Nominalization in Intensional Type Theory

Jørgen Villadsen
Technical University of Denmark

LICS 2006

Note added 2007:

Nominalistic Logic (NL) is based on "An Intensional Type Theory: Motivation and Cut-Elimination" by Paul Gilmore (Journal of Symbolic Logic 383-400 2001).

The following short presentation is from the Annual IEEE Symposium on Logic in Computer Science (Seattle USA 2006) and explains the idea of nominalization (but the name NL is from 2007).

Introduction

Oxford English Dictionary:

By nominalization we understand the process of converting a word or phrase into a noun

Nominalization is important for natural language processing "The apple is red and red is a colour"

In a logic with types for individuals and predicates it involves using predicates as individuals

Problem:

Nominalization is impossible in Church's Simple Type Theory (Journal of Symbolic Logic 1940)

Also need Infinity Axiom to get a foundation of mathematics!

Idea:

1. Start with Gilmore's Intensional Type Theory (Journal of Symbolic Logic 2001) ASL book 2005
2. Recast so-called Intensional Rules as Nominalization Rules

No need for Infinity Axiom to get a foundation of mathematics!

Intensionality possible but not necessary...

Types and Terms

Let $\overline{\tau}$ be a sequence of types $\tau_1 \cdots \tau_n$ ($n \geq 0$, hence the sequence can be empty)

We have types for individuals and predicates:

$$\tau ::= \iota \mid [\overline{\tau}]$$

In particular we have $[]$, or o, for formulas

We have terms with application, abstraction, constants and variables:

$$t ::= ps \mid \lambda x.\, q \mid c \mid v$$

We require that if $p : [\tau\overline{\tau}]$ and $s : \tau$ then $t : [\overline{\tau}]$ and if $x : \tau$ and $q : [\overline{\tau}]$ then $t : [\tau\overline{\tau}]$

Special nominalization feature:

We allow a term of type $[\overline{\tau}]$ with free occurrences of type ι variables only to have type ι too

Constants and Abbreviations

We assume special constants for negation $: [o], conjunction & : [oo], and existence $\odot$: [[τ]]

We use the following abbreviations with suitable priorities:

$$ps_1 \cdots s_n \rightsquigarrow (ps_1) \cdots s_n \qquad \lambda x_1 \cdots x_n.\, q \rightsquigarrow \lambda x_1.\, \cdots \lambda x_n.\, q$$

$$\neg p \rightsquigarrow \$p \qquad \neg\neg p \rightsquigarrow \neg(\neg p)$$

$$p \wedge q \rightsquigarrow \&pq \qquad p \wedge q_1 \wedge \cdots \wedge q_n \rightsquigarrow (p \wedge q_1) \wedge \cdots \wedge q_n$$

$$p \vee q \rightsquigarrow \neg(\neg p \wedge \neg q) \qquad p \vee q_1 \vee \cdots \vee q_n \rightsquigarrow (p \vee q_1) \vee \cdots \vee q_n$$

$$p \to q \rightsquigarrow \neg(p \wedge \neg q) \qquad p_1 \to \cdots \to p_n \to q \rightsquigarrow p_1 \to \cdots \to (p_n \to q)$$

$$p \leftrightarrow q \rightsquigarrow (p \to q) \wedge (q \to p) \qquad p_1 \leftrightarrow \cdots \leftrightarrow p_n \leftrightarrow q \rightsquigarrow p_1 \leftrightarrow \cdots \leftrightarrow (p_n \leftrightarrow q)$$

$$\exists x.\, p \rightsquigarrow \odot \lambda x.\, p \qquad \exists x_1 \cdots x_n.\, p \rightsquigarrow \exists x_1.\, \cdots \exists x_n.\, p$$

$$\forall x.\, p \rightsquigarrow \neg\exists x.\, \neg p \qquad \forall x_1 \cdots x_n.\, p \rightsquigarrow \forall x_1.\, \cdots \forall x_n.\, p$$

$$s \neq t \rightsquigarrow \exists x.\, xs \wedge \neg xt \qquad s = t \rightsquigarrow \neg(s \neq t)$$

$$\bot \rightsquigarrow \$ \neq \$ \qquad \top \rightsquigarrow \neg\bot$$

We implicit use abstractions in order to avoid binding free variables

Lambdas and Sequents

As usual renaming of bound variables is allowed (α)

We define for a term p the contraction term p^λ (substitution must not bind free variables):

(β)	An application	$(\lambda x.\, s)\, t$	contracts to	$s[x := t]$	
(η)	An abstraction	$\lambda x.\, sx$	contracts to	s	(x must not be free in s)

The contraction can appear anywhere in the term

We view a sequent $\Gamma \vdash \Delta$ as two sequences of alternatives Γ, Δ where for any interpretation (at least) one of the formulas in Δ must be true or one of the formulas in Γ must not be true (hence false)

The order and possible repetitions of formulas in the sequences does not matter

In the following rules types are occasionally mentioned in terms in order to indicate restrictions

Rules

Structural Rules (*contraction*, *cut*, *left thinning*, *right thinning*):

$$p \vdash p^{\lambda} \qquad \frac{p, \Gamma \vdash \Delta \qquad \Gamma \vdash \Delta, p}{\Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta}{p, \Gamma \vdash \Delta} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, p}$$

Propositional Rules (*left negation*, *right negation*, *left conjunction*, *right conjunction*):

$$\neg p, p \vdash \qquad \vdash p, \neg p \qquad \frac{p, q, \Gamma \vdash \Delta}{p \wedge q, \Gamma \vdash \Delta} \qquad p, q \vdash p \wedge q$$

Quantificational Rules (*left existence*, *right existence*):

$$\frac{pc, \Gamma \vdash \Delta}{\odot p, \Gamma \vdash \Delta}\ \text{Constant } c \text{ must not occur in } p, \Gamma \text{ or } \Delta \qquad ps \vdash \odot p$$

Rules — Cont.

Nominalization Rules (*equals nominalization*, *non-equals nominalization*):

$$p =_{[\overline{\tau}]} q \vdash p =_\iota q \qquad p \neq_{[\overline{\tau}]} q \vdash p \neq_\iota q$$

The main novelty is in the last rules, denoted (=) and (≠)

The first states that the nominalizations of equal predicates are equal

The second similarly — as a kind of Infinity Axiom of course — for non-equal predicates

We can obtain the natural numbers as the following abbreviations:

$$\begin{aligned} \mathsf{0} &\rightsquigarrow \lambda x.\, x \neq x \\ \mathsf{s}n &\rightsquigarrow \lambda x.\, n = x \\ \mathsf{1} &\rightsquigarrow \mathsf{s0} \quad \mathsf{2} \rightsquigarrow \mathsf{s1} \quad \ldots \end{aligned}$$

We obtain an infinite sequence of pairwise non-equal terms: $\mathsf{0}, \mathsf{1}, \mathsf{2}, \ldots$

The non-equality comes essentially from the built-in nominalization and the rule ≠

Related and Future Work

We have added nominalization to a Finite Type Theory

In Transfinite Type Theory, for example the formulation with type variables by Andrews, all finite types are included in the first transfinite type, but no nominalization is possible for the transfinite types as such

A paraconsistent logic is possible by unifying the basic types ι and o such that $\tau ::= [\overline{\tau}]$

Classical logic retained as a special case

Cf. my recent papers (Journal of Applied Non-Classical Logics 2005)

References

[1] P. B. Andrews. *A Transfinite Type Theory with Type Variables*. North-Holland, 1965.

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

[3] P. C. Gilmore. An intensional type theory: Motivation and cut-elimination. *Journal of Symbolic Logic*, 66(1):383–400, 2001.

[4] P. C. Gilmore. *Logicism Renewed: Logical Foundations for Mathematics and Computer Science*. Association for Symbolic Logic, 2005.

[5] J. Villadsen. A paraconsistent higher order logic. In B. Buchberger and J. A. Campbell, editors, *Artificial Intelligence and Symbolic Computation*, pages 38–51. Lecture Notes in Computer Science 3249, Springer-Verlag, 2004. A preliminary version appeared in the informal proceedings of the workshop on Paraconsistent Computational Logic 2002.

[6] J. Villadsen. Supra-logic: Using transfinite type theory with type variables for paraconsistency. *Journal of Applied Non-Classical Logics*, 15(1):45–58, 2005.