Institute for Logic, Language and Computation

THE VANILLA META-INTERPRETER FOR DEFINITE LOGIC PROGRAMS AND AMBIVALENT SYNTAX

Marianne Kalsbeek

ILLC Prepublication Series
for Computation and Complexity Theory CT-93-01

University of Amsterdam

The ILLC Prepublication Series

1990

Logic, Semantics and Philosophy of Language

LP-90-01	Jaap van der Does	A Generalized Quantifier Logic for Naked Infinitives
LP-90-02	Jeroen Groenendijk, Martin Stokhof	Dynamic Montague Grammar
LP-90-03	Renate Bartsch	Concept Formation and Concept Composition
LP-90-04	Aarne Ranta	Intuitionistic Categorial Grammar
LP-90-05	Patrick Blackburn	Nominal Tense Logic
LP-90-06	Gennaro Chierchia	The Variablity of Impersonal Subjects
LP-90-07	Gennaro Chierchia	Anaphora and Dynamic Logic
LP-90-08	Herman Hendriks	Flexible Montague Grammar
LP-90-09	Paul Dekker	The Scope of Negation in Discourse, towards a Flexible Dynamic Montague grammar
LP-90-10	Theo M.V. Janssen	Models for Discourse Markers
LP-90-11	Johan van Benthem	General Dynamics
LP-90-12	Serge Lapierre	A Functional Partial Semantics for Intensional Logic
LP-90-13	Zhisheng Huang	Logics for Belief Dependence
LP-90-14	Jeroen Groenendijk, Martin Stokhof	Two Theories of Dynamic Semantics
LP-90-15	Maarten de Rijke	The Modal Logic of Inequality
LP-90-16	Zhisheng Huang, Karen Kwast	Awareness, Negation and Logical Omniscience
LP-90-17	Paul Dekker	Existential Disclosure, Implicit Arguments in Dynamic Semantics

Mathematical Logic and Foundations

ML-90-01	Harold Schellinx	Isomorphisms and Non-Isomorphisms of Graph Models
ML-90-02	Jaap van Oosten	A Semantical Proof of De Jongh's Theorem
ML-90-03	Yde Venema	Relational Games
ML-90-04	Maarten de Rijke	Unary Interpretability Logic
ML-90-05	Domenico Zambella	Sequences with Simple Initial Segments
ML-90-06	Jaap van Oosten	Extension of Lifschitz' Realizability to Higher Order Arithmetic, and a Solution to a Problem of F. Richman
ML-90-07	Maarten de Rijke	A Note on the Interpretability Logic of Finitely Axiomatized Theories
ML-90-08	Harold Schellinx	Some Syntactical Observations on Linear Logic
ML-90-09	Dick de Jongh, Duccio Pianigiani	Solution of a Problem of David Guaspari
ML-90-10	Michiel van Lambalgen	Randomness in Set Theory
ML-90-11	Paul C. Gilmore	The Consistency of an Extended NaDSet

Computation and Complexity Theory

CT-90-01	John Tromp, Peter van Emde Boas	Associative Storage Modification Machines
CT-90-02	Sieger van Denneheuvel, Gerard R. Renardel de Lavalette	A Normal Form for PCSJ Expressions
CT-90-03	Ricard Gavaldà, Leen Torenvliet, Osamu Watanabe, José L. Balcázar	Generalized Kolmogorov Complexity in Relativized Separations
CT-90-04	Harry Buhrman, Edith Spaan, Leen Torenvliet	Bounded Reductions
CT-90-05	Sieger van Denneheuvel, Karen Kwast	Efficient Normalization of Database and Constraint Expressions
CT-90-06	Michiel Smid, Peter van Emde Boas	Dynamic Data Structures on Multiple Storage Media, a Tutorial
CT-90-07	Kees Doets	Greatest Fixed Points of Logic Programs
CT-90-08	Fred de Geus, Ernest Rotterdam, Sieger van Denneheuvel, Peter van Emde Boas	Physiological Modelling using RL
CT-90-09	Roel de Vrijer	Unique Normal Forms for Combinatory Logic with Parallel Conditional, a case study in conditional rewriting

Other Prepublications

X-90-01	A.S. Troelstra	Remarks on Intuitionism and the Philosophy of Mathematics, Revised Version
X-90-02	Maarten de Rijke	Some Chapters on Interpretability Logic
X-90-03	L.D. Beklemishev	On the Complexity of Arithmetical Interpretations of Modal Formulae
X-90-04		Annual Report 1989
X-90-05	Valentin Shehtman	Derived Sets in Euclidean Spaces and Modal Logic
X-90-06	Valentin Goranko, Solomon Passy	Using the Universal Modality: Gains and Questions
X-90-07	V.Yu. Shavrukov	The Lindenbaum Fixed Point Algebra is Undecidable
X-90-08	L.D. Beklemishev	Provability Logics for Natural Turing Progressions of Arithmetical Theories
X-90-09	V.Yu. Shavrukov	On Rosser's Provability Predicate
X-90-10	Sieger van Denneheuvel, Peter van Emde Boas	An Overview of the Rule Language RL/1
X-90-11	Alessandra Carbone	Provable Fixed points in $I\Delta_0+\Omega_1$, revised version
X-90-12	Maarten de Rijke	Bi-Unary Interpretability Logic
X-90-13	K.N. Ignatiev	Dzhaparidze's Polymodal Logic: Arithmetical Completeness, Fixed Point Property, Craig's Property
X-90-14	L.A. Chagrova	Undecidable Problems in Correspondence Theory
X-90-15	A.S. Troelstra	Lectures on Linear Logic

1991

Logic, Semantics and Philosophy of Langauge

LP-91-01	Wiebe van der Hoek, Maarten de Rijke	Generalized Quantifiers and Modal Logic
LP-91-02	Frank Veltman	Defaults in Update Semantics
LP-91-03	Willem Groeneveld	Dynamic Semantics and Circular Propositions
LP-91-04	Makoto Kanazawa	The Lambek Calculus enriched with Additional Connectives
LP-91-05	Zhisheng Huang, Peter van Emde Boas	The Schoenmakers Paradox: Its Solution in a Belief Dependence Framework
LP-91-06	Zhisheng Huang, Peter van Emde Boas	Belief Dependence, Revision and Persistence
LP-91-07	Henk Verkuyl, Jaap van der Does	The Semantics of Plural Noun Phrases
LP-91-08	Víctor Sánchez Valencia	Categorial Grammar and Natural Reasoning
LP-91-09	Arthur Nieuwendijk	Semantics and Comparative Logic
LP-91-10	Johan van Benthem	Logic and the Flow of Information

Mathematical Logic and Foundations

ML-91-01	Yde Venema	Cylindric Modal Logic
ML-91-02	Alessandro Berarducci, Rineke Verbrugge	On the Metamathematics of Weak Theories
ML-91-03	Domenico Zambella	On the Proofs of Arithmetical Completeness for Interpretability Logic
ML-91-04	Raymond Hoofman, Harold Schellinx	Collapsing Graph Models by Preorders
ML-91-05	A.S. Troelstra	History of Constructivism in the Twentieth Century
ML-91-06	Inge Bethke	Finite Type Structures within Combinatory Algebras
ML-91-07	Yde Venema	Modal Derivation Rules
ML-91-08	Inge Bethke	Going Stable in Graph Models
ML-91-09	V.Yu. Shavrukov	A Note on the Diagonalizable Algebras of PA and ZF
ML-91-10	Maarten de Rijke, Yde Venema	Sahlqvist's Theorem for Boolean Algebras with Operators
ML-91-11	Rineke Verbrugge	Feasible Interpretability
ML-91-12	Johan van Benthem	Modal Frame Classes, revisited

Computation and Complexity Theory

CT-91-01	Ming Li, Paul M.B. Vitányi	Kolmogorov Complexity Arguments in Combinatorics
CT-91-02	Ming Li, John Tromp, Paul M.B. Vitányi	How to Share Concurrent Wait-Free Variables
CT-91-03	Ming Li, Paul M.B. Vitányi	Average Case Complexity under the Universal Distribution Equals Worst Case Complexity
CT-91-04	Sieger van Denneheuvel, Karen Kwast	Weak Equivalence
CT-91-05	Sieger van Denneheuvel, Karen Kwast	Weak Equivalence for Constraint Sets
CT-91-06	Edith Spaan	Census Techniques on Relativized Space Classes
CT-91-07	Karen L. Kwast	The Incomplete Database
CT-91-08	Kees Doets	Levationis Laus

Institute for Logic, Language and Computation
Plantage Muidergracht 24
1018TV Amsterdam
Telephone 020-525.6051, Fax: 020-525.5101

THE VANILLA META-INTERPRETER FOR DEFINITE LOGIC PROGRAMS AND AMBIVALENT SYNTAX

Marianne Kalsbeek
Department of Mathematics and Computer Science
University of Amsterdam

ILLC Prepublications
for Computation and Complexity Theory
ISSN 0928-3323

Coordinating editor: Dick de Jongh

Research supported by
the Netherlands Organization for Scientific Research (NWO)

received January 1993

The Vanilla Meta-Interpreter for Definite Logic Programs and Ambivalent Syntax

Marianne Kalsbeek[*]

Department of Mathematics and Computer Science
University of Amsterdam
Plantage Muidergracht 24, 1018 TV Amsterdam
e-mail: marianne@fwi.uva.nl

Abstract
This paper presents a simple syntax and semantics for the Vanilla meta-interpreter. We discuss ambivalent syntax, which is characterised by the occurrence of formulas as terms, and provide it with a suitable semantics. We show that ambivalent syntax is appropriate for the Vanilla meta-interpreter. We prove correctness of the Vanilla meta-interpreter for definite programs.

1 Introduction: Correctness of Meta-Logic Programming

In this paper we will study the simplest meta-interpreter for definite logic programs, usually known as the Vanilla meta-interpreter. The Vanilla meta-interpreter is a definite logic program which consists of two parts: a general part M, which consists of an intensional formalisation of derivability by SLD-resolution from definite object programs, and an object program-specific part meta-P, which consists of a meta-level description of the clauses of an object program.

Definition 1.1 The standard *Vanilla meta interpreter* M_P for definite object programs P.

M	[M1]	demo(empty) ←
	[M2]	demo(x) ← clause(x,y), demo(y)
	[M3]	demo(x&y) ← demo(x), demo(y)
meta-P	[M4]	clause(A, B_1&...&B_n) ← for every clause A ← B_1,...,B_n in P
	[M5]	clause(A, empty) ← for every fact A ← in P

[*] I want to thank Frank van Harmelen for many pleasant discussions and careful commenting on drafts; I want to thank Dov Gabbay for discussions on meta-languages.

Example Consider the following object program P:

```
p(c)  ←
r(x)  ←
q(x)  ← p(x)
```

The associated Vanilla meta-interpreter M_P is:

```
demo(empty) ←
demo(x)       ← clause(x,y), demo(y)
demo(x&y)   ← demo(x), demo(y)
clause(p(c), empty)   ←
clause(r(x), empty)   ←
clause(q(x), p(x))     ←
```

Now consider the variables which occur in M_P above. The variables occurring in the second and third clause are intended to range over object level atoms, whereas the variables which occur in the last three clauses are meant to range over the domain of the object program. Thus, intuitively, the Vanilla meta interpreter is a typed program with two types: the variables occurring in the clauses M2 and M3 are intended as meta-variables which range over object level atoms; the variables that occur in the part which represents the object level program, are meant to be object level variables ranging over object level terms. Motivated by the observation that the intuitive interpretation of the Vanilla meta-interpreter is typed, Hill and Lloyd advocate a typed version of the Vanilla meta-interpreter in their paper [HL] and prove its correctness.
However, it is the untyped version of the Vanilla meta interpreter, and extensions of it, that is used in general Prolog-practise and in applications (see [SS] and [KK]). Typically, it does not distinguish between object- and meta-level variables and terms. As a result it employs a non-standard syntax, which is mainly characterised by the occurrence of atoms as terms. This syntax admits atoms like 'demo(c)' and 'clause(q(demo(r(empty))), q(empty))', which are not admissable in any standard syntax. In this paper we will give a definition of this syntax, which was introduced in [R]. We call this syntax 'ambivalent syntax', for the dual role played by atoms. This phenomenon of duality can also be interpreted as overloading of the predicate symbols of the language, which double as function symbols. We will define a suitable (Herbrand style) semantics for first-order logic with ambivalent syntax, which also gives us a (least Herbrand) semantics for the Vanilla meta-interpreter. This enables us to prove satisfying correctness results for the untyped Vanilla meta-interpreter.

The main result of this paper is a proof of the following theorem, which expresses correctness of the Vanilla meta-interpreter with respect to definite object programs, by connecting the least Herbrand model H_P of an object program P with a part of the least Herbrand model H_{M_P} of M_P:

Main Theorem (Theorem 5.10) Let P be a definite program and M_P the Vanilla meta interpreter associated with P. Then for all p(t) in the Herbrand base of P,

$p(t) \in H_P$ iff $demo(p(t)) \in H_{M_P}$.

The part of the main theorem that expresses completeness (if p(t) is derivable from the object program P, then demo(p(t)) is derivable from M_P), has a straightforward proof. Because the Vanilla meta-interpreter may be considered as a formalisation of derivability from a definite object program, each derivation from the object program can be mimicked by a derivation via the associated Vanilla meta-interpreter.
It is not completely trivial, however, to show that M_P is sound with respect to P. The problem partly originates in the ambivalent syntax of the Vanilla meta interpreter, which causes Herbrand models of M_P (even its least Herbrand model) to contain meta-language atoms like demo(p(demo(empty))), which cannot be directly related to object level atoms. The main idea behind the soundness proof is made explicit in Lemma 5.6. Subterms occurring in derivations from the Vanilla meta interpreter which are terms of the meta-language but not terms of the object language, have universal force. I.e., within certain boundaries, such terms can be replaced by arbitrary terms.

The outline of this paper is as follows: In Section 2 we will develop some basic theory of languages with ambivalent syntax. In Section 3 we will define a Herbrand style semantics for these languages. In Section 4 we will study some properties of substitutions in ambivalent syntax which we need in the remainder of the paper. In Section 5 we apply the results of the preceding sections to the theory of the Vanilla meta interpreter, and we prove our main theorem, the correctness result. In Section 6 we discuss our results and related results by Hill and Lloyd [HL] and Martens and De Schreye [SM, MS].

2 Ambivalent Syntax

In this paragraph we will give a formal definition of the syntax underlying the Vanilla meta- interpreter. It has the following main features:

1. There is no typing. No distinction is made between object and meta-level variables, nor between object and meta-level function symbols, predicates, and constants.

2. Atoms may occur as terms. This can be interpreted as a naive naming mechanism: instead of using a code to refer to an atom, one uses the atom proper - atoms serve as their own names (the metalevel representation of an object level expression is the object level expression itself).

The first feature is easily established by taking as the predicates of the language of the Vanilla meta-interpreter the (meta-) predicates 'clause' and 'demo' together with the predicates of the object level program, do the same for the constants, and take as function symbols the function symbols of the object program and the conjunction '&'. We will thus adopt the following convention: The non-logical constants of the underlying language of an object program P are given by a triple (R_P, F_P, C_P), where R_P, F_P, and C_P are, respectively, the set of relation symbols, the set of function symbols and the set of constants occurring in P. The non-logical constants of the underlying language of the associated interpreter M_P are then given by a triple (R_{M_P},F_{M_P},C_{M_P}), where $R_{M_P} = R_P \cup \{\text{demo}(\cdot), \text{clause}(\cdot,\cdot)\}$, $F_{M_P} = F_P \cup \{(\cdot)\&(\cdot)\}$, $C_{M_P} = C_P \cup \{\text{empty}\}$.

The second feature is really where one sees that the syntax involved is non-standard. Because of the double role that formulas play in this syntax, we will call it *ambivalent syntax*. In the Vanilla meta interpreter, there are only two syntactical categories to be dealt with: terms and atoms of the language. We will take a more general approach and define ambivalent syntax for first order predicate logic.

In this paper we will not consider ambivalent syntax for languages with equality.

Definition 2.1 The non-logical constants of a language with ambivalent syntax $\mathcal{L}_{amb}$ are given as a triple ($R_\mathcal{L}$, $F_\mathcal{L}$, $C_\mathcal{L}$), where $R_\mathcal{L}$ is a set of relation symbols, $F_\mathcal{L}$ is a set of function symbols, $C_\mathcal{L}$ is a set of constants. The alphabet of $\mathcal{L}_{amb}$ also contains countably many variables and the usual connectives and auxiliary symbols and one proposition symbol: $\perp$. The sets of atoms, formulas, and terms of $\mathcal{L}_{amb}$ are defined as follows:

$\text{ATOM}_{\mathcal{L}amb}$	(1) p(t) is an atom if $p \in R_{\mathcal{L}amb}$ and $t \in \text{TERM}_{\mathcal{L}amb}$;
	(2) there are no other atoms.
$\text{FORM}_{\mathcal{L}amb}$	(1) $\varphi \in \text{FORM}_{\mathcal{L}amb}$ if $\varphi \in \text{ATOM}_{\mathcal{L}amb}$;
	(2) $\varphi\wedge\psi$, $\varphi\vee\psi$, $\varphi\rightarrow\psi \in \text{FORM}_{\mathcal{L}amb}$ if φ and $\psi \in \text{FORM}_{\mathcal{L}amb}$;

(3) $\bot \in \mathrm{FORM}_{\mathcal{L}\mathrm{amb}}$;

(4) $\neg\varphi \in \mathrm{FORM}_{\mathcal{L}\mathrm{amb}}$ if $\varphi \in \mathrm{FORM}_{\mathcal{L}\mathrm{amb}}$;

(5) $\forall x\varphi, \exists x\varphi \in \mathrm{FORM}_{\mathcal{L}\mathrm{amb}}$ if x is a variable and $\varphi \in \mathrm{FORM}_{\mathcal{L}\mathrm{amb}}$;

(6) there are no other formulas.

$\mathrm{TERM}_{\mathcal{L}\mathrm{amb}}$ (1) $c \in \mathrm{TERM}_{\mathcal{L}\mathrm{amb}}$ for all $c \in C_{\mathcal{L}\mathrm{amb}}$;

(2) $x \in \mathrm{TERM}_{\mathcal{L}\mathrm{amb}}$ for all variables x;

(3) $t \in \mathrm{TERM}_{\mathcal{L}\mathrm{amb}}$ if $t \in \mathrm{FORM}_{\mathcal{L}\mathrm{amb}}$;

(4) $f(t) \in \mathrm{TERM}_{\mathcal{L}\mathrm{amb}}$ if $t \in \mathrm{TERM}_{\mathcal{L}\mathrm{amb}}$ and $f \in F_{\mathcal{L}\mathrm{amb}}$;

(5) there are no other terms.

Note that these definitions differ from the corresponding definitions for standard syntax in the following way: the definition for the term set refers to the set of formulas. Also note that $\mathrm{FORM}_{\mathcal{L}\mathrm{amb}} \subseteq \mathrm{TERM}_{\mathcal{L}\mathrm{amb}}$.

Example Let the nonlogical constants of $\mathcal{L}$ be ({p},{f},{c}). Then p(c), f(p(c)) and f(x) are terms of $\mathcal{L}_{\mathrm{amb}}$, whereas f(x) is a term of $\mathcal{L}$ with the standard syntax, but p(c) and f(p(c)) are not; p(c) is both an atom of $\mathcal{L}$ with standard syntax and of $\mathcal{L}_{\mathrm{amb}}$; p(p(c) $\vee$ p(f(c))) is both an atom and a term of $\mathcal{L}_{\mathrm{amb}}$, but neither of those in $\mathcal{L}$ with standard syntax.

One part of the Unique Readability Property holds for ambivalent syntax: formulas and terms can only be written in one unique way. We wil not give the proof, as it is identical to the corresponding proof for standard syntax. But by the very definition of ambivalent syntax, the part of the Property that states that expressions of the language are either formulas or terms or none of these, does not hold, of course: all formulas are terms. However, if one focuses on occurrences of formulas in well formed expressions, one notes that in each occurrence a formula either serves a formula or as a term, but not as both. A simple way to study occurrences is identifying terms t with term trees. We then define a formula φ to occur in t iff φ is a subtree of t. Specific occurrences of φ in t are identified with specific subtrees t' of t. We define a subtree t' of t to be a term occurrence of φ in t iff t' is equal to φ and there is a node n in t above t' which is a predicate or a function symbol; an occurrence t' of φ in t is a formula occurrence otherwise. By this definition, an occurrence can never be both a term occurrence and a formula occurrence.

Next we give a definition of the set of free variables of terms and formulas, in order to define ground formulas and closed terms. The definitions are standard.

Definition 2.2 The set FV(t) of free variables of a term t is inductively defined as follows:

$FV(x) = x$ for all variables x;

$FV(c) = \emptyset$;

$FV(f(t_1,...,t_n)) = FV(t_1) \cup ... \cup FV(t_n)$, for $f \in F_L$ and $t_1,...,t_n \in TERM_{L_{amb}}$;

$FV(p(t_1,...,t_n)) = FV(t_1) \cup ... \cup FV(t_n)$, for $p \in R_L$ and $t_1,...,t_n \in TERM_{L_{amb}}$;

$FV(\varphi\wedge\psi) = FV(\varphi\vee\psi) = FV(\varphi\rightarrow\psi) = FV(\varphi) \cup FV(\psi)$, for φ and $\psi \in FORM_{L_{amb}}$;

$FV(\neg\varphi) = FV(\varphi)$, for $\varphi \in FORM_{L_{amb}}$;

$FV(\forall x\varphi) = FV(\exists x\varphi) = FV(\varphi)\backslash\{x\}$, for $\varphi \in FORM_{L_{amb}}$.

Definition 2.3 The sets $GRFORM_{L_{amb}}$, $GRATM_{L_{amb}}$ and $CLTERM_{L_{amb}}$ of ground formulas, ground atoms and closed terms of L_{amb} are defined as follows:

$t \in GRFORM_{L_{amb}}$ iff $\varphi \in FORM_{L_{amb}}$ and $FV(\varphi) = \emptyset$;

$t \in GRATM_{L_{amb}}$ iff $\varphi \in ATOM_{L_{amb}}$ and $FV(\varphi) = \emptyset$;

$t \in CLTERM_{L_{amb}}$ iff $t \in TERM_{L_{amb}}$ and $FV(t) = \emptyset$.

3 Semantics for Ambivalent Syntax

The semantics we propose for languages with ambivalent syntax is the class of term models in which every closed term of the language is interpreted by the identity function. Herbrand models fall within this class. With respect to this class, strong completeness holds and can be proven in the standard way (Theorem 3.6). We will assume T to be a consistent first order predicate theory with a language $\mathcal{L}=(R_{\mathcal{L}},F_{\mathcal{L}},C_{\mathcal{L}})$ with ambivalent syntax. We will identify theories with sets of axioms, which are closed formulas; a theory need not be closed under derivability. We work with a standard system of natural deduction, in which the deduction rules are defined in a standard way. Note however that $\forall xp(x) / p(\exists xp(f(x)))$ is now an instance of the standard rule " $\forall x\varphi / \varphi(t)$, for all closed terms t ".

Definition 3.1 A *pre-interpretation* J for $\mathcal{L}$ consists of

a) a *domain* D such that

(1) $CLTERM_{\mathcal{L}} \subseteq D$.

(2) for all $t(x_1,...,x_n) \in TERM_{\mathcal{L}}$ with $x_1,...,x_n$ free,

and for all $d_1,...,d_n \in D$, $t(d_1,...,d_n) \in D$.

b) for each term $t \in CLTERM_{\mathcal{L}}$, the assignment of $t \in D$.

Definition 3.2 An *interpretation* M of $\mathcal{L}$ consists of a pre-interpretation J with domain D and, for each n-ary predicate symbol in $P_{\mathcal{L}}$, the assignment of a subset of D^n.

Definition 3.3 Let M be an interpretation of a language $\mathcal{L}$. Truth of closed formulas of $\mathcal{L}$ in M is inductively defined as follows:

$M \models p(\underline{t})$	for $p \in R_{\mathcal{L}}$ and $\underline{t}$ an n-tuple of closed terms of $\mathcal{L}$, iff (the interpretation of) $\underline{t}$ is in the subset of D^n assigned to p.
$M \models \varphi \wedge \psi$	iff $M \models \varphi$ and $M \models \psi$,
$M \models \neg\varphi$	iff $M \not\models \varphi$,
$M \models \varphi \vee \psi$	iff $M \models \varphi$ or $M \models \psi$,
$M \models \varphi \rightarrow \psi$	iff $M \not\models \varphi$ or $M \models \psi$,
$M \models \forall x \varphi$	iff $M \models \varphi\,[x/\underline{a}]$, for all $a \in D_M$,
$M \models \exists x \varphi$	iff there is an $a \in D_M$ such that $M \models \varphi\,[x/\underline{a}]$.

Definition 3.4 Let T be a theory in a language $\mathcal{L}$. Let M be an interpretation of $\mathcal{L}$. M is a *model* for T if all the axioms of T are true in M.

Definition 3.5 A *Herbrand pre-interpretation* J of $\mathcal{L}$ is a pre-interpretation for which the domain D consists of $CLTERM_{\mathcal{L}}$;

A *Herbrand interpretation* is an interpretation based on a Herbrand pre-interpretation.

A *Herbrand model* is a model based on a Herbrand interpretation.

Theorem 3.6 First order predicate logic with ambivalent language is sound and complete with respect to the semantics defined in Definitions 3.1–3.4.

Proof We leave the checking of soundness to the reader.

The completeness proof we present here is strikingly similar to the completeness for standard syntax.

Let T be a consistent first order predicate theory with a language $\mathcal{L}=(R_{\mathcal{L}},F_{\mathcal{L}},C_{\mathcal{L}})$ with ambivalent syntax. Extend T to a Henkin theory T* in the following way:

Extend the language of T with infinitely many fresh constants c_n. The result will be a new language $\mathcal{L}' = (R_{\mathcal{L}'},F_{\mathcal{L}'},C_{\mathcal{L}'})$, with $R_{\mathcal{L}'} = R_{\mathcal{L}}$, $F_{\mathcal{L}'} = F_{\mathcal{L}}$, while $C_{\mathcal{L}'} = C_{\mathcal{L}} \cup \{c_n : \in \omega\}$.

Let $\varphi_0, \varphi_1, \varphi_2, \ldots$ be an enumeration of all formulas of $\mathcal{L}'$ with only x free. Define C_n as the set of constants that occur in $\varphi_0 \wedge \ldots \wedge \varphi_n$. Define:

$T_0 := T,$

$m_0 := \min\{k : c_k \notin C_0\},$

$T_1 := T_0 \cup \{\exists x \varphi_0(x) \rightarrow \varphi_0(c_{m_0})\},$

$m_{n+1} := \min\{k > m_n : t_k \notin C_n\},$

$T_{n+2} := T_{n+1} \cup \{\exists x \varphi_{n+1}(x) \rightarrow \varphi_{n+1}(c_{m_{n+1}})\},$

$T^* := \cup T_n.$

For all n, consistency of T_{n+1} can be inferred from consistency of T_n in the usual way. So T^* is consistent. Extend T^* to a maximally consistent theory T^m. The language of T^m is $\mathcal{L}'$, the language of T^*. A model M for T^m can now be constructed as follows: Take as the domain D of M the set of closed terms of $\mathcal{L}'$. (As we are dealing with ambivalent syntax, every closed formula of $\mathcal{L}'$ will be an element of D.) Take the identity function for the interpretation function. Define atoms to be true in M iff they belong to T^m. M is a model for T^m (by induction on the complexity of formulas), and thus M is a model for T. Moreover, M is a Herbrand model for T^m and T^*. This concludes the proof. $\boxtimes$

For infinitely extendible theories a model can be constructed in a slightly more elegant way: the closed terms of the language that do not occur as terms in the theory can be used as Henkin constants themselves. The resulting model is a Herbrand model for the theory. Logic programs, as they are finite, are infinitely extendible. Note that theories that are axiomatised with axiom schemes are not infinitely extendible.

A language with ambivalent syntax was first defined by Richards in his paper [R], in the context of intensional predicate logics. Kim and Kowalski [KK] advanced ambivalent langauge as appropriate for the Vanilla meta-interpreter, especially for extensions of it in the field of metalogic programming for multi-agent knowledge, and [R] has become a standard reference in the literature on the Vanilla meta-interpreter. However, the semantics for ambivalent syntax proposed in [R] is flawed, due to the fact that the interpretation function is not well-defined[1]. A broader perspective on ambivalent syntax

[1] The following example shows what is wrong in the semantics proposed in [R]:
Take an ambivalent language $\mathcal{L}$ with one binary relation symbol R, one unary relation symbol Q, one unary function symbol f, and one constant symbol c. Then $\forall x R(x,Q(x)) \rightarrow \forall x R(f(x),Q(f(x)))$ should be

and its semantics is given in [G], where several meta-languages are proposed which allow for great freedom of expression. Among them is HFP, which is essentially first-order predicate language enriched with meta-level predicate and function symbols. Ambivalent syntax can be considered as HFP with one function symbol f that transforms formulas into terms. A semantics for ambivalent syntax which is more general than the Herbrand semantics proposed in the present paper can be obtained by considering models for HFP (but we must leave this matter to a future occasion).

4 Preliminaries on Substitution

A crucial step in our main proof relies on a shift from (sub)terms of the language underlying M_P into terms of the language underlying an object program P. In the present section we will define and discuss the necessary transformation (Definition 4.3). We will define this transformation not for just the languages of M_P and P, but somewhat more generally for two languages of which one comprises the other.
We will identify languages $\mathcal{L}$ with pairs (L,S), where L is the triple indicating the non-logical constants of $\mathcal{L}$, and S indicates the particular syntax of $\mathcal{L}$. For instance, the underlying language $\mathcal{L}_P$ of an object program P is identified with $((R_P, F_P, C_P)$, standard syntax), and the underlying language $\mathcal{L}_{M_P}$ of the associated meta interpreter M_P is identified with $((R_{M_P},F_{M_P},C_{M_P})$, ambivalent syntax). In order to keep matters simple, we will only consider languages which, like $\mathcal{L}_P$ and $\mathcal{L}_{M_P}$, have no logical constants - that is, languages of which the set of formulas consists of atoms only. The theory developed in the present section is thus tailored for languages underlying definite logic programs, but can be generalised to the case of languages with connectives and quantifiers. $\mathcal{L}_P$ can be considered as a part of $\mathcal{L}_{M_P}$: all terms of $\mathcal{L}_P$ are also terms of $\mathcal{L}_{M_P}$, and, likewise, the set of formulas of $\mathcal{L}_P$ is included in the set of formulas of $\mathcal{L}_{M_P}$. We can define this notion of inclusion as follows:

valid in every model. However, one can construct the following model M according to Richards' definitions which validates the antecedent and in which the consequent is wrong:
Take, according to the definition in [R], as a domain $D_M = \{c\} \cup \{\varphi \mid \varphi \in \text{CLFORM}_{\mathcal{L}}\}$. The extension of Q is arbitrary. V(f) is the identity function on D. The extension of R is taken as follows: $V(R) = \{<d,Q(d)> \mid d \in D\}$. By Richards' definition, the interpretation function * is the identity function on constants and closed formulas. Thus $\forall x R(x,Q(x))$ is true in M. However, $(f(c))^* = V(f)(c^*) = c$, while $(Q(f(c)))^* = Q(f(c))$. So $<(f(c))^*,(Q(f(c)))^*> = <c,Q(f(c))>$, which does not belong to V(R).

Definition 4.1 For two languages $\mathcal{L} = (L,S)$ and $\mathcal{L}'=(L',S')$, we say that *$\mathcal{L}$ is a part of $\mathcal{L}'$* ($\mathcal{L} \subseteq \mathcal{L}'$) if

1. $R_L \subseteq R_{L'}$, $F_L \subseteq F_{L'}$, and $C_L \subseteq C_{L'}$;
2. either S is standard syntax and S' is ambivalent syntax, or S and S' are both standard syntax, or S and S' are both ambivalent syntax.

By this definition we have $\mathcal{L}_P \subseteq \mathcal{L}_{MP}$. The property that the sets of terms and formulas of $\mathcal{L}_{MP}$ contain their respective equivalents for $\mathcal{L}_P$ generally holds for two languages of which one is part of the other:

Lemma 4.2 For two languages $\mathcal{L}$ and $\mathcal{L}'$, if $\mathcal{L} \subseteq \mathcal{L}'$ then $TERM_{\mathcal{L}} \subseteq TERM_{\mathcal{L}'}$ and $FORM_{\mathcal{L}} \subseteq FORM_{\mathcal{L}'}$.

Proof Immediate, by the definitions. ⊠

In the remainder of this section we will always suppose $\mathcal{L}$ and $\mathcal{L}'$ to be two languages such that $\mathcal{L} \subseteq \mathcal{L}'$.
If one takes a closer look at the (tree-) structure of a term t of $\mathcal{L}'$, one sees that certain subterms of t are terms of $\mathcal{L}$, while others are proper $\mathcal{L}'$-terms. By substituting in an $\mathcal{L}'$-term t all of its subterms which are proper $\mathcal{L}'$-terms by variables, one can transform t into an $\mathcal{L}$-term. The intuition behind the definition of the *$\mathcal{L}'/\mathcal{L}$-abstraction* of a term t in the language $\mathcal{L}'$, is that one can descend in the termtree of t until one encounters a symbol s that is in $\mathcal{L}'$ but not in $\mathcal{L}$, replace the subtree starting with s with a fresh variable which does not occur in t; this procedure yields a term of $\mathcal{L}$. The $\mathcal{L}'/\mathcal{L}$-abstraction thus transforms terms from $\mathcal{L}'$ to terms from $\mathcal{L}$. Moreover, this is done in a systematic way: equal terms are replaced by equal variables. This makes the abstraction procedure reversible (Lemma 4.5).

Definition 4.3 Let t be a term of $\mathcal{L}'$. The *$\mathcal{L}'/\mathcal{L}$-abstraction of* t, $abs_{\mathcal{L}'/\mathcal{L}}(t)$, is inductively defined as follows:

$abs_{\mathcal{L}'/\mathcal{L}}(x) = x$, for all variables x;

$abs_{\mathcal{L}'/\mathcal{L}}(c) = c$, for all constants $c \in C_{\mathcal{L}}$;

$abs_{\mathcal{L}'/\mathcal{L}}(c) = x_c$, for all constants $c \in C_{\mathcal{L}'} \backslash C_{\mathcal{L}}$, where x_c is a variable;

$abs_{\mathcal{L}'/\mathcal{L}}(f(t_1,...,t_n)) = f(abs_{\mathcal{L}'/\mathcal{L}}(t_1),...,abs_{\mathcal{L}'/\mathcal{L}}(t_n))$, for all function symbols $f \in F_{\mathcal{L}}$;

$abs_{\mathcal{L}'/\mathcal{L}}(f(t_1,...,t_n)) = x_{f(t_1,...,t_n)}$, for all function symbols $f \in F_{\mathcal{L}'} \backslash F_{\mathcal{L}}$, where $x_{f(t_1,\dots,t_n)}$ is a variable;

$abs_{\mathcal{L}'/\mathcal{L}}(R(t_1,\dots,t_n)) = R(abs_{\mathcal{L}'/\mathcal{L}}(t_1),\dots,abs_{\mathcal{L}'/\mathcal{L}}(t_n))$,

for all relation symbols $R \in R_{\mathcal{L}}$;

$abs_{\mathcal{L}'/\mathcal{L}}(R(t_1,\dots,t_n)) = x_{R(t_1,\dots,t_n)}$, for all relation symbols $R \in R_{\mathcal{L}'} \backslash R_{\mathcal{L}}$,

where $x_{R(t_1,\dots,t_n)}$ is a variable.

Implicitely we have assumed that we have extended the set of variables of $\mathcal{L}$ with a set of fresh variables which are indexed by $\mathcal{L}'$-terms. This means that, while our purpose was to get a term of $\mathcal{L}$ as the result of the abstraction, we get a term in a language which is in fact an extension of $\mathcal{L}$. However, by renaming the fresh variables in $abs_{\mathcal{L}'/\mathcal{L}}(t)$, we can get a term of $\mathcal{L}$ proper.

The following lemma mentions two properties of the $\mathcal{L}'/\mathcal{L}$-abstraction of $\mathcal{L}'$-terms which we will need below.

Lemma 4.4 Let t be a term of $\mathcal{L}'$. Then the following holds:

a) $abs_{\mathcal{L}'/\mathcal{L}}(t)$ is unique.

b) $abs_{\mathcal{L}'/\mathcal{L}}(t)$ is, modulo renaming of variables, a term of $\mathcal{L}$.

Proof Immediate from the definition. ⊠

Lemma 4.5 For every term v in $\mathcal{L}'$ there is an $\mathcal{L}$-substitution τ_v, such that

$v = abs_{\mathcal{L}'/\mathcal{L}}(v)\tau_v$.

Proof We define τ_v by induction on the term structure of v.

1) $\tau_c = \emptyset$, for all constants $c \in C_{\mathcal{L}}$;

2) $\tau_c = \{x_c/c\}$, for all constants $c \in C_{\mathcal{L}'} \backslash C_{\mathcal{L}}$;

3) $\tau_x = \emptyset$, for all variables x without an index from $TERM_{\mathcal{L}'}$;

4) $\tau_{f(t_1,\dots,t_n)} = \tau_{t_1} \cup \dots \cup \tau_{t_n}$ for all function symbols $f \in F_{\mathcal{L}}$;

5) $\tau_{f(t_1,\dots,t_n)} = \{x_{f(t_1,\dots,t_n)}/f(t_1,\dots,t_n)\}$ for all function symbols $f \in F_{\mathcal{L}'} \backslash F_{\mathcal{L}}$;

6) $\tau_{R(t_1,\dots,t_n)} = \tau_{t_1} \cup \dots \cup \tau_{t_n}$ for all relation symbols $R \in R_{\mathcal{L}}$;

7) $\tau_{R(t_1,\dots,t_n)} = \{x_{R(t_1,\dots,t_n)}/R(t_1,\dots,t_n)\}$ for all relation symbols $R \in R_{\mathcal{L}'} \backslash R_{\mathcal{L}}$..

We have to check that τ_v is well defined. This is clear for all cases except 3) and 5). Suppose, for case 3) that $v = f(t_1,\dots,t_n)$, for some $f \in F_{\mathcal{L}}$, or, for case 5), that $v=R(t_1,\dots,t_n)$, for some $R \in R_{\mathcal{L}}$. Suppose that for some variable x, $x \in Dom(\tau_{t_i}) \cap Dom(\tau_{t_j})$ for some $i \neq j$. By definition of τ_v, x must be x_s for some $s \in TERM_{\mathcal{L}'}$. Then by definition of τ_{t_i} and τ_{t_i}, $x_s\tau_{t_i} = s = x_s\tau_{t_j}$. This shows that τ_v is well-defined.

Again by induction on the term structure of ground terms v we will prove that for τ_v thus defined, $v = abs_{\mathcal{L}'/\mathcal{L}}(v)\tau_v$.

1) For constants $c \in C_{\mathcal{L}}$, $abs_{\mathcal{L}'/\mathcal{L}}(c)\tau_c = c\emptyset = c$.
2) For constants $c \in C_{\mathcal{L}'}\backslash C_{\mathcal{L}}$, $abs_{\mathcal{L}'/\mathcal{L}}(c)\tau_c = x_c\{x_c/c\} = c$.
3) For variables x without an index from $TERM_{\mathcal{L}'}$, $abs_{\mathcal{L}'/\mathcal{L}}(x)\tau_x = x\emptyset = x$
4) Let $f \in F_{\mathcal{L}}$ and $t_1,...,t_n \in TERM_{\mathcal{L}'}$. Suppose that for all i, $abs_{\mathcal{L}'/\mathcal{L}}(t_i)\tau_{t_i} = t_i$. Then $abs_{\mathcal{L}'/\mathcal{L}}(f(t_1,...,t_n))\tau_{f(t_1,...,t_n)} = f(abs_{\mathcal{L}'/\mathcal{L}}(t_1),...,abs_{\mathcal{L}'/\mathcal{L}\mathcal{L}}(t_n))(\tau_{t_1} \cup ... \cup \tau_{t_n})$. Now, by the same argument that proved τ_v to be well defined, $abs_{\mathcal{L}'/\mathcal{L}}(t_i)(\tau_{t_1} \cup ... \cup \tau_{t_n}) = abs_{\mathcal{L}'/\mathcal{L}}(t_i)\tau_{t_i}$. Thus, by the inductive hypothesis, $f(abs_{\mathcal{L}'/\mathcal{L}}(t_1)\tau_{t_1}, ... , abs_{\mathcal{L}'/\mathcal{L}}(t_n)\tau_{t_n}) = f(t_1,...,t_n)$.
5) Let f be a function symbol in $F_{\mathcal{L}'}\backslash F_{\mathcal{L}}$ and $t_1,...,t_n$ $\mathcal{L}'$-terms. Then by definition of the $\mathcal{L}'$-abstraction of $f(t_1,...,t_n)$ and $\tau_{f(t_1,...,t_n)}$, we have $abs_{\mathcal{L}'/\mathcal{L}}(f(t_1,...,t_n))\tau_{f(t_1,...,t_n)} = x_{f(t_1,...,t_n)}\{x_{f(t_1,...,t_n)}/f(t_1,...,t_n)\} = f(t_1,...,t_n)$.

The remaining cases 6) and 7) have proofs similar to those of 4) and 5), respectively. ⊠

In the following definition, we generalise the concept of term abstraction to the concept of abstraction of a substitution.

Definition 4.6 Let θ be an $\mathcal{L}'$-substitution. The *$\mathcal{L}'/\mathcal{L}$-abstraction of* θ, written as $abs_{\mathcal{L}'/\mathcal{L}}(\theta)$, is defined as follows: $abs_{\mathcal{L}'/\mathcal{L}}(\theta)= \{x/abs_{\mathcal{L}'/\mathcal{L}}(t) : x/t \in \theta\}$.

The following lemma characterises the connection between the $\mathcal{L}/\mathcal{L}'$-abstraction of an $\mathcal{L}'$-ground term t and $\mathcal{L}$-terms s of which t is an instantiation.

Lemma 4.7 Let s be a term of $\mathcal{L}$, t a term of $\mathcal{L}'$, θ an $\mathcal{L}'$-substitution such that $s\theta = t$. Then $s\,abs_{\mathcal{L}'/\mathcal{L}}(\theta) = abs_{\mathcal{L}'/\mathcal{L}}(t)$.

Proof By induction on the term structure of s.

a) s=x, for some variable x. Then $\theta\{x/t\} \in \theta$, so $\{x/abs_{\mathcal{L}'/\mathcal{L}}(t)\} \in abs_{\mathcal{L}'/\mathcal{L}}(\theta)$. So $s\,abs_{\mathcal{L}'/\mathcal{L}}(\theta)=$
$= abs_{\mathcal{L}'/\mathcal{L}}(t)$.
b) s=c, for some $c \in C_{\mathcal{L}}$. Then t=c, so $abs_{\mathcal{L}'/\mathcal{L}}(t)=c$, and $s\,abs_{\mathcal{L}'/\mathcal{L}}(\theta) =c$.
c) $s=f(s_1,...,s_n)$, where the s_i are $\mathcal{L}$-terms and $f \in F_{\mathcal{L}}$. Then $t = f(t_1,...,t_n) = f(s_1,...,s_n)\theta$
$= f(s_1\theta,...,s_n\theta)$. So $s_i\theta = t_i$, for $i \in [1,n]$. And $s\,abs_{\mathcal{L}'/\mathcal{L}}(\theta) = f(s_1,...,s_n)abs_{\mathcal{L}'/\mathcal{L}}(\theta) =$
$= f(s_1\theta^*,...,s_n\theta^*)$. By the inductive hypothesis, $s_i abs_{\mathcal{L}'/\mathcal{L}}(\theta) = abs_{\mathcal{L}'/\mathcal{L}}(t_i)$,
for $i \in [1,n]$. Thus $s\,abs_{\mathcal{L}'/\mathcal{L}}(\theta) = f(abs(t_1), ... , abs(t_n)) = abs(f(t_1,...,t_n))$. ⊠

Lemma 4.7 says: $s\,abs_{\mathcal{L}'/\mathcal{L}}(\theta)=abs_{\mathcal{L}'/\mathcal{L}}(s\theta)$, or, writing θ^* for $abs_{\mathcal{L}'/\mathcal{L}}(\theta)$ and t^* for $abs_{\mathcal{L}'/\mathcal{L}}(t)$: $s\theta^*=(s\theta)^*$.

We extend the usual definition of ground substitution in the following way:

Definition 4.8 A substitution σ is *$\mathcal{L}$-ground* if $\sigma = \{x_i/t_i\text{: } i \in [1,n]\}$, and for all i, t_i is a closed term of $\mathcal{L}$.

The following lemma is an application of a well-known fact about ground substitutions.

Lemma 4.9 Let t be a ground term of $\mathcal{L}'$. Then for every $\mathcal{L}$-ground substitution σ for which $\mathrm{dom}(\sigma) \supseteq \mathrm{Var}(abs_{\mathcal{L}'/\mathcal{L}}(t))$, the following holds: $abs_{\mathcal{L}'/\mathcal{L}}(t)\sigma$ is a ground term of $\mathcal{L}$.

Proof Let t, $\mathcal{L}$, and $\mathcal{L}'$ be as in the suppositions for the lemma. As σ is a ground substitution, with $\mathrm{dom}(\sigma) \supseteq \mathrm{Var}(abs_{\mathcal{L}'/\mathcal{L}}(t))$, $abs_{\mathcal{L}'/\mathcal{L}}(t)\sigma$ is a ground term. As σ is an $\mathcal{L}$-ground substitution, and, by Lemma 4.4.b, $abs_{\mathcal{L}'/\mathcal{L}}(t)$ is an $\mathcal{L}$-term, it follows that $abs_{\mathcal{L}'/\mathcal{L}}(t)\sigma$ is an $\mathcal{L}$-ground term. ⊠

5 Proof of the Main Theorem

We are now ready to apply the machinery set up above to the Vanilla meta-interpreter for definite logic programs. We will have to use some of the basic concepts and results of the theory of definite logic programs. We will mainly adhere to the terminology used in [LL], with the single exception that we denote the least Herbrand model of a program P by H_P, as we have already reserved M_P to indicate the Vanilla meta-interpreter of P. The basic theory for definite programs as developed in [LL, Chapter 2] holds without any restriction for definite programs with ambivalent language, and thus for the Vanilla meta-interpreter, as long as all the relevant concepts are defined with respect to the underlying ambivalent language. (Herbrand models for such programs are the Herbrand models defined in Section 3; the fixpoint operator T is then defined with respect to those models; ground instantiations are ground with respect to the ambivalent language.) The main theorem is basically a result about the relation between the least Herbrand model of a program and the least Herbrand model of its associated Vanilla meta interpreter, and its proof proceeds by comparing the different stages of the respective T-operators. Hence we will be only concerned with ground versions of programs. We remarked in Section 1 that variables which occur in the clause-facts [M4] and [M5] of the Vanilla meta-interpreter (the clauses that describe the object program), can be instantiated with terms from the meta-language. We need the following extension of the usual concept of the ground version of a program.

Definition 5.1 Let P be a definite program. Let $\mathcal{L}$ be a language containing the underlying language of P. With *$\mathcal{L}$-ground*(P) we mean the set of all clauses that are $\mathcal{L}$-ground instantiations of clauses of P. The natural ground version of a program P, $\mathcal{L}_P$-ground(P), will be indicated simply as 'ground(P)'.

The next lemma is an immediate application of Lemma 4.7 and Corollary 4.10 to clauses of definite programs. It establishes some simple relations between a definite program P, its natural ground version, and $\mathcal{L}_{M_P}$-ground(P). We will speak about P-ground and M_P-ground substitutions to refer to respectively $\mathcal{L}_P$-ground and $\mathcal{L}_{M_P}$-ground substitutions. We will leave out the indices of the abstraction operator, i.e., we will write $abs(\cdot)$ instead of $abs_{\mathcal{L}_{M_P}/\mathcal{L}_P}(\cdot)$.

Lemma 5.2 Let P be a definite program. Let $C = p(s_0) \leftarrow p_1(s_1),..., p_n(s_n)$ be a clause of P, let $C' = p(t_0) \leftarrow p_1(t_1),..., p_n(t_n)$ be in M_P-ground(P), le t θ be an M_P-ground substitution with $dom(\theta) \supseteq Var(C)$, such that $C\theta = C'$. Then

a) $C\,abs(\theta) = p(abs(t_0)) \leftarrow p_1(abs(t_1)),..., p_n(abs(t))$;

b) for all M_P-ground substitutions σ with $dom(\sigma) \supseteq ran(abs(\theta))$, $C\,abs(\theta)\sigma$ is in M_P-ground(P);

c) for all P-ground substitutions σ with $dom(\sigma) \supseteq ran(abs(\theta))$, $C\,abs(\theta)\sigma$ is in ground(P).

Proof Part a) follows immediately from Lemma 4.7.
Part b) follows from the definitions.
Part c) follows from a) and Lemma 4.9. ⊠

We will first establish some easy facts about H_{M_P}, the least Herbrand model of M_P. We have to assume that the language of the object programs we consider does not contain any of the predicate and function symbols that are proper to the language of M_P, that is, '&', 'demo', and 'clause' do not belong to $P_{\mathcal{L}_P}$ or $F_{\mathcal{L}_P}$. If that is the case, the least Herbrand model of M_P does not contain atoms of the form demo(demo(t)) and demo(clause(t,s)):

Lemma 5.3 If $demo(p(t_1,...,t_n)) \in H_{M_P}$, and p is not &, then $p \in P_{\mathcal{L}_P}$.
Proof Suppose $demo(p(t_1,...,t_n)) \in H_{M_P}$. Then it must have 'entered' the Herbrand model at some stage, that is, there is a d such that $demo(p(t_1,...,t_n)) \in T_{M_P}\uparrow d \setminus T_{M_P}\uparrow(d-1)$. If p

is not & , demo($p(t_1,...,t_n)$) must have entered by application of the meta-clause [M2], so clause($p(t_1,...,t_n)$, C) and demo(C) are in $T_{M_P}\uparrow(d-1)$ for some C. That means that $p(t_1,...,t_n) \leftarrow C$ belongs to M_P-ground(P), or, if C is the constant empty, $p(t_1,...,t_n) \leftarrow$ belongs to M_P-ground(P). But then p must be a predicate from the language of P. ⊠

Lemma 5.4 If demo(A&B) $\in T_{M_P}\uparrow n$, for some n, then there are $m, k < n$ such that demo(A) $\in T_{M_P}\uparrow m$ and demo(B) $\in T_{M_P}\uparrow k$.

Proof Let demo(A&B) $\in T_{M_P}\uparrow n$. There is a $d \leq n$ such that demo(A&B) $\in T_{M_P}\uparrow d \backslash T_{M_P}\uparrow(d-1)$, i.e., d is the stage where demo(A&B) enters the Herbrand model. Now suppose there are no $m, k < n$ such that demo(A) $\in T_{M_P}\uparrow m$ and demo(B) $\in T_{M_P}\uparrow k$. Then demo(A&B) cannot have entered the Herbrand model by application of [M3], as that would require demo(A) and demo(B) in $T_{M_P}\uparrow(d-1)$. So demo(A&B) must have entered by an application of [M2]. That is, clause(A&B,C) and demo(C) are in $T_{M_P}\uparrow(d-1)$ for some C. So $A\&B \leftarrow C$ belongs to M_P-ground(P) or, if C is the constant empty, $A\&B \leftarrow$ belongs to M_P-ground(P). This however contradicts the assumption that & is not a predicate symbol in the language of P. ⊠

Corollary 5.5 If demo($A_1\&...\&A_k$) $\in T_{M_P}\uparrow n$, for some n, then demo(A_i) $\in T_{M_P}\uparrow(n-i)$, for some $i \in [1,k]$, and demo($A_{i+1}\&...\&A_k$) $\in T_{M_P}\uparrow(n-i)$, for some $i \in [2, k-1]$.

Proof By Lemma 5.4 and an easy induction argument. ⊠

The next lemma is crucial for the proof of the soundness part of the main theorem. It expresses the main idea behind the soundness proof. Consider a derivation from the Vanilla meta-interpreter starting with an atom demo(p(t)) in which p is a predicate from $\mathcal{L}_P$. Then all subterms occurring in this derivation which are terms of the metalanguage but not terms of the object language, can, with certain restrictions, be interpreted as being universally quantified. I.e., subterms which are not object language terms can be replaced by arbitrary terms – under the condition that, in the derivation as a whole, equal metalanguage subterms are replaced by equal terms.

Lemma 5.6 Let P be a definite program, p a predicate from the language underlying P, t a term from $\mathcal{L}_{M_P}$. Suppose demo(p(t)) $\in T_{M_P}\uparrow n$, for some n. Then for all M_P-ground substitutions σ with $\mathrm{dom}(\sigma) \supseteq \mathrm{Var}(\mathit{abs}(t))$, demo($p(\mathit{abs}(t)\sigma)$) $\in T_{M_P}\uparrow n$.

Proof Let P be a definite program, p a predicate from the language underlying P, t a term from the language of M_P. The proof will be by course of values induction on the stages of the T_{M_P}-operator.

• $T_{M_P}\uparrow 1$ only contains the atom demo(empty) and atoms of the form clause(t,s), so the first stage of the T_{M_P}-operator in which atoms of the form demo(p(t)) occur is $T_{M_P}\uparrow 2$.
• Suppose demo(p(t)) $\in T_{M_P}\uparrow 2$. Then there must be an atom clause(p(t), empty) in $T_{M_P}\uparrow 1$. So we must have

1) clause(p(t), empty) $\leftarrow$ $\in$ ground(M_P),
2) p(s) $\leftarrow$ in P , for some s $\in$ TERM$_{\mathcal{L}_P}$,
3) an M_P-ground substitution θ such that $s\theta = t$.

By Lemma 4.7, s*abs*(θ) = *abs*(t). Let σ be an M_P-ground substitution with dom(σ) $\supseteq$ Var(*abs*(t)). By Lemma 5.2.b we have that clause(p(*abs*(t)σ), empty) $\leftarrow$ $\in$ ground(M_P). So by definition of the T_{M_P}-operator, clause(p*abs*(t)σ), empty) $\in T_{M_P}\uparrow 1$, and thus demo(p*abs*(t)σ)) $\in T_{M_P}\uparrow 2$.

Suppose that for all $m < n$, the following holds:

If demo(p(t)) $\in T_{M_P}\uparrow m$, and σ is an M_P-ground substitution with dom(σ) $\supseteq$ Var(*abs*(t)), then demo(p(*abs*(t)σ) $\in T_{M_P}\uparrow m$. [IH]

• Suppose demo(p(t)) $\in T_{M_P}\uparrow n$. Let σ be an M_P-ground substitution with dom(σ) $\supseteq$ Var(*abs*(t)). By definition of the T_{M_P}-operator, there are $p_1,...,p_k \in P_{\mathcal{L}_P}$, with $\mathcal{L}_P$-terms $s,s_1,...,s_k$, and an M_P-ground substitution θ such that

1) $p(s) \leftarrow p_1(s_1),...,p_k(s_k) \in P$;
2) $(p(s) \leftarrow p_1(s_1),...,p_k(s_k))\theta = p(t) \leftarrow p_1(t_1),...,p_k(t_k)$;
3) clause(p(t), $p_1(t_1)$&...&$p_k(t_k)$) $\in T_{M_P}\uparrow(n\text{-}1)$;
4) demo($p_1(t_1)$&...&$p_k(t_k)$) $\in T_{M_P}\uparrow(n\text{-}1)$.

Now let σ be an M_P-ground substitution with dom(σ) $\supseteq$ Var(*abs*(θ)), that is, dom(σ) $\supseteq$ Var(*abs*(t)) $\cup$ Var(*abs*(t_1)) $\cup$ ... $\cup$ Var(*abs*(t_k)). By 1), 2), and Lemma 5.2.b we have that p(*abs*(t)σ) $\leftarrow$ p_1(*abs*(t_1)σ),..., p_k(*abs*(t_k)σ) $\in$ M_P-ground(P). So clause(p(*abs*(t)σ), p_1(*abs*(t_1)σ) & ... & p_n(*abs*(t_n)σ)) $\in T_{M_P}\uparrow 1$, and hence in $T_{M_P}\uparrow(n\text{-}1)$. By 4) and Corollary 5.5 we know that demo($p_i(t_i)$) $\in T_{M_P}\uparrow(n\text{-}1\text{-}i)$, for $i \in [1,k]$. From the inductive hypotheses we know that demo(p_i(*abs*(t_i)σ)) $\in T_{M_P}\uparrow(n\text{-}1\text{-}i)$ for $i \in [1,k]$. By applying the M_P-clause demo(A&B) $\leftarrow$ demo(A), demo(B) k-1 times we have demo(p_1(*abs*(t_1)σ)&...&p_n(*abs*(t_k)σ)) $\in T_{M_P}\uparrow(n\text{-}1)$.
By definition of the T_{M_P}-operator, we may conclude that p(*abs*(t)σ) $\in T_{M_P}\uparrow n$.

⊠

Corollary 5.7 Let P be a definite program, $p_1,...,p_k \in P_{\mathcal{L}_P}$, $t_1,...,t_k$ terms in $\mathcal{L}_{M_P}$. Suppose demo($p_1(t_1)$&...&$p_k(t_k)$) $\in T_{M_P}\uparrow n$, for some n. Let σ be an M_P-ground substitutions with dom(σ) $\supseteq$ Var(*abs*(t_1)) $\cup$ ... $\cup$ Var(*abs*(t_k)). Then demo(p_1(*abs*(t_1)σ) & ... & p_n(*abs*(t_k)σ) $\in T_{M_P}\uparrow n$.

We are now ready to prove the main theorem. We will split the proof in two parts. Lemma 5.8 is the soundness part, whose proof heavily depends on Lemma 5.6. and its corollary. Lemma 5.9 is the completeness part. With B_P we indicate the Herbrand base of a definite program P.

Lemma 5.8 (Soundness of M_P with respect to a definite program P)

Let P be a definite program and M_P the Vanilla meta interpreter associated with P. Then for all $p(t) \in B_P$, $\forall n \in \mathbb{N}$ $[demo(p(t)) \in T_{M_P}\uparrow n \Rightarrow \exists m \in \mathbb{N}.\ p(t) \in T_P\uparrow m]$.

Proof By induction on n. Let P be a definite program and M_P the Vanilla meta-interpreter associated with P. Let $p(t) \in B_P$ throughout the proof. Again, the base case of the induction is $n = 2$.

• Suppose $demo(p(t)) \in T_{M_P}\uparrow 2$. Then $clause(p(t), empty) \in T_{M_P}\uparrow(1)$. So $clause(p(t), empty) \leftarrow$ is an M_P-ground instance of a clause of M_P. So there must be a P-term s such that $p(s) \leftarrow\ \in P$, and an M_P-ground instantiation θ such that $s\theta = t$. However, by the assumption that $p(t) \in B_P$, θ must be a P-ground substitution. So $p(t) \leftarrow\ \in ground(P)$. Thus, by definition of the T_P-operator, we can conclude that $p(t) \in T_P\uparrow 1$.

• Suppose that for some n>2, $demo(p(t)) \in T_{M_P}\uparrow n$, and suppose that, for all $p'(t') \in B_P$,

$\forall n'<n\ [demo(p'(t')) \in T_{M_P}\uparrow n' \Rightarrow \exists m \in \mathbb{N}.\ p'(t') \in T_P\uparrow m]$ [IH]

If $demo(p(t)) \in T_{M_P}\uparrow n'$, for some n'<n, we immediately have $\exists m \in \mathbb{N}.\ p(t) \in T_P\uparrow m$. So suppose that for all n'<n, $demo(p(t)) \notin T_{M_P}\uparrow n'$. That means that $clause(p(t), empty) \notin T_{M_P}\uparrow(1)$, i.e. demo(p(t)) must have entered the Herbrand model by an application of [M3]. So there must be ground atoms $B_1,...,B_k \in B_{M_P}$, such that

1) $clause(p(t), B_1\&...\&B_k) \in T_{M_P}\uparrow(1)$,

2) $demo(B_1\&...\&B_k) \in T_{M_P}\uparrow(n-1)$.

As the B_i are terms in $\mathcal{L}_{M_P}$, we cannot expect to apply the inductive hypothesis [IH] directly to the atoms $demo(B_i)$. But from 1), it follows that there are $p_1,...,p_n \in P_{\mathcal{L}P}$, and P-terms $s,s_1,...,s_n$, and an M_P-ground substitution θ such that

3) $p(s) \leftarrow p_1(s_1),...,p_k(s_k) \in P$,

4) $(p(s) \leftarrow p_1(s_1),...,p_k(s_k))\theta = p(t) \leftarrow B_1,...,B_k$.

Let, for $i \in [1,k]$, $t_i = s_i\theta$, i.e. $B_i = p_i(t_i)$. Now take a P-ground substitution σ such that $dom(\sigma) = ran(\mathit{abs}(\theta))$. Note that, by the assumption that $p(t) \in B_P$, $t = \mathit{abs}(t) = \mathit{abs}(t)\sigma$. By Lemma 5.2.c we know that

$p(t) \leftarrow p_1(s_1\mathit{abs}(\theta)\sigma),...,p_k(s_k\mathit{abs}(\theta)\sigma) \in ground(P)$. [*]

Also, by Corollary 5.7, $demo(p_1(\mathit{abs}(t_1)\sigma)\&...\&p_n(\mathit{abs}(t_k)\sigma)) \in T_{M_P}\uparrow n$. And by Lemma 5.5, $demo(p_i(\mathit{abs}(t_i)\sigma) \in T_{M_P}\uparrow n$ for $i \in [1,k]$. As $p_i(\mathit{abs}(t_i)\sigma) \in B_P$, we can apply the inductive hypothesis to the atoms $demo(p_i(\mathit{abs}(t_i)\sigma)$:

$\exists m_1, \ldots, m_k \in \mathbb{N}.\ p_i(\mathit{abs}(t_i)\sigma) \in T_P{\uparrow}m_i]$. [**]

Now by [*] and [**] and the definition of the T_P-operator, we can conclude that there is an m such that $p(t) \in T_P{\uparrow}m$. ⊠

The proof of the following completeness lemma is inspired by De Schreye and Martens, who proved this result in [SM] for a version of the Vanilla meta-interpreter with a different underlying language.

Lemma 5.9 (Completeness of M_P with respect to a definite program P)

Let P be a definite program and M_P the Vanilla meta interpreter associated with P. Then for all $p(t) \in B_P$, $\forall n \in \mathbb{N}\ [p(t) \in T_P{\uparrow}n \Rightarrow \exists m \in \mathbb{N}.\ \mathrm{demo}(p(t) \in T_{M_P}{\uparrow}m]$.

Proof By course of values induction on n. Let $p(t) \in B_P$.

The case n=0 holds trivially by definition of $T_P{\uparrow}0$ as the empty set.

• Suppose $p(t) \in T_P{\uparrow}1$. Then $p(t) \leftarrow$ is a ground instance of a clause of P. So clause(p(t), empty) $\leftarrow$ is a ground instance of a clause of M_P, and clause(p(t), empty) belongs to $T_{M_P}{\uparrow}1$. Also, demo(empty) $\in T_{M_P}{\uparrow}1$, and demo(p(t)) $\leftarrow$ clause(p(t), empty), demo(empty) is a ground instance of [M3]. Thus, by definition of the T_P-operator, demo(p(t)) $\in T_{M_P}{\uparrow}2$.

• Suppose $p(t) \in T_P{\uparrow}n$. And suppose that, for all $p'(t') \in B_P$,

$\forall n'{<}n\ [p(t) \in T_P{\uparrow}n' \Rightarrow \exists m \in \mathbb{N}.\ \mathrm{demo}(p(t)) \in T_{M_P}{\uparrow}m]$. [IH]

If there is an n'<n such that $p(t) \in T_P{\uparrow}n'$, then by [IH] $\exists m \in \mathbb{N}.\ \mathrm{demo}(p(t)) \in T_{M_P}{\uparrow}m]$. So assume that for all n'<n, $p(t) \notin T_{M_P}{\uparrow}n'$. Then there are $p_1(s_1),\ldots,p_k(s_k) \in B_P$, $(k \geq 1)$ such that $p(t) \leftarrow p_1(s_1),\ldots,p_k(s_k) \in \mathrm{ground}(P)$ and $p_i(s_i) \in T_P{\uparrow}(n-1)$ for $i \in [1,k]$. Thus from [IH] we can conclude that there are m1,...,mk, such that demo($p_i(s_i)$) $\in T_{M_P}{\uparrow}m_i$ for $i \in [1,k]$. By monotonicity of the T_P-operator, demo($p_i(s_i)$) $\in T_{M_P}{\uparrow}m'$ for all $i \in [1,k]$ and for all $m' \geq \max(m1,\ldots,mk)$. So, applying [M3] k-1 times, we find an m' such that demo($p_1(s_1)$&...&$p_k(s_k)$) $\in T_{M_P}{\uparrow}m'$. Also, clause(p(t), $p_1(s_1)$&...&$p_k(s_k)$)) $\leftarrow$ belongs to ground(M_P). Therefore, clause(p(t), $p_1(s_1)$&...&$p_k(s_k)$) $\in T_{M_P}{\uparrow}m'$. So by an application of [M2], demo(p(t)) $\in T_{M_P}{\uparrow}(m'+1)$. ⊠

Theorem 5.10 (MAIN THEOREM) Let P be a definite program and M_P the Vanilla meta-interpreter associated with P. Then for all $p(t) \in B_P$,

$p(t) \in H_P$ iff $\mathrm{demo}(p(t)) \in H_{M_P}$.

Proof Combine Lemmas 5.8 and 5.9. ⊠

6 Related Results and Discussion

In this section, we will compare our results with other approaches and results from the literature. First, in paragraph 6.1, we discuss the approach of De Schreye and Martens [SM, MS], who have an interesting result for the Vanilla meta interpreter of a subclass of the class of definite programs and a version of the main theorem in the context of S-semantics. In paragraph 6.2 we discuss a typed version of the Vanilla meta interpreter, as proposed by Hill and Lloyd [HL].

6.1 De Schreye and Martens

In their work on the Vanilla meta interpreter [MS] and [SM], Martens and De Schreye propose two approaches. In [SM] they concentrate on the class of 'language independent' programs, for which they prove a strengthening of our main result. A definite object program P is *language independent* if for all standard extensions $\mathcal{L}$ of $\mathcal{L}_P$, the least Herbrand model for P with respect to $\mathcal{L}$ is equal to the least Herbrand model for P with respect to $\mathcal{L}_P$ [SM, Definitions 2.1– 2.3]. For this class the following strengthening of our main theorem for the Vanilla meta-interpreter holds:

Theorem 6.1 [SM, Theorem 2.11] Let P be a language independent program and M_P the Vanilla meta-interpreter associated with P. Then for all $p \in R_P$ and for all $t \in U_{M_P}$,
$$demo(p(t)) \in H_{M_P} \text{ iff } p(t) \in H_P.$$

By Lemma 5.3 this result can be strengthened in the following way:

Corollary 6.2 Let P be a language independent program and M_P the Vanilla meta-interpreter associated with P. Then for all $p(t) \in B_{M_P}$,
$$demo(p(t)) \in H_{M_P} \text{ iff } p(t) \in H_P.$$

Theorem 6.1 expresses that for language independent programs, if an atom demo(p(t)) belongs to the least Herbrand model of M_P, then t is necessarily an object level term: this property does not hold for all definite programs. Thus Theorem 6.1 shows that there is a one-to-one correspondence between atoms p(t) in the least Herbrand model of a definite language independent program P and atoms demo(p(t)) in the least Herbrand model of the associated Vanilla meta-interpreter M_P. This correspondence does not hold for general definite programs.

The class of language independent programs essentially consists of the 'range restricted' programs: consider, for a language independent program P, the program P* which consists of the range restricted clauses of P. P and P* have the same success set and both programs have the same set of computed answer substitutions. Moreover, while the class of range restricted programs is syntactically defined, the class of language independent programs is undecidable.

De Schreye and Martens prove their main result for Vanilla meta-interpreters which do not use ambivalent syntax, but, instead, a standard syntax in which the relations of the object level language are represtented as functions. They remark that their results also hold for Vanilla meta interpreters with ambivalent syntax, but do not pursue the subject in detail.

After this paper was completed, we learnt about a new paper [MS]. In that paper the Vanilla meta-interpreter for definite programs is studied in the context of S-semantics, which was introduced in [FLPM] as a declarative semantics which fully characterises the procedural behaviour of definite programs. S-semantics for definite programs are Herbrand models which can contain non-ground atoms. In an S-model, atoms which are variable renamings of each other are considered equal. Every definite program P has a unique least S-Herbrand model $H^s{}_P$, which can be obtained as the least fixpoint of a suitable T-operator. The main properties of $H^s{}_P$ are the following:
a) H_P is the set of ground instantiations of atoms in $H^s{}_P$;
b) if θ is a computed answer substitution for $P \cup \{\leftarrow A\}$, then $A\theta$ is an instantiation of an atom A' that belongs to $H^s{}_P$;
c) if A and A' are atoms such that A' belongs to $H^s{}_P$ and $A\theta = A'$, with $dom(\theta)=Var(A')$, then θ is a computed answer substitution for $P \cup \{\leftarrow A\}$.
The main theorem for the Vanilla meta-interpreter for definite programs in the context of S-semantics reads as follows:

Theorem 6.3 [MS, Theorem 8.15] For all $p \in R_P$ and for all t in $TERM_{\mathcal{L}M_P}$,
demo(p(t)) $\in H^s{}_{M_P}$ iff p(t) $\in H^s{}_P$.

Also in this case our Lemma 5.3 applies. And again, this result establishes a one-to-one correspondence beween atoms p(t) in the least S-Herbrand models of, respectively, a definite program P and the asociated interpreter M_P. By this theorem the procedual aspects of the Vanilla meta interpreter can be characterised more fully, as follows:

Theorem 6.4 For all $p \in R_P$ and for all t in $TERM_{\mathcal{L}M_P}$,

θ is a computed answer substitution for $M_P \cup \{\leftarrow \text{demo}(p(t))\}$
iff θ is a computed answer substitution for $P \cup \{\leftarrow p(t)\}$.

This follows from Theorem 6.3 and b) and c) above.
Our main theorem 5.10 can be obtained from Theorem 6.4 by only considering substitutions with an $\mathcal{L}_P$-ground range. (It is impossible however, to compute the set of pairs of goals and computed answer substitutions from the success set of a program. So clearly one can not derive Theorems 6.3 and 6.4 from our version of the main theorem.)
Lemma 5.6, which shows that terms from the meta-language that occur as subterms in atoms in the least Herbrand model of M_P, have universal force, can also be derived from Theorem 6.4.
The value of our main result 5.10 still lies in the fact that it shows there is a suitable *standard* Herbrand semantics for the Vanilla meta interpreter for definite programs, which is based on ambivalent syntax, and with respect to which correctness can be proved. Together with Lemma 5.6, our main result exhaustively treats the declarative aspects of the Vanilla meta interpreter for definite programs. Also, it shows that ambivalent syntax is useful and appropriate in the context of meta interpreters.

6.2 Hill and Lloyd: A typed Vanilla meta-interpreter

Motivated by the observation that the declarative meaning of the Vanilla meta-interpreter is unclear because the variables in the definition of demo and those in the definition of clause intuitively range over different domains, Hill and Lloyd [HL] propose a typed version of the Vanilla meta interpreter for normal programs. (In the same article Hill and Lloyd propose a typed meta interpreter which uses non-ground representation of object level variables in the meta-level.) [HL] has two results on the typed interpreter, which together establish its correctness.
The first result is that the completion of V_P, the typed Vanilla meta-interpreter, is consistent iff the completion of P is consistent. This result implies that, for normal goals G and substitutions t, t is a correct answer for $P \cup \{G\}$ iff t' is a correct answer for $V_P \cup \{\text{demo}(G')\}$. (Here t' and G' are the translations of t and G in the typed meta-language.) The technique used to get this result is a transformation of a model I for the completion of P into a model I^* for the completion of V_P and vice versa. Although not discussed in [HL], this transformation technique could be used for the typed Vanilla meta-interpreter for definite programs, to transform H_{V_P} into H_P (one can easily

establish $H_P{}^* = H_{VP}$) and vice versa (for the inverse transformation $^\sim$ the following holds: $H_{VP}{}^\sim = H_P$).
The second result is procedural correctness of the typed Vanilla meta-interpreter (a version of Theorem 6.4). This result is obtained by transformations of finitely failed trees for the meta-interpreter to finitely failed trees for its object program and vice versa, using a partial evaluation technique. The same strategy can be used for the untyped Vanilla meta-interpreter using ambivalent syntax; this gives a slightly weaker result because of the 'pollution' of the computed answer substitutions with meta-level terms. Typing typically prevents such pollution.

In general Prolog practice the untyped Vanilla meta-interpreter is used, rather than the typed version, and here our main result is of practical importance. Of course, it is the extensions of the Vanilla meta interpreter, and not so much the Vanilla meta-interpreter itself, where the real interest lies. The correctness of the Vanilla meta-interpreter, demonstrated by the different versions of the main theorem, in fact shows that the Vanilla meta-interpreter proves no more nor less than the object program. In the present paper we do not have any results on extensions of the Vanilla meta-interpreter, but we feel confident that the theory developed here can be used to get the desired results for extensions. An interesting variety of extensions of the Vanilla meta-interpreter is formed by the 'amalgamated' programs. The language of the typed Vanilla meta-interpreter is less suitable for amalgamated programs, whereas the ambivalent syntax underlying the untyped interpreter is especially suitable for amalgamation. Amalgamated programs contain clauses in which both meta- and object level atoms appear. Typical examples of amalgamation are reflection clauses, e.g. $demo(X) \leftarrow X$ and $X \leftarrow demo(X)$. Kim and Kowalski [KK] propose a formalisation of multi-agent knowledge and belief in the logic programming format. An extension of the Vanilla meta interpreter with a binary demo-predicate formalises knowledge of agents (demo(a,b) for "agent a knows b"), in the context of the three wise man problem. Employing ambivalent syntax, the reflexive knowledge axiom can be formalised as a clause $X \leftarrow demo(a,X)$. The language of the typed Vanilla meta interpreter is hardly suitable for those applications, as it can not identify the two variables X.

References

[FLPM] M. Falaschi, G. Levi, C. Palamidessi, and M. Martelli, ' Declarative modeling of the operational behaviour of logic languages', *Theoretical Computer Science* 69 (1989), pp. 289–318.

[G] D.M. Gabbay, 'Metalevel features in the object level: modal and temporal logic programming III', in: *Intensional Logics for Programming*, Eds. L. Fariñas del Cerro and M. Penttonen, Clarendon Press, 1992, pp. 85–124.

[HL] P.M. Hill and J.W. Lloyd, 'Analysis of Meta-Programs', in: *Meta-Programming in Logic Programming*, Eds. Harvey Abramson and M.H. Rogers, MIT Press, 1989, pp. 23–52.

[KK] J.S. Kim and R.A. Kowalski, 'A metalogic programming approach to multi-agent knowledge and belief', *Artificial Intelligence and Mathematical Theory of Computation*, Ed. V. Lifschitz, Academic Press, 1991.

[LL] J.W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, 1987.

[MS] B. Martens and D. De Schreye, 'Why untyped non-ground meta-programming is not (much of) a problem', Report CW 159, Department of Computer Science, K.U. Leuven, Belgium, 1992.

[R] B. Richards, 'A Point of Reference', *Synthese* 28 (1974), pp. 431–445.

[SM] D. De Schreye and B. Martens, 'A sensible least Herbrand Semantics for untyped Vanilla meta-programming and its extension to a limited form of amalgamation', in: *Proceedings of the 3rd International Workshop on Meta-Programming in Logic*, Ed. A. Pettorossi, Springer-Verlag, 1993, pp.127–141.

[SS] L. Sterling and E. Shapiro, *The Art of Prolog*, MIT Press, 1986.