mainly Hilbert-style systems [7, 8, 15, 26]. There is only one remarkable exception, albeit unpublished, of ND-style System for Deterministic DL due to C. Stirling [25] (see Sect.5).

Besides absolute validity and absolute truth, various CR's can be introduced according to the class of models that one focuses on. Since in this paper we focus on the language of Peano Arithmetic (PA), we consider two classes of models: the class of *all* first-order structures which are models of PA, and that consisting only of the standard model (denoted by $\mathbb{N}$). Truth and validity CR's for DL are defined by suitably specializing the following general definition:

Definition 1 Truth and Validity on First-Order Structures. Let L be a first-order language, and let Γ range over sets of formulæ, p over formulæ of L.

1. Let $\mathcal{M}$ be a first-order model for L (see Appendix A);
   – the *truth CR* $\models^L_{\mathcal{M}}$ *wrt* $\mathcal{M}$ is the relation defined by

$$\Gamma \models^L_{\mathcal{M}} p \iff [\![\Gamma]\!]_{\mathcal{M}} \subseteq [\![p]\!]_{\mathcal{M}};$$

   – the *validity CR* $\Vdash^L_{\mathcal{M}}$ *wrt* $\mathcal{M}$ is the relation defined by

$$\Gamma \Vdash^L_{\mathcal{M}} p \iff ([\![\Gamma]\!]_{\mathcal{M}} = \mathbb{S}_{\mathcal{M}} \Rightarrow [\![p]\!]_{\mathcal{M}} = \mathbb{S}_{\mathcal{M}}).$$

2. The *(absolute) truth CR* is the relation $\models^L \stackrel{\text{def}}{=} \bigcap_{\mathcal{M}} \models^L_{\mathcal{M}}$; the *(absolute) validity CR* is the relation $\Vdash^L \stackrel{\text{def}}{=} \bigcap_{\mathcal{M}} \Vdash^L_{\mathcal{M}}$, where $\mathcal{M}$ ranges over all first-order models for L. □

We introduce a ND-style system for $\models$, $\mathcal{S}_{\text{ND}}(\text{DL})$, by adding to the usual ND-style system for Peano Arithmetic [22] the rules in Fig.1. By p^t_x we denote the formula obtained by replacing all occurrences of x, which are not bound by the $\forall$-quantifier, with t (possibly α-converting p in order to avoid capturing free variables in t). The set of variables in p whose occurrences are not all bound by $\forall$ is denoted by $\text{FV}(p)$. We write Natural Deduction rules and proofs in the *linearized* notation, hence "$\pi : \Gamma \vdash p$" denotes a proof tree π whose premises and conclusion are Γ and p respectively. The system is *infinitary* system, i.e. Γ is possibly an infinite set.

The system $\mathcal{S}_{\text{ND}}(\text{DL})$ is sound and complete with respect to the truth CR:

Theorem 2. $\forall \Gamma, p : \Gamma \vdash_{\mathcal{S}_{\text{ND}}(\text{DL})} p \iff \Gamma \models p.$

A proof can be obtained by modifying suitably the proof of Theorem 3.15 in [8]; see [20] for further details.

The system $\mathcal{S}_{\text{ND}}(\text{DL})$ is indeed a ND-style system, since there are introduction rules for each program constructor and the corresponding elimination rules are induced by the introduction rules. The rules for equality and the quantifier are more involved than the usual ones for FOL, due to the presence of commands. Reflexivity of equality can be encoded immediately, but the rules of congruence have to be rephrased with care: derivations like $[x := 0]\,(x = 0), x =$

$$:=\text{-I}\ \frac{\Gamma, y = t \vdash p^y_x}{\Gamma \vdash [x := t]\, p}\ y \notin \mathrm{FV}(\Gamma, p, t) \qquad :=\text{-E}\ \frac{\Gamma_1 \vdash [x := t]\, p \quad \Gamma_2, p^y_x, y = t \vdash q}{\Gamma_1, \Gamma_2 \vdash q}\ y \notin \mathrm{FV}(\Gamma_2, p, q, t)$$

$$;\text{-I}\ \frac{\Gamma \vdash [c_1]\,[c_2]\, p}{\Gamma \vdash [c_1; c_2]\, p} \qquad ;\text{-E}\ \frac{\Gamma \vdash [c_1; c_2]\, p}{\Gamma \vdash [c_1]\,[c_2]\, p}$$

$$^*\text{-I}\ \frac{\{\Gamma_n \vdash [c]^n\, p \mid n \in \mathbb{N}\}}{\cup_n \Gamma_n \vdash [c^*]\, p} \qquad ^*\text{-E}\ \frac{\Gamma \vdash [c^*]\, p}{\Gamma \vdash [c]^n\, p}\ n \in \mathbb{N} \qquad \text{where } [c]^0\, p = p,\ [c]^{n+1}\, p = [c]\,[c]^n\, p$$

$$?\text{-I}\ \frac{\Gamma, b \vdash p}{\Gamma \vdash [b?]\, p} \qquad ?\text{-E}\ \frac{\Gamma_1 \vdash [b?]\, p \quad \Gamma_2 \vdash b}{\Gamma_1, \Gamma_2 \vdash p}$$

$$+\text{-I}\ \frac{\Gamma_1 \vdash [c_1]\, p \quad \Gamma_2 \vdash [c_2]\, p}{\Gamma_1, \Gamma_2 \vdash [c_1 + c_2]\, p} \qquad +\text{-E}\ \frac{\Gamma \vdash [c_1 + c_2]\, p}{\Gamma \vdash [c_i]\, p}$$

$$\forall\text{-I}\ \frac{\Gamma \vdash p}{\Gamma \vdash \forall x p}\ x \notin \mathrm{FV}(\Gamma) \qquad \forall\text{-E}\ \frac{\Gamma_1 \vdash \forall x p \quad \Gamma_2, p^y_x, y = t \vdash q}{\Gamma_1, \Gamma_2 \vdash q}\ y \notin \mathrm{FV}(\Gamma_2, \forall x p, t, q)$$

$$\textsc{CongrId}\ \frac{\Gamma_1 \vdash p \quad \Gamma_2 \vdash x = y}{\Gamma_1, \Gamma_2 \vdash p^y_x}\ y \notin \mathrm{FV}(p) \qquad \textsc{Congr}\ \frac{\Gamma_1 \vdash p^{t_1}_x \quad \Gamma_2 \vdash t_1 = t_2}{\Gamma_1, \Gamma_2 \vdash p^{t_2}_x}\ p \text{ is command-free}$$

Fig. 1. The system $\mathcal{S}_{\text{ND}}(\text{DL})$.

$1 \vdash [x := 0]\,(1 = 0)$ have to be prevented. To this end, we introduce two rules: Congr and CongrId. Congr can be applied only to command-free formulæ, i.e. formulæ where no command appears (see App.A). CongrId, can be applied to any formula, since it merely replaces all occurrences of an identifier with a new identifier.

The non traditional form of $\forall$-elimination is due to the fact that, in general, the quantified formula p may contain commands, and therefore not all occurrences of a bound variable can be replaced by a term. For instance, $\forall x. [x := 0]\,(x = 0)$ holds, but its naïve instantiation $[1 := 0]\,(1 = 0)$ is clearly meaningless. A correct formulation of instantiation of quantified variables is in fact one of the most difficult technical issues to deal with in encoding DL. In Hilbert systems this is usually achieved by replacing, whenever required, any program c with the equivalent "normal form" $z_1 := x_1; \ldots; z_n := x_n; c'; x_1 := z_1; \ldots; x_n := z_n$ where the x_i's are all the identifiers appearing in c, the z_i's are fresh and c' is obtained from c by replacing the x_i's with z_i's (see [8]). This solution is clearly cumbersome if we want to use practically the formal system. The problematic nature of instantiation of quantifiers lies, as in the case of the congruence rules, in the different nature of pure logical identifiers and program variables. In fact, the property "$s \in [\![p^t_x]\!] \iff s[x \mapsto [\![t]\!]s] \in [\![p]\!]$" does not hold for DL.

Our solution to the instantiation problem is to replace the bound variable x with a fresh variable y, and to assume $y = t$ in the minor premise. The usual $\forall$-elimination rule is derivable in the case of command-free predicates.

The infinitary nature of rule *-I is essential for achieving the completeness of $\vdash_{\mathcal{S}_{\text{ND}}(\text{DL})}$ with respect to the full $\models$ and not only to $\models \cap(\mathcal{P}_{<\omega}(\mathbb{P}) \times \mathbb{P})$. In fact, proofs in finitary systems can take into account only a *finite* number of assumptions, and since DL does not satisfy compactness (consider e.g. the set $\{[x := x-1]^n\, x \neq 0 \mid n \in \mathbb{N}\} \cup \{\neg\,[(x := x-1)^*]\, x \neq 0\}$), we can easily find a true consequence which is underivable in any finitary system (e.g. $\{[x := x-1]^n\, x \neq 0 \mid n \in \mathbb{N}\} \models [(x := x-1)^*]\, x \neq 0$).

The useful, albeit "impure", $[\cdot]$-intro rule $\frac{\emptyset \vdash p}{\emptyset \vdash [c]p}$ and Scott's rule $\textsc{Sc}\frac{\Gamma \vdash p}{[c]\Gamma \vdash [c]p}$ (where $[c]\,\Gamma \stackrel{\text{def}}{=} \{[c]\,p \mid p \in \Gamma\}$) are clearly admissible for $\mathcal{S}_{\text{ND}}(\text{DL})$.

Focusing on consequences true in all models of Peano Arithmetic, the system $\mathcal{S}_{\text{ND}}(\text{DL})$ rules out many interesting consequences which are true when reasoning about real programs which utilize as datatype the real integers. For instance, the formula $p \stackrel{\text{def}}{=} \langle (x := x-1)^* \rangle\,(x=0)$ is not valid: take any nonstandard model $\mathcal{N}^*$, and consider the state s such that $s(x) = \nu$, ν a nonstandard integer; then, $s \notin [\![p]\!]_{\mathcal{N}^*}$. The same happens with the **while**-termination formula $\langle \textbf{while}\ x > 0\ \textbf{do}\ x := x-1 \rangle\,(x=0)$. This is the reason for focusing on the sole standard model of arithmetic and the associated CR's $\models\!\!\!=_{\mathbb{N}}$, $\models_{\mathbb{N}}$.

In order to represent $\models\!\!\!=_{\mathbb{N}}$, we extend the standard ND-style system $\mathcal{S}_{\text{ND}}(\text{DL})$ to a hybrid Natural Deduction-Modal system, namely $\mathcal{S}^a_{\text{ND}}(\text{DL})$, by adding either the *convergence rule* or the equivalent dual *induction principle rule*:

$$\textsc{Conver}\ \frac{\emptyset \vdash p^{x+1}_x \supset \langle c \rangle\, p \quad \Gamma \vdash p^t_x}{\Gamma \vdash \langle c^* \rangle\, p^0_x}\ x \notin \text{FV}(c)$$

$$\textsc{Induc}\ \frac{\emptyset \vdash [c]\, p \supset p^{x+1}_x \quad \Gamma \vdash [c^*]\, p^0_x}{\Gamma \vdash p^t_x}\ x \notin \text{FV}(c)$$

Both rules are "impure" in the sense of Avron [2], and are proof-rules, since the first premise is a theorem. One can easily see that $\vdash_{\mathcal{S}^a_{\text{ND}}(\text{DL})} \langle (x := x-1)^* \rangle\,(x = 0)$ and $\vdash_{\mathcal{S}^a_{\text{ND}}(\text{DL})} \langle \textbf{while}\ x > 0\ \textbf{do}\ x := x-1 \rangle\,(x=0)$. Indeed, $\mathcal{S}^a_{\text{ND}}(\text{DL})$ is sound and complete with respect to the standard model of integers.

Theorem 3. $\forall \Gamma, p : \Gamma \vdash_{\mathcal{S}^a_{\text{ND}}(\text{DL})} p \iff \Gamma \models\!\!\!=_{\mathbb{N}} p$.

A proof can be readily derived from that of Th.2.

It is interesting to notice that the rule *-I is enough to recover the full power of the ω-rule of infinitary first order logic:

Theorem 4. *Let p be any command-free formula; then, the ω-rule $\frac{\{\Gamma \vdash p^n_x \mid n \in \omega\}}{\forall x p}$ is derivable in $\mathcal{S}^a_{\text{ND}}(\text{DL})$.*

Proof. (Sketch) The proof relies upon the fact that command iteration is nondeterministic, hence $\forall x p$ is equivalent to $y = 0 \supset [y := y + 1^*]\, p^y_x$ (y fresh). Each premise p^n_x in the ω-rule can be rendered by means of the formula $y = 0 \supset [y := y+1]^n\, p^y_x$ (y fresh); applications of *-I and $\textsc{Induc}$ yield the ω-rule. $\square$

Instead of introducing proof rules, we could have used alternatively *non-interference* judgments *à la* Reynolds [24] as side conditions of the rules. These are judgments which generalize side-conditions such as $x \notin \text{FV}(A)$. See [20] for further details.

2 Encoding ND-style Systems for DL

In this section we apply and generalize the methodology developed in [9, 3] and define an encoding of $\mathcal{S}_{\text{ND}}(\text{DL})$ and of $\mathcal{S}^a_{\text{ND}}(\text{DL})$ within the Calculus of Inductive Constructions, as it is implemented by the Coq V5.10 proof assistant [14].

$\mathsf{X}, \mathsf{Te}, \mathsf{B}, \mathsf{C}, \mathsf{P}$	$: \mathsf{Set}$	isId	$: \mathsf{X} \to \mathsf{Te}$	$[\cdot]\,\cdot$	$: \mathsf{C} \to \mathsf{P} \to \mathsf{P}$
$\neg_b$	$: \mathsf{B} \to \mathsf{B}$	$0, 1$	$: \mathsf{Te}$	*	$: \mathsf{C} \to \mathsf{C}$
$\supset_b, \wedge_b$	$: \mathsf{B} \to \mathsf{B} \to \mathsf{B}$	$+, *$	$: \mathsf{Te} \to \mathsf{Te} \to \mathsf{Te}$	$?$	$: \mathsf{B} \to \mathsf{C}$
$\neg$	$: \mathsf{P} \to \mathsf{P}$	$=_b, <_b$	$: \mathsf{Te} \to \mathsf{Te} \to \mathsf{B}$	$;, +$	$: \mathsf{C} \to \mathsf{C} \to \mathsf{C}$
$\supset, \wedge$	$: \mathsf{P} \to \mathsf{P} \to \mathsf{P}$	$=, <$	$: \mathsf{Te} \to \mathsf{Te} \to \mathsf{P}$	$:=$	$: \mathsf{X} \to \mathsf{Te} \to \mathsf{C}$

Fig. 2. Representation of $\mathcal{L}(\mathrm{DL})$ in $\Sigma(\mathrm{DL})$ (some constructors).

An important difference with respect to the encoding of HOL in [9] is that we can no longer treat on a par object language identifiers and metalanguage schematic variables (see [3] for similar difficulties in handling Hoare Logic). In fact, the presence of identifiers in formulæ standing for left-hand values which cannot be substituted for, forces us to introduce a specific type for identifiers. Therefore, substitutions of terms for identifiers cannot be handled any mor "for free" by the metalanguage, using *higher order syntax*. Nevertheless, we can still handle at the metalevel substitution of identifiers for identifiers.

2.1 The Encoding of $\mathcal{S}_{\mathrm{ND}}(\mathrm{DL})$: the Signature $\Sigma(\mathbf{DL})$

Syntax. Each syntactic category is represented by an inductive set (denoted by the same name in this font), and each syntactic constructor is represented by a functional constant (Fig.2). There is also a function $\mathsf{b2p} : \mathsf{B} \to \mathsf{P}$, defined by induction on the syntax, which embeds propositional formulæ into formulæ. When clear form the context, it will be omitted for sake of readability. Applications of b2p are computable (Simplifable) in the Coq environment.

Let $\xi : \mathbb{B} \cup \mathbb{X} \cup \mathbb{T} \cup \mathbb{C} \cup \mathbb{P} \to \mathsf{B} \cup \mathsf{X} \cup \mathsf{T} \cup \mathsf{C} \cup \mathsf{P}$ be the compositional bijective representation of syntactic classes. For the sake of simplicity, ξ will be often omitted; therefore, with the same term we will denote a formula as well as its encoding in the LF signature; similarly we shall deal with sets of assumptions.

We represent the universal quantifier by the syntactic constructor $\forall : (\mathsf{X} \to \mathsf{P}) \to \mathsf{P}$ and hence we can take care of α-conversion of bound variables at the metalevel. Consequently, $\xi(\forall x p) = \forall(\lambda x.\xi(p))$, and, for instance, $\forall x\, [x := 0]\, (x = 0)$ is represented by $\forall(\lambda x : \mathsf{X}.\, [x := 0]\, (\mathsf{isId}(x) = 0))$.

Rules. Since $\mathcal{S}_{\mathrm{ND}}(\mathrm{DL})$ is in ND-style, most of the rules are encoded straightforwardly following the methodology of [9, 3], using as judgment $\mathsf{T} : \mathsf{P} \to \mathsf{Prop}$ (Fig.3). The intended meaning of $(\mathsf{T}\ p)$ is that the formula p holds.

In the following, we will briefly discuss some interesting points concerning the encoding of the most complex rules.

The infinitary rule *-I. Due the presence of *-I, the system $\mathcal{S}_{\mathrm{ND}}(\mathrm{DL})$ has to take into account infinite sets of premises. Hence we need to be able to refer to infinite sets of formulæ. We represent infinte sets of assumptions by a Coq term of type $\mathsf{nat} \to \mathsf{Prop}$. Thus, the version of the rule *-I we encode in Coq is the following:

$$^*\text{-I}\ \frac{\text{for all } n \in \mathbb{N} : I(c,p,n)}{[c^*]\, p} \quad \text{where} \quad \begin{array}{l} I : \mathbb{C} \to \mathbb{P} \to \mathbb{N} \to \mathbb{P} \\ I(c,p,0) = p, \quad I(c,p,n+1) = [c]\, I(c,p,n) \end{array}$$

$$\wedge\text{-I}: \prod_{p,q:\mathsf{P}} (\mathsf{T}\ p) \to (\mathsf{T}\ q) \to (\mathsf{T}\ (p \wedge q)) \qquad ;\text{-I}: \prod_{p:\mathsf{P}} \prod_{c_1,c_2:\mathsf{C}} (\mathsf{T}\ [c_1]\,[c_2]\,p) \to (\mathsf{T}\ [c_1; c_2]\,p)$$

$$\supset\text{-I}: \prod_{p,q:\mathsf{P}} ((\mathsf{T}\ p) \to (\mathsf{T}\ q)) \to (\mathsf{T}\ (p \supset q)) \quad ;\text{-E}: \prod_{p:\mathsf{P}} \prod_{c_1,c_2:\mathsf{C}} (\mathsf{T}\ [c_1; c_2]\,p) \to (\mathsf{T}\ [c_1]\,[c_2]\,p)$$

$$^*\text{-I}: \prod_{p:\mathsf{P}} \prod_{c:\mathsf{C}} \left(\prod_{n:\mathsf{nat}} (\mathsf{T}\ (\mathsf{I}\ c\ p\ n)) \right) \to (\mathsf{T}\ [c^*]\,p) \qquad \text{where } (\mathsf{I}\ c\ p\ 0) = p,$$

$$^*\text{-E}: \prod_{p:\mathsf{P}} \prod_{c:\mathsf{C}} (\mathsf{T}\ [c^*]\,p) \to \prod_{n:\mathsf{nat}} (\mathsf{T}\ (\mathsf{I}\ c\ p\ n)) \qquad (\mathsf{I}\ c\ p\ (S\ n)) = [c]\,(\mathsf{I}\ c\ p\ n)$$

Fig. 3. Representation of some rules of $\mathcal{S}_{\mathrm{ND}}(\mathrm{DL})$ in the signature $\Sigma(\mathrm{DL})$.

$$:=\text{-I}: \prod_{A:\mathsf{X}\to\mathsf{P}} \prod_{x:\mathsf{X}} \prod_{t:\mathsf{Te}} \left(\prod_{y:\mathsf{X}} (\mathsf{isnotin}\ y\ \mathsf{P}\ \forall A) \to (\mathsf{isnotin}\ y\ \mathsf{Te}\ t) \to (\mathsf{T}\ (y = t)) \to (\mathsf{T}\ (A\ y)) \right)$$
$$\to (\mathsf{isnotin}\ x\ \mathsf{P}\ \forall A) \to (\mathsf{T}\ ([x := t](A\ x)))$$

$$:=\text{-E}: \prod_{A:\mathsf{X}\to\mathsf{P}} \prod_{q:\mathsf{P}} \prod_{x:\mathsf{X}} \prod_{t:\mathsf{Te}} \left(\prod_{y:\mathsf{X}} (\mathsf{isnotin}\ y\ \mathsf{P}\ \forall A) \to (\mathsf{isnotin}\ y\ \mathsf{Te}\ t) \to (\mathsf{isnotin}\ y\ \mathsf{P}\ q) \to \right.$$
$$\left. (\mathsf{T}\ (y = t)) \to (\mathsf{T}\ (Ay)) \to (\mathsf{T}\ q) \right) \to (\mathsf{isnotin}\ x\ \mathsf{P}\ \forall A) \to$$
$$(\mathsf{T}\ ([x := t]\,(A\ x))) \to (\mathsf{T}\ q)$$

Fig. 4. The LF encoding of the rules for assignment.

Therefore, using this encoding, we can refer only to premises which can be enumerated by a function provably total in PA^ω, pratically, this is more than enough.

The assignment rules. As remarked earlier, we cannot exploit higher-order syntax directly to encode $()^t_x$, the substitution operator, as was possible in [3, 9, 18]. The naïve encoding of the assignment constructor, $:=: \mathsf{Te} \to \mathsf{Te} \to \mathsf{C}$, could yield meaningless commands such as $0 := 1$. Substitution has to be dealt with differently from [9], rather in the style of [4]. The encodings of the rules :=-I and :=-E appear in Fig.4. We need to express the fact that an identifier is "fresh", i.e. that it is different from any other pre-existing identifier. To this end, we generalize Mason's idea [3] later expounded in [4, 19], and we introduce the two auxiliary judgments, $\mathsf{isin}, \mathsf{isnotin} : \mathsf{X} \to \prod_{A:\mathsf{Set}} A \to \mathsf{Prop}$. The intuitive meaning of $(\mathsf{isin}\ x\ A\ a)$ is "the identifier x appears in the phrase a whose type is A;" dually for $\mathsf{isnotin}$. These two judgments are derivable by means of a simple set of rules which are polymorphic in the syntactic constructors (Fig.5). The inference of these judgments is completely syntax-driven: it is sufficient to look at the top-level constructor of the phrase for deciding which rule has to be applied. The premise $(\mathsf{isnotin}\ x\ \mathsf{P}\ \forall A)$ of the :=-I rule enforces the fact that the context $A(\cdot)$ does not contain any occurrence of x. In both rules we have also to reify the "freshness condition" of variables locally quantified in premises. This is achieved by assuming suitable $\mathsf{isnotin}$ judgments. Such reified assumptions are needed to deal with "contexts" such as $A(\cdot)$ above, or the CONGRID rule below.

$$\mathsf{isin\_x} : \prod_{x:\mathsf{X}} (\mathsf{isin}\ x\ \mathsf{X}\ x)$$

$$\mathsf{isin\_1} : \prod_{x:\mathsf{X}} \prod_{s_1,s_2:\mathsf{Set}} \prod_{op:s_1\to s_2} \prod_{p:s_1} (\mathsf{isin}\ x\ s_1\ p) \to (\mathsf{isin}\ x\ s_2\ (op\ p))$$

$$\mathsf{isin\_2l} : \prod_{x:\mathsf{X}} \prod_{s_1,s_2,s_3:\mathsf{Set}} \prod_{op:s_1\to s_2\to s_3} \prod_{p_1:s_1} \prod_{p_2:s_2} (\mathsf{isin}\ x\ s_1\ p_1) \to (\mathsf{isin}\ x\ s_3\ (op\ p_1\ p_2))$$

$$\mathsf{isin\_2r} : \prod_{x:\mathsf{X}} \prod_{s_1,s_2,s_3:\mathsf{Set}} \prod_{op:s_1\to s_2\to s_3} \prod_{p_1:s_1} \prod_{p_2:s_2} (\mathsf{isin}\ x\ s_2\ p_2) \to (\mathsf{isin}\ x\ s_3\ (op\ p_1\ p_2))$$

$$\mathsf{isin\_n} : \prod_{s_1,s_2:\mathsf{Set}} \prod_{op:(\mathsf{X}\to s_1)\to s_2} \prod_{p:X\to s_1} \left(\prod_{y:\mathsf{X}} (\mathsf{isin}\ x\ s_1\ (p\ y)) \right) \to (\mathsf{isin}\ x\ s_2\ (op\ p))$$

$$\mathsf{isnotin\_symm} : \prod_{x,y:\mathsf{X}} (\mathsf{isnotin}\ y\ \mathsf{X}\ x) \to (\mathsf{isnotin}\ x\ \mathsf{X}\ y)$$

$$\mathsf{isnotin\_zero} : \prod_{x:\mathsf{X}} (\mathsf{isnotin}\ x\ \mathsf{Te}\ \mathsf{zero}) \qquad \mathsf{isnotin\_false} : \prod_{x:\mathsf{X}} (\mathsf{isnotin}\ x\ \mathsf{P}\ \mathsf{false})$$

$$\mathsf{isnotin\_1} : \prod_{x:\mathsf{X}} \prod_{s_1,s_2:\mathsf{Set}} \prod_{op:s_1\to s_2} \prod_{p:s_1} (\mathsf{isnotin}\ x\ s_1\ p) \to (\mathsf{isnotin}\ x\ s_2\ (op\ p))$$

$$\mathsf{isnotin\_2} : \prod_{x:\mathsf{X}} \prod_{s_1,s_2,s_3:\mathsf{Set}} \prod_{op:s_1\to s_2\to s_3} \prod_{p:s_1} \prod_{p:s_2} (\mathsf{isnotin}\ x\ s_1\ p_1) \to (\mathsf{isnotin}\ x\ s_2\ p_2) \to (\mathsf{isnotin}\ x\ s_3\ (op\ p_1\ p_2))$$

$$\mathsf{isnotin\_el} : \prod_{x,y:\mathsf{X}} \prod_{s:\mathsf{Set}} \prod_{p:s} (\mathsf{isnotin}\ x\ s\ p) \to (\mathsf{isin}\ y\ s\ p) \to (\mathsf{isnotin}\ y\ \mathsf{X}\ x)$$

$$\mathsf{isnotin\_n} : \prod_{s_1,s_2:\mathsf{Set}} \prod_{op:(\mathsf{X}\to s_1)\to s_2} \prod_{p:X\to s_1} \left(\prod_{y:\mathsf{X}} (\mathsf{isnotin}\ x\ \mathsf{X}\ y) \to (\mathsf{isnotin}\ x\ s_1\ (p\ y)) \right) \to (\mathsf{isnotin}\ x\ s_2\ (op\ p))$$

Fig. 5. The rules for auxiliary judgments isin, isnotin of Σ(DL).

The congruence rules. The encodings of CONGR and CONGRID appear in Fig.6. In encoding CONGRID, we have to check that the context $A(\cdot)$ does not contain any occurrence of x, y. This is enforced as for :=-I, :=-E, via the premises (isnotin $x\ P\ \forall A$) and (isnotin $y\ P\ \forall A$) In encoding CONGR we have to check that the predicate A is command-free. This is easily achieved by introducing a new judgment BF : P $\to$ Prop, whose rules are the following:

$$\mathsf{BF\_false} : (\mathsf{BF}\ \mathsf{false}) \qquad \mathsf{BF\_forall} : \prod_{p:\mathsf{X}\to\mathsf{P}} \left(\prod_{x:\mathsf{X}} (\mathsf{BF}\ (p\ x))\right) \to (\mathsf{BF}\ (\forall p)$$

$$\mathsf{BF\_eq} : \prod_{t_1,t_2:\mathsf{Te}} (\mathsf{BF}\ (t_1 = t_2)) \qquad \mathsf{BF\_and} : \prod_{p,q:\mathsf{P}} (\mathsf{BF}\ p) \to (\mathsf{BF}\ q) \to (\mathsf{BF}\ (p \wedge q))$$

$$\mathsf{BF\_not} : \prod_{p:\mathsf{P}} (\mathsf{BF}\ p) \to (\mathsf{BF}\ (\neg p)) \qquad \mathsf{BF\_imp} : \prod_{p,q:\mathsf{P}} (\mathsf{BF}\ p) \to (\mathsf{BF}\ q) \to (\mathsf{BF}\ (p \supset q))$$

Clearly, derivations of BF are syntax-driven and can be mostly automated in the Coq environment using the Auto tactic.

$$\textsc{CongrId} : \prod_{x,y:\mathsf{X}} \prod_{A:\mathsf{X}\to\mathsf{P}} \prod_{w:U} (\mathsf{isnotin}\ x\ P\ \forall A) \to (\mathsf{isnotin}\ y\ P\ \forall A) \to$$
$$(\mathsf{T}\ (A\ x)) \to (\mathsf{T}\ ((\mathsf{isId}\ x) = (\mathsf{isId}\ y))) \to (\mathsf{T}\ (A\ y))$$

$$\textsc{CongR} : \prod_{t_1,t_2:\mathsf{Te}} \prod_{A:\mathsf{Te}\to\mathsf{P}} (\mathsf{T}\ (A\ t_1)) \to (\mathsf{T}\ (t_1 = t_2)) \to (\mathsf{BF}\ (A\ t_2)) \to (\mathsf{T}\ (A\ t_2))$$

Fig. 6. The LF encoding of the congruence rules.

$$\forall\text{-I} : \prod_{A:\mathsf{X}\to\mathsf{P}} \left(\prod_{x:\mathsf{X}} (\mathsf{isnotin}\ x\ \mathsf{P}\ \forall A) \to (\mathsf{T}\ (A\ x))\right) \to (\mathsf{T}\ (\forall A))$$
$$\forall\text{-E} : \prod_{A:\mathsf{X}\to\mathsf{P}} \prod_{q:\mathsf{P}} \prod_{t:\mathsf{Te}} \left(\prod_{x:\mathsf{X}} (\mathsf{isnotin}\ x\ \mathsf{Te}\ t) \to (\mathsf{isnotin}\ x\ \mathsf{P}\ q) \to (\mathsf{isnotin}\ x\ \mathsf{P}\ \forall A) \to \right.$$
$$\left. (\mathsf{T}\ (x = t)) \to (\mathsf{T}\ (A\ x)) \to (\mathsf{T}\ q)\right) \to (\mathsf{T}\ \forall A) \to (\mathsf{T}\ q)$$

Fig. 7. The LF encoding of the $\forall$-I, $\forall$-E rules.

The $\forall$-quantifier rules. The encoding of the rules for $\forall$ appearing in Fig.7, is not as straightforward as in the standard FOL case. We have to deal with side-conditions and reify "freshness" assumptions on the variables locally quantified in premises, as was the case for the :=-I and :=-E rules.

Adequacy of the encoding. The statement of the Adequacy Theorem for the encoding $\Sigma(\mathrm{DL})$ is more problematic than in the "paradigm case" of FOL [9], since we have to take into account infinite sets of formulæ. Clearly, this cannot be done in full generality and we will be able to state the Adequacy Theorem only with respect to *representable* sets of assumptions, i.e. sets of formulæ whose encodings can be enumerated in Coq. Formally, $\Gamma = \{p_n \mid n \in \mathbb{N}\}$ is *representable (in a context Δ)* if there exists a term G such that $\Delta \vdash_{\Sigma(\mathrm{DL})} G : \mathsf{nat} \to \mathsf{P}$ and for all $n \in \mathbb{N} : \Delta \vdash_{\Sigma(\mathrm{DL})} (G\ \bar{n}) = \xi(p_n)$

Given a representable set of assumptions Γ, in order to define $\gamma(\Gamma)$, the *Coq representation of Γ*, we proceed as follows. First of all, we assume, for each free identifier appearing in Γ, the identifier itself and the judgment asserting that it is different from any other identifier (notice that, for obvious reasons, we are interested in considering only a finite set of identifiers at any given time); we put

$$\iota(\Gamma) \stackrel{\mathrm{def}}{=} \{x : \mathsf{X} \mid x \in \mathrm{FV}(\Gamma)\} \cup \{i_{xy} : (\mathsf{isnotin}\ x\ \mathsf{X}\ y) \mid x, y \in \mathrm{FV}(\Gamma), x \neq y\}$$

If $\Gamma = \{p_1, \ldots, p_n\}$ is finite then we put

$$\gamma(\{p_1, \ldots, p_n\}) = \iota(\Gamma) \cup \{u_1 : (\mathsf{T}\ \xi(p_1)), \ldots, u_n : (\mathsf{T}\ \xi(p_n))\}$$

Otherwise, if $\Gamma = \{p_n \mid n \in \mathbb{N}\}$ is infinite and representable by a term G in $\iota(\Gamma)$, we put $\gamma(\Gamma) = \iota(\Gamma) \cup \{U : \prod_{n:\mathsf{nat}}(\mathsf{T}\ (G\ n))\}$. Thus we have the following theorem, which is proved by induction.

Theorem 5 Adequacy of Σ(DL). *Let Γ be a representable (in $\iota(\Gamma)$) set of assumptions. Then*

1. $\forall \Gamma$, *if* $\gamma(\Gamma) \vdash M : A$, *where* $A \in \{\mathsf{X}, \mathsf{Te}, \mathsf{B}, \mathsf{C}, \mathsf{P}\}$, *then*

$$(\exists u.\gamma(\Gamma) \vdash_{\mathcal{S}_{\mathrm{ND}}(\mathrm{DL})} u : (\mathsf{isin}\ x\ A\ M)) \iff x \in \mathrm{FV}(M)$$
$$(\exists u.\gamma(\Gamma) \vdash_{\mathcal{S}_{\mathrm{ND}}(\mathrm{DL})} u : (\mathsf{isnotin}\ x\ A\ M)) \iff x \notin \mathrm{FV}(M)$$

2. $\forall \Gamma, p : \quad \Gamma \vdash_{\mathcal{S}_{\mathrm{ND}}(\mathrm{DL})} p \iff \exists d.\ \gamma(\Gamma) \vdash_{\Sigma(\mathrm{DL})} d : (\mathsf{T}\ p)$.

2.2 The Encoding of $\mathcal{S}^a_{\mathrm{ND}}$(DL): the Signature Σ^a(DL)

The new problematic issues is that of encoding proof rules. In fact, in the underlying theory there is no direct way of 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. The solution we give exploits again the possibility provided by the Logical Frameworks of considering locally quantified premises, i.e. general judgments in the terminology of Martin-Löf.

The basic judgment of Σ^a(DL) is $\mathsf{U} : \mathsf{P} \to \mathsf{W} \to \mathsf{Prop}$ where W is a set with *no* constructors. Elements of W will be called *worlds* for suggestive reasons. We can now define a new signature for $\mathcal{S}_{\mathrm{ND}}$(DL), namely Σ_w(DL), whose rules are obtained from the corresponding rules of Σ(DL) by just replacing T with U, and quantifying universally over the extra parameter; e.g.,

$$\supset\text{-I} : \prod_{p,q:\mathsf{P}} \prod_{w:\mathsf{W}} ((\mathsf{U}\ w\ p) \to (\mathsf{U}\ w\ q)) \to (\mathsf{U}\ w\ (p \supset q))$$

The CONVER rule can now be adequately encoded as follows:

$$\textsc{Conver} : \prod_{A:\mathsf{Te}\to\mathsf{P}} \prod_{c:\mathsf{C}} \prod_{t:\mathsf{Te}} \prod_{w:\mathsf{W}} \left(\prod_{w':\mathsf{W}} \prod_{x:\mathsf{X}} (\mathsf{U}\ w'\ (A\ t)) \to (\mathsf{isnotin}\ x\ \mathsf{P}\ \forall A) \to (\mathsf{isnotin}\ x\ \mathsf{C}\ c) \right.$$
$$\left. \to (\mathsf{U}\ w'\ (p\ (\mathsf{succ}\ (\mathsf{isId}\ x)))) \to (\mathsf{U}\ w'\ (\langle c \rangle\ (A\ (\mathsf{isId}\ x)))) \right) \to (\mathsf{U}\ w\ (\langle c^* \rangle\ (p\ 0)))$$

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 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 W, amounts to estabilishing the judgment for a generic world on which no assumptions are made, i.e. on no assumptions. This simple encoding of the proof rule $[\cdot]$-I illustrates the point:

$$[\cdot]\text{-I} : \prod_{p:\mathsf{P}} \prod_{c:\mathsf{C}} \left(\prod_{w:\mathsf{W}} (\mathsf{U}\ w\ p) \right) \to \prod_{w:\mathsf{W}} (\mathsf{U}\ w\ \ [c]\,p)$$

$$\text{Ass}\ \frac{}{\Gamma \vdash \{p[t/x]\}x := t\{p\}} \qquad \text{Cons}\ \frac{\Gamma_1, p \vdash p_1 \quad \Gamma_2 \vdash \{p_1\}c\{q_1\} \quad q_1 \vdash q}{\Gamma_1, \Gamma_2 \vdash \{p\}c\{q\}}$$

$$\text{If}\ \frac{\Gamma_1 \vdash \{p \wedge b\}c_1\{q\} \quad \Gamma_2 \vdash \{p \wedge \neg b\}c_2\{q\}}{\Gamma_1, \Gamma_2 \vdash \{p\}\textbf{if}\ b\ \textbf{then}\ c_1\ \textbf{else}\ c_2\{q\}} \qquad \text{While}\ \frac{\vdash \{p \wedge b\}c\{p\}}{\vdash \{p\}\textbf{while}\ b\ \textbf{do}\ c\{p \wedge \neg b\}}$$

$$\text{Or}\ \frac{\Gamma \vdash \{p\}c_1\{q\} \quad \Gamma \vdash \{p\}c_2\{q\}}{\Gamma \vdash \{p\}c_1 + c_2\{q\}} \qquad \text{Comp}\ \frac{\Gamma_1 \vdash \{p\}c_1\{r\} \quad \vdash \{r\}c_2\{q\}}{\Gamma_1 \vdash \{p\}c_1; c_2\{q\}}$$

$$\text{While\_Termin}\ \frac{\vdash p(n+1) \supset b \quad \Gamma \vdash [p(n+1)]c[p(n)] \quad \vdash p(0) \supset \neg b}{\Gamma \vdash [p(n)]\textbf{while}\ b\ \textbf{do}\ c[p(0)]}\ n \notin \mathrm{FV}(c)$$

Fig. 8. The rules of the system $\mathcal{S}_{\mathrm{ND}}(\mathrm{HL})$.

This idea, suitably generalized to take care of infinite sets of premises, can be used also to encode Scott's rule:

$$\text{Sc}: \prod_{G:\mathsf{nat}\to\mathsf{P}} \prod_{p:\mathsf{P}} \prod_{c:\mathsf{C}} \left(\prod_{w:\mathsf{W}} \left(\prod_{n:\mathsf{nat}} (\mathsf{U}\ w\ (G\ n))\right) \to (\mathsf{U}\ w\ p)\right) \to \prod_{w:\mathsf{W}} \left(\prod_{n:\mathsf{nat}} (\mathsf{U}\ w\ \ [c]\,(G\ n))\right) \to (\mathsf{U}\ w\ \ [c]\,p)$$

This is a general methodology which allows to encode adequately arbitrary proper sequent-like rules. For lack of space we do not discuss adequacy formally; see [20, 12] for more details.

3 Derivation of Truth Hoare Logic

In this section we outline the derivation in Coq of the rules of a ND-style system for representing the truth CR for Hoare Logic $\Sigma(\mathrm{DL})$. The truth CR for Hoare Logic can be obtained from Definition 1 by istantiating the appropriate parameters, which appear in App.A.

For lack of space we cannot elaborate on the different CR's for Hoare Logic and on the formal systems for representing them. The area of truth CR's and ND-style systems for Hoare Logic is almost unexplored (see [11, 20]). Even the system in [3] is sound only wrt the validity CR. There are various possibilities of defining ND-style systems for Hoare Logic utilizing the *non-interference* judgements of [24]. Interesting systems, which successfully scale up to languages with procedures, arise also if we take seriously reasoning under assumptions. Such are conceptually appealing in that they connect naturally to the language of DL. We expect them to be practically significant. Here we consider the system $\mathcal{S}_{\mathrm{ND}}(\mathrm{HL})$, appearing in Fig.8, which is sound and complete for the truth CR.

Proposition 6. *The partial correctness rules of* $\mathcal{S}_{\mathrm{ND}}(\mathrm{HL})$ *are derivable in* $\mathcal{S}_{\mathrm{ND}}(\mathrm{DL}) \cup \{\text{Sc}\}$*; the rule* While_Termin *is derivable in* $\mathcal{S}^{a}_{\mathrm{ND}}(\mathrm{DL}) \cup \{\text{Sc}\}$.

Proof. (Sketch) We examine only the case of While (Fig.8). Recall that $\textbf{while}\ b\ \textbf{do}\ c \stackrel{\text{def}}{=} (b?; c)^*; \neg b?$, and suppose that $\pi_h :\vdash p \wedge b \supset [c]\,p$. Then, for all

$n \in \mathbb{N}, p \vdash [b?;c]^n\, p$, where $\pi_0 = p \vdash p$ and π_{n+1} is defined inductively:[3]

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}(p)_2\\ \pi_n\\ {[b?;c]^n\,p}\end{array} \quad \dfrac{\dfrac{p\ (b)_1}{p\wedge b}\quad \begin{array}{c}\emptyset\\ \pi_h\\ p\wedge b\supset[c]\,p\end{array}}{[c]\,p}}{[c]\,[b?;c]^n\,p}(2);\dagger}{[b?]\,[c]\,[b?;c]^n\,p}(1)}{[b?;c]\,[b?;c]^n\,p}$$

where $\dagger$ is an application of Sc for $\Gamma = \{p\}$. Then, the following derivation is a proof of While in $\mathcal{S}_{\mathrm{ND}}(\mathrm{DL})$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(p)_2\ (\neg b)_3}{p\wedge\neg b}}{[\neg b?]\,(p\wedge\neg b)}(3)\quad \dfrac{\left\{\begin{array}{c|c}\begin{array}{c}(p)_1\\ \pi_n\\ {[b?;c]^n\,p}\end{array} & n\in\mathbb{N}\end{array}\right\}}{[(b?;c)^*]\,p}\dagger}{[(b?;c)^*]\,[\neg b?]\,(p\wedge\neg b)}(2);\ddagger}{[(b?;c)^*;\neg b?]\,(p\wedge\neg b)}}{p\supset[(b?;c)^*;\neg b?]\,(p\wedge\neg b)}(1)$$

where $\dagger, \ddagger$ are sound applications of *-I and Sc respectively. $\square$

The use of Sc is not essential, since this rule is admissible. However, the derivation of $\mathcal{S}_{\mathrm{ND}}(\mathrm{HL})$ is much easier if we assume Sc as an explicit rule of our system. In fact, due to the rules with discharged hypotheses, Coq does not allow for an inductive definition of the truth judgment U. Hence, we cannot reason inductively on proofs and derive in the system the admissibily of Sc.

The formal counterpart to Prop.6 has been carried out in Coq quite easily in the signature $\Sigma_w(\mathrm{DL}) \cup \{\mathrm{Sc}\}$.

4 Comparison with Related Work

To our knowledge, there is no published ND-style proof system for Dynamic Logic. Our approach was inspired by some unpublished notes by Colin Stirling [25], where a ND-style system for Deterministic Dynamic Logic is sketched. Stirling's fundamental idea is to "divorce the notion of free occurrence of a variable from that of substitution". The system deals with assertions of the form $p\theta$, where θ is called an *(explicit) substitution*: $\theta ::= \varepsilon \mid (\tfrac{t}{x}\theta)$. A prefix of the form $\tfrac{t_1}{x_1}\ldots\tfrac{t_n}{x_n}$ represents a sequence of "delayed" substitutions. Substitutions are not performed until the formula on which they are applied is command-free. This idea is inspiring but it is clearly impractical. $\mathcal{S}_{\mathrm{ND}}(\mathrm{DL})$ retains something of this idea, while it overcomes the "explicit substitution" problem in the assignment rules,

[3] The display of derivations is slightly non-standard but should be self explicatory.

by taking full advantage of assumptions, i.e. distributing the substitution in the proof context. Of course, this is sound only with respect to the truth consequence relation. The technique of treating substitutions by means of sets of assumptions has been introduced by Burstall and Honsell [4] and fully exploited in [19] in the context of encoding Natural Operational Semantics of programming languages in Type Theories.

A number of interesting issues arise if we compare the proof development environments generated by the signatures $\Sigma(\mathrm{DL})$ and $\Sigma_w(\mathrm{DL})$, to two remarkable examples of mechanized environments for program logics: the *Karlsruhe Interactive Verifier* (KIV) [10, 23] system and the implementation of Hoare Logic in the Cambridge HOL [6].

The KIV system is a tactical theorem prover based on (Deterministic) Dynamic Logic which realizes an environment for the development of verified software. In the tradition of the Edinburgh LCF, KIV provides a metalanguage which can be used for representing both the logic as well as the tactics and strategies for proof search and proof management. KIV is an Hilbert-style proof system: as in [7, 15], Dynamic Logic is axiomatised by means of several axioms and few rules. User-defined strategies and tactics make this unnatural calculus more user-friendly. The intended consequence relation of KIV is that of validity, not that of truth. As a consequence of this, KIV does not enjoy the Deduction Theorem ("$\Gamma, p_1 \vdash p_2 \iff \Gamma \vdash p_1 \supset p_2$" fails), which on the contrary is built in the system $\mathcal{S}_{\mathrm{ND}}(\mathrm{DL})$ which deals with "truth". Both KIV and the encodings of $\mathcal{S}_{\mathrm{ND}}(\mathrm{DL})$ represent the infinitary rule by means of a quantification over naturals, but while in the KIV system the quantification is at the level of the logic, in our approach it is at the meta-level (at the level of Coq). This makes our encoding closer in spirit to the original proof system. Furthermore, the higher order features of Coq provide "metavariables" for free: we can quantify over programs and carry out "schematic" proofs which can be reused.

The Hilbert-style proof system $\mathcal{S}_{\mathrm{H}}(\mathrm{HL})$ for Hoare Logic, implemented in the Cambridge HOL, among other aspects features a very interesting treatment of program variables: they are represented by objects of type *string*. This encoding provides naturally an infinite set of variables, different from one other, without the need of supplementary assumptions. This technique could be used to simplify our treatment of identifiers. However we still need the judgments isin, $\mathsf{isnotin}$, to deal with, e.g., variables quantified locally to assumptions.

5 Final Remarks and Directions for Future Work

Pragmatics. Although the systems presented in this paper are quite powerful and rather natural, the proof development environments Coq-generated by their encodings are probably not yet effectively usable on large case studies. A serious pragmatic problem is that we have to duplicate at the level of the object logic (i.e. P), a lot of the machinery already present in Coq, and hence we cannot take full advantage of built-in tactics and strategies. However, it is still open whether it is possible to extend the formula-as-types paradigm to boxed formulæ of Dynamic Logic, or to explain them away using HOL constructs.

A possible pragmatic improvement of our approach would be that of automatizing derivations connected to side-condition judgments such as isin, $\mathsf{isnotin}$, BF which are deterministically syntax-driven. This could be done using a logic programming language like Elf [21], or defining suitable tactics.

Systems of Dynamic Logic over other data types, beyond PA, should be investigated.

Finitary vs. Infinitary systems. Our systems are essentially infinitary, since we are interested in strongly complete representations of $\models$. It would be interesting to investigate the power of *finitary* proof systems. For instance, we could replace the *-I rule by the finitary *invariance rule*:

$$^*_f\text{-I}\ \frac{\Gamma \vdash p \quad p \vdash [c]\,p}{\Gamma \vdash [c^*]\,p}$$

The system $\mathcal{S}_{\mathrm{ND}}{}^f(\mathrm{DL}) \stackrel{\mathrm{def}}{=} \mathcal{S}_{\mathrm{ND}}(\mathrm{DL}) \setminus \{^*\text{-I}\} \cup \{^*_f\text{-I}\}$ is incomplete, since it does not allow to derive the fundamental axiom of iteration ([15, Theor.3(7)]), i.e. $\not\vdash_{\mathcal{S}_{\mathrm{ND}}{}^f(\mathrm{DL})} [c^*]\,p \supset [c]\,[c^*]\,p$. However, $\mathcal{S}_{\mathrm{ND}}{}^f(\mathrm{DL}) \cup \{[c^*]\,p \supset [c]\,[c^*]\,p\}$ is strongly complete with respect to $\models \cap (\mathcal{P}_{<\omega}(\mathbb{P}) \times \mathbb{P})$.

Equivalences of Programs. An interesting application of the proof editor generated from the signature $\Sigma(\mathrm{DL})$ using Coq is the possibility of proving formally the equivalences of programs. Following Meyer and Halpern [17], two programs $c, d \in \mathbb{C}$ are *equivalent* ($[\![c]\!] = [\![d]\!]$) if $\forall \mathcal{M} : [\![c]\!]_{\mathcal{M}} = [\![d]\!]_{\mathcal{M}}$. In other words, there is no model in which we can distinguish between the two programs. The encodings of $\mathcal{S}_{\mathrm{ND}}(\mathrm{DL})$ could be particularly suited for *computer-assisted* proofs of equivalence of programs, since they naturally provide metalogical facilities such as quantifications on predicates (i.e. second-order quantifications) and proofs by induction on the structure of predicates.

Arithmetical Completeness. Completeness of Dynamic and Hoare Logics is usually discussed in terms of *arithmetical* (or *expressive*) models and *arithmetical* (*Cook's*) completeness [1, 8, 15]. A Hilbert-style system $\mathcal{S}_{\mathrm{H}}$ is Cook complete w.r.t. a class A of arithmetical models if $\forall \mathcal{M} \in A, \forall p : (\models_{\mathcal{M}} p \Rightarrow \mathrm{Th}(\mathcal{M}) \vdash_{\mathcal{S}_{\mathrm{H}}} p)$, where $\mathrm{Th}(\mathcal{M})$ denotes the collection of all command-free formulæ valid in the model $\mathcal{M}$. This means that completeness w.r.t. a particular model $\mathcal{M}$ is achieved by adding to the system *all* the first order properties of *that* model. This is different from our completeness results (Th.2, 3), where no extra axioms are needed. Indeed, the whole first order theory of $\mathbb{N}$ can be derived by $\mathcal{S}^a_{\mathrm{ND}}(\mathrm{DL})$, due to the power of the infinitary rule *-I. On the other hand, Theorem 3 holds only for the special case of the standard model of arithmetic. If we want to give a natural deduction formulation of systems such as those of [1, 8, 15], we need to introduce an auxiliary unary predicate symbol, *isnat*, whose intended meaning is the set of standard integers (see [8, p.29]). In this case, the CONVER rule has to be modified as follows:

$$\textsc{Conver}\ \frac{\emptyset \vdash (\mathit{isnat}(x) \wedge p^{x+1}_x) \supset \langle c\rangle\, p \quad \Gamma \vdash \mathit{isnat}(t) \supset p^t_x}{\Gamma \vdash \langle c^*\rangle\, p^0_x}\ x \notin \mathrm{FV}(c)$$

A Syntax and Semantics of DL

Syntax and Semantics of PA. The language $\mathcal{L}(\mathrm{PA})$ of Peano Arithmetic is defined as follows:

Identifiers	$\mathbb{X} : x ::= i_0 \mid i_1 \mid i_2 \mid i_3 \mid \dots$
Terms	$\mathbb{T} :\ t ::= 0 \mid 1 \mid x \mid t+t \mid t*t$
Propositional Formulæ	$\mathbb{B} :\ b ::= t=t \mid t<t \mid b \supset b \mid b \wedge b \mid \neg b$
Formulæ	$\mathbb{P} : p ::= t=t \mid t<t \mid p \supset p \mid p \wedge p \mid \neg p \mid \forall x p$

The interpretation functions $\mathcal{T}[\![\cdot]\!]_{\mathcal{M}} : \mathbb{T} \to \mathbb{S}_{\mathcal{M}} \to \mathbb{D}_{\mathcal{M}}$, $\mathcal{F}[\![\cdot]\!]_{\mathcal{M}} : \mathbb{P} \to \mathcal{P}(\mathbb{S}_{\mathcal{M}})$ are defined in the style of Denotational Semantics over a model $\mathcal{M} = \langle D_{\mathcal{M}}, 0, 1, +, \cdot, \dots \rangle$ for Peano Arithmetic. $\mathbb{S}_{\mathcal{M}} = \mathbb{X} \to D_{\mathcal{M}}$ is the domain of *environments* and it is ranged over by s, s_1, s_2. These two semantic functions are defined on the syntax of phrases in the obvious way. The semantics of formulæ is naturally extended to sets of formulæ: i.e. if $\Gamma \subseteq \mathbb{F}$ then $\mathcal{F}[\![\Gamma]\!]_{\mathcal{M}} \stackrel{\text{def}}{=} \bigcap_{p \in \Gamma} \mathcal{F}[\![p]\!]_{\mathcal{M}}$. Then, the usual Tarski's interpretation relation $\models_{\mathcal{M}} \subset \mathbb{S}_{\mathcal{M}} \times \mathbb{F}$ amounts to membership, i.e. $s \models_{\mathcal{M}} p \iff s \in \mathcal{F}[\![p]\!]_{\mathcal{M}}$.

Syntax and Semantics of Hoare Logic. Since in this paper we focus on PA, we give the definition of HL and DL only with respect to the first order theory of PA. The language $\mathcal{L}(\mathrm{HL})$ is defined by restricting $\mathcal{L}(\mathrm{PA})$ to quantifier-free formulæ and by introducing the new syntactic domains of *nondeterministic* **while** *programs*, *Hoare triples* and *assertions* as follows:

Non-deterministic While Programs	$\mathbb{W} : c ::= x := t \mid c; c \mid c + c$ $\mid$ **if** b **then** c **else** c $\mid$ **while** b **do** c	Hoare Triples Assertions	$\mathbb{H} : h ::= \{p\}C\{q\}$ $\mathbb{A} : a ::= p \mid h$

The semantics of Hoare Logic is given by naturally extending the interpretation to the new syntactic domains, i.e. $\mathcal{W}[\![\cdot]\!]_{\mathcal{M}} : \mathbb{W} \to \mathbb{S}_{\mathcal{M}} \to \mathcal{P}(\mathbb{S}_{\mathcal{M}})$, $\mathcal{H}[\![\cdot]\!]_{\mathcal{M}} : \mathbb{H} \to \mathcal{P}(\mathbb{S}_{\mathcal{M}})$, $\mathcal{A}[\![\cdot]\!]_{\mathcal{M}} : \mathbb{A} \to \mathcal{P}(\mathbb{S}_{\mathcal{M}})$. Hoare triples are interpreted as usual: $\mathcal{H}[\![\{p\}c\{q\}]\!]_{\mathcal{M}} \stackrel{\text{def}}{=} \{s \in \mathbb{S}_{\mathcal{M}} \mid s \in \mathcal{F}[\![p]\!]_{\mathcal{M}} \Rightarrow \mathcal{C}[\![c]\!]_{\mathcal{M}} s \subseteq \mathcal{F}[\![q]\!]_{\mathcal{M}}\}$.

Syntax and Semantics of DL. The language $\mathcal{L}(\mathrm{DL})$ is defined by extending $\mathcal{L}(\mathrm{PA})$ with a new formula constructor, $[\cdot]\cdot$, and by introducing the new syntactic domains, of *command-free formulæ* and of *regular programs*, as follows:

Command-free Formulæ	$\mathbb{F} : p ::= t=t \mid t<t \mid p \supset p \mid p \wedge p \mid \neg p \mid \forall x p$
Regular Programs	$\mathbb{C} : c ::= x := t \mid b? \mid c; c \mid c + c \mid c^*$
Formulæ	$\mathbb{P} : p ::= t=t \mid t<t \mid p \supset p \mid p \wedge p \mid \neg p \mid \forall x p \mid [c]\, p$

The semantics of DL is given by extending the interpretation to the domain $\mathbb{C}$. The function $\mathcal{C}[\![\cdot]\!]_{\mathcal{M}} : \mathbb{C} \to \mathbb{S}_{\mathcal{M}} \to \mathcal{P}(\mathbb{S}_{\mathcal{M}})$ is defined as follows (the composition operator is extended in the obvious way):

$$\mathcal{C}[\![x := t]\!]_{\mathcal{M}} \stackrel{\text{def}}{=} \lambda s.\{s[x \mapsto \mathcal{T}[\![t]\!]_{\mathcal{M}} s]\}$$

$$\mathcal{C}[\![c_1; c_2]\!] \stackrel{\text{def}}{=} \mathcal{C}[\![c_2]\!] \circ \mathcal{C}[\![c_1]\!] \qquad \mathcal{C}[\![c^*]\!] \stackrel{\text{def}}{=} \lambda s. \cup_{n \in \omega} \mathcal{C}[\![c]\!]^n s$$

$$\mathcal{C}[\![b?]\!] \stackrel{\text{def}}{=} \lambda s. \mathcal{F}[\![b]\!] \cap \{s\} \qquad \mathcal{C}[\![c_1 + c_2]\!] \stackrel{\text{def}}{=} \mathcal{C}[\![c_1]\!] \cup \mathcal{C}[\![c_2]\!]$$

Finally, the semantics of formulæ $\mathcal{F}[\![\cdot]\!]_{\mathcal{M}} : \mathbb{P} \to \mathcal{P}(\mathbb{S}_{\mathcal{M}})$ is extended in the extra case, by putting $\mathcal{F}[\![[c]\, p]\!]_{\mathcal{M}} \stackrel{\text{def}}{=} \{s \in \mathbb{S}_{\mathcal{M}} \mid \mathcal{C}[\![c]\!] s \subseteq \mathcal{F}[\![p]\!]\}$.

B Consequence Relations

Definition 7 CR. A *(single-conclusioned) Consequence Relation* on a set $\mathbb{F}$ of formulæ is a binary relation $\models \subseteq \mathcal{P}(\mathbb{F}) \times \mathbb{F}$ which satisfies the following properties:
Reflexivity: $p \models p$ for every formula $p \in \mathbb{F}$;
Transitivity, or "Cut": if $\Gamma_1 \models p$ and $\Gamma_2, p \models q$ then $\Gamma_1, \Gamma_2 \models p$.
Γ is called the *antecedent* or *set of assumptions*, and p is the *conclusion*. □

This definition differs from the one of [2] only in that we allow for possibly infinite sets of assumptions and exactly one conclusion.

CR's are usually defined in a completely abstract way, e.g. using semantics. Therefore, definitions of CR's are usually *ineffective*, and cannot be used in practice in order to establish consequences of formulæ from sets of assumptions. In order to use a CR one needs a *concrete* way of representing it. This is achieved by defining a *formal proof system* (called "calculus"). The objects of a formal proof systems usually are not simply formulæ of the logic, but can be formal representations of consequentiality (i.e. *sequents*, or even proofs of formulæ).

Definition 8 FPS. A *Formal Proof System* $\mathcal{S}$, or *Calculus*, for a CR $\models$ on a set $\mathbb{F}$ of formulæ is a method for defining a CR on $\mathbb{F}$, denoted by $\vdash_{\mathcal{S}}$.

1. $\vdash_{\mathcal{S}}$ is *sound* (*faithful*) if $\vdash_{\mathcal{S}} \subseteq \models$, that is, $\forall \Gamma, p : \Gamma \vdash_{\mathcal{S}} p \to \Gamma \models p$;
2. $\vdash_{\mathcal{S}}$ is *complete* if $\forall p : \emptyset \models p \to \emptyset \vdash_{\mathcal{S}} p$;
3. $\vdash_{\mathcal{S}}$ is *strongly complete* if $\models \subseteq \vdash_{\mathcal{S}}$, that is, $\forall \Gamma, p : \Gamma \models p \to \Gamma \vdash_{\mathcal{S}} p$. □

The assertion "$\Gamma \vdash_{\mathcal{S}} A$" is called *a (formal) sequent* and is read "A is derivable from Γ (in the system $\mathcal{S}$)."

Following [2], rules in ND-style calculi are general schemata of the form

$$\forall \Gamma_1, \ldots, \Gamma_n \frac{\Gamma_1, \Delta_1 \vdash p_1 \ldots \Gamma_n, \Delta_n \vdash p_n}{\Gamma_1, \ldots, \Gamma_n \vdash p} C$$

where C is a possible *side condition*, that is a restriction (max. level 2, in the terminology of [2]) on the applicability of the schemata.

References

1. K. R. Apt. Ten years of Hoare's logic: A survey — part I. *ACM Transactions on Programming Languages and Syms*, 3(4):431–483, Oct. 1981.
2. A. Avron. Simple consequence relations. *Inform. Comput.*, 92:105–139, Jan. 1991.
3. 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.
4. R. Burstall and F. Honsell. Operational semantics in a natural deduction setting. In Huet and Plotkin [13], pages 185–214.
5. T. Coquand and G. Huet. The calculus of constructions. *Information and Control*, 76:95–120, 1988.
6. M. J. C. Gordon. Mechanizing program logics in higher order logic. In P. A. Subrahmanyam and G. Birtwistle, editors, *Current Trends in Hardware Verification and Automated Theorem Prover*, pages 387–439. Springer-Verlag, 1989.

7. D. Harel. *First-Order Dynamic Logic.* No.68 in LNCS. Springer-Verlag, 1979.
8. D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II, pages 497–604. Reidel, 1984.
9. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, Jan. 1993.
10. M. Heisel, W. Reif, and W. Stephan. A dynamic logic for program verification. In A. Meyer and M. Taitslin, editors, *Proc. of LFCS (Logic at Botik)*, number 363 in Lecture Notes in Computer Science, pages 134–145. Springer-Verlag, 1989.
11. F. Honsell and M. Miculan. Encoding program logics in type theories. In J. Despeyroux, editor, *Deliverables of the TYPES Workshop* Proving Properties of Programming Languages, Sophia-Antipolis, Sept. 1993.
12. F. Honsell, M. Miculan, and C. Paravano. Encoding modal logics in Logical Frameworks. To appear, 1996.
13. G. Huet and G. Plotkin, editors. *Logical Frameworks.* CUP, June 1990.
14. INRIA, Rocquencourt. *The Coq Proof Assistant Reference Manual*, July 1995.
15. D. Kozen and J. Tiuryn. Logics of Programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 789–840. North Holland, 1990.
16. Z. Luo, R. Pollack, and P. Taylor. *How to use LEGO (A Preliminary User's Manual).* Department of Computer Science, University of Edinburgh, Oct. 1989.
17. A. R. Meyer and J. Y. Halpern. Axiomatic definition of programming languages: A theoretical assessment. *J. ACM*, 29(2):555–576, Apr. 1982.
18. S. Michaylov and F. Pfenning. Natural Semantics and some of its Meta-Theory in Elf. In L.-H. Eriksson, L. Hallnäs, and P. Schroeder-Heister, editors, *Proceedings of the Second International Workshop on Extensions of Logic Programming*, number 596 in LNAI, pages 299–344, Stockolm, Sweden, Jan. 1991. Springer-Verlag.
19. M. Miculan. The expressive power of structural operational semantics with explicit assumptions. In H. Barendregt and T. Nipkow, editors, *Proceedings of TYPES'93*, number 806 in LNCS, pages 292–320. Springer-Verlag, 1994.
20. M. Miculan. *Encoding Logical Theories of Programs.* PhD thesis, Università di Pisa, 1997. To appear.
21. F. Pfenning. Elf: A language for logic definition and verified metaprogramming. In *Fourth Annual Symposium on Logic in Computer Science*, pages 313–322. IEEE, June 1989.
22. D. Prawitz. *Natural Deduction.* Almqvist & Wiksell, Stockholm, 1965.
23. W. Reif. The KIV system: Systematic construction of verified software. In D. Kapur, editor, *Proc. of CADE-11*, number 607 in Lecture Notes in Computer Science, pages 753–757. Springer-Verlag, 1992.
24. J. C. Reynolds. Syntactic control of interference. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 39–46, Tucson, Oct. 1978. The Association for Computing Machinery.
25. C. Stirling. Logics for While Programs: Algorithmic/Dynamic Logics. Unpublished notes, 1985.
26. C. Stirling. Modal and Temporal Logics. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 477–563. Oxford University Press, 1992.
27. B. Werner. *Une théorie des constructions inductives.* PhD thesis, Université Paris 7, 1994.