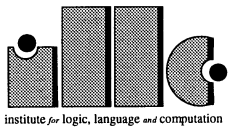


MARIANNE KALSBEEK

A Vademecum of Ambivalent Logic

CT-95-01, *received*: Januari 1995

ILLC Research Report and Technical Notes Series
Series editor: Dick de Jongh

Computation and Complexity Theory (CT) Series, ISSN: 0928-3323

Institute for Logic, Language and Computation (ILLC)
University of Amsterdam
Plantage Muidergracht 24
NL-1018 TV Amsterdam
The Netherlands
e-mail: illc@fwi.uva.nl

A Vademecum of Ambivalent Logic

Marianne Kalsbeek*
ILLC
Plantage Muidergracht 24
1018 TV Amsterdam
email: marianne@fwi.uva.nl

Yuejun Jiang†
Department of Computing
Imperial College, London SW7 2AZ
email: yj@doc.ic.ac.uk

Abstract

Ambivalent Logic AL, first introduced in its full sense in Jiang [Jia94a] is obtained from first order predicate logic FOL by relaxing several restrictions on its usual syntax. In particular, the usual distinctions between predicates, functions, formulas and terms are not made in AL. We show that Ambivalent Logic provides a general and flexible framework for various ambivalent syntactic phenomena that occur in Prolog, in meta-logic programming and in formalisations of knowledge and belief. A series of formal results justifies the use of syntaxes with ambivalent phenomena in these areas. We discuss a closed term semantics for AL, and show that the standard derivational calculus for first order predicate logic is sound and complete w.r.t. this semantics. A conservativity result shows that AL should be considered as a conservative extension of FOL. We define a version of the Martelli Montanari unification algorithm for AL, and show it has the usual properties. In combination with various other basic proof theoretic results, this shows that resolution is a complete and sound inference method for AL. We also discuss the relation with Hilog.

Keywords Prolog, meta-logic programming, first order predicate logic, unification, Ambivalent Logic

Note This research was partly supported by the ESPRIT Basic Research Action 6810 (Compulog 2). This paper will appear in: Meta-Programming in Logic Programming, Eds. K.R. Apt and F. Turini, The MIT Press, 1995.

*Supported by the Netherlands organisation for Scientific Research N.W.O.
†Advanced Fellow of the British Science and Engineering Research Council

1 Introduction

A (mostly tacit) assumption in the syntax for first order predicate logic (FOL) is that the sets of predicates, functions, and constants have mutually empty intersections. As a consequence, the syntactic categories of formulas and terms are mutually exclusive, as the Unique Reading Lemma for FOL witnesses. While Logic Programming is in principle based on FOL, in various of its application areas, syntaxes are used which do not satisfy this property of the usual syntax for FOL.

A principal example, analysed in Apt and Teusink [AT], is the syntax underlying Prolog. Prolog's meta-variable facility allows for variables to occur both in terms and in atoms positions. Two well-known examples are the cut-fail definition of negation: $neg(X) :- X, !, fail$, and the definition of the solve-predicate as $solve(X) :- X$. Clearly, this use of variables is not allowed in the standard syntax of FOL. Additionally, queries which involve instantiations of the heads of the above clauses with terms $p(t)$, presuppose a syntax which allows for function symbols to be accepted as predicate symbols. Conversely, an inductive definition of a predicate involving its negation presupposes that Prolog's syntax allows for predicate symbols to be accepted as function symbols. Also, in Prolog's syntax, predicates and functions do not have fixed arities.

Another example of the use of a deviant syntax in (meta-)Logic Programming practice is the untyped, non-ground Vanilla meta-interpreter which uses identity naming for atoms and terms of the object level language (cf. Kalsbeek [Kal]). While in this simple, unamalgamated case the (partial) correctness of Vanilla could still be guaranteed by the possibility of renaming the object level predicates to meta-level function symbols, this solution does not generalise to various interesting extensions of Vanilla. For example, extending Vanilla with reflective clauses like $demo(X) :- X$ or instances of these clauses, eliminates the possibility of renaming and introduces ambivalence between function and predicate symbols and variables and terms, similar to the ambivalences observed above in the case of Prolog's syntax. A related example is the formalisation of the Three Wise Man problem as an extension of Vanilla in Kowalski and Kim [KK91]. This formalisation involves clauses of the form $demo(wise0, not\ demo(wise1, white1)) :-$, in which the predicates *demo* and *not* occur in function positions.

The above examples require only limited forms of ambivalence: atoms-as-terms ambivalence (atoms occurring in term positions) and terms-as-atoms ambivalence (terms occurring in atom positions). As observed in Chen et al. [CKW93] and Jiang [Jia94a], more advanced ambivalent features are required to obtain a syntax which allows for efficient formalisations of generic predicates. An example is the generic closure predicate $(cl(Z))(X, Y)$, which, given any binary predicate R, returns its transitive closure $cl(R)$. Its definition, $((cl(Z))(X, Y) \leftrightarrow Z(X, Y) \vee (Z(X, V) \wedge (cl(Z))(V, Y))$, presupposes a syntax which allows for predicates to occur in term positions, and for variables and

atoms to occur in predicate positions. Similar ambivalent phenomena occur in languages which allow for data retrieval and schema browsing. In databases, such forms of syntactic ambivalence are a desirable option, allowing caching of data.

Other areas where ambivalent phenomena occur, are formalisations of knowledge and belief. While the use of a real naming function, avoiding syntactic ambivalence between terms and formulas, is often possible, it is not always desirable. Also, the above atoms-terms ambivalence is not always sufficient. In the predicate case, it is desirable that a reflective axiom like $x \rightarrow K(x)$ can be instantiated with all formulas, not just atomic formulas. This then supposes a syntax in which all formulas, and not just atomic formulas, are allowed to occur as terms. While the quantifiers could, in principle, be represented by functions, the effect of a functional representation does not in all cases give the desired effect. The obvious reason is that quantifiers, in contrast to functions, bind variables. For example, consider the formula $\forall x.bel(John, (friend(John, x) \rightarrow exists(y, loves(x, y)))$, which expresses the proposition 'John believes about all his friends that they are loved by somebody'. Instantiation of both x and y with the same constant is possible, yielding unintended statements like

$$bel(John, (friend(John, Mary) \rightarrow exists(Mary, loves(Mary, Mary))).$$

In contrast, the use of a quantifier representation for the same proposition yields the formula $\forall x.bel(John, (friend(John, x) \rightarrow \exists y.loves(x, y)))$. Here y is bound by the existential quantifier, and cannot be instantiated. Thus, unintended instantiations like the above are not possible.

In the present paper, we discuss Ambivalent Logic AL as a general framework for first order logic with a syntax in which all of the above mentioned ambivalent phenomena occur. AL has a fully ambivalent syntax, in which the usual distinctions between the syntactic categories of terms, functions, predicates, and formulas, cannot be made. While the part of Unique Reading that says that a well-formed string in the language is either a formula or a term, but not both, does not hold for AL, these syntactic distinctions do retain their usual contextual meaning.

AL has a standard (first order predicate logic) derivational calculus, and a (first order) closed term semantics. We develop some basic proof theory for AL, including soundness and completeness of the derivational calculus w.r.t. the semantics, an s-equivalence theorem, and Herbrand's theorem. We discuss unification for AL. In particular, we show how the Martelli-Montanari unification algorithm can be adapted for AL. We prove that the appropriate AL version has the usual properties. In combination with various other results discussed in this paper, this shows that resolution is a complete and sound inference method for AL. In addition, we show that AL is a conservative extension of FOL. We also show how various known ambivalent syntaxes (such as Prolog's syntax and the syntax discussed in Kalsbeek [Kal]) can be obtained as special instances of

the full ambivalent syntax of AL. The proof theoretic results we obtain for AL also hold for the various specialisations of AL. These results justify the use of AL and various of its subsystems as a basis for (meta-) logic programming. In addition, we argue that AL provides an interesting framework for formalisations of knowledge and belief.

AL was first introduced in Jiang [Jia94a] and [Jia94b]. Some of the results we present are improvements of results announced in Jiang [Jia94a]. A proper subsystem of AL, appropriate for amalgamated extensions of the Vanilaa meta-interpreter, was introduced in Kalsbeek [Kal].

There are various examples of logics which, like AL, have a syntax incorporating ambivalent phenomena. Recently, Hilog was proposed by Chen et al. [CKW93] as a basis for higher order logic programming. Both AL and Hilog combine a second order syntax with a first order semantics. While the syntax of AL extends Hilog syntax, Hilog is, in contrast to AL, not an extension of FOL. Another example is the logic proposed by Richards [Ric74], which is intended for formalisations of intensional logics. For a comparison between AL and Richards logic, we refer to Jiang [Jia94a]. The reader is also referred to Gabbay [Gab92], where other flexible meta-languages are proposed.

The outline of this paper is as follows. In Section 2, we introduce the fully ambivalent syntax of AL and we show how specialisations of it may be obtained. In Section 3 we develop a closed term semantics for AL. In Section 4 we discuss how equality can be incorporated in AL. Section 5 is devoted to various basic proof theoretic results for AL. In Section 6 we discuss the appropriate version of the Martelli-Montanari unification algorithm for AL. In Section 7 we discuss the relations between Ambivalent Logic and Hilog.

2 Syntax for Ambivalent Logic

We define in Section 2.1, a syntax which is fully ambivalent, that is, in which every well formed expression can act as a formula, as a term, as a function, and as a predicate. Whether an expression is evaluated as a term, a formula, a function, or a predicate will be determined by the context. We allow for free arity of predicates and functions. This syntax generates a multi-purpose language.

The full ambivalent syntax extends the syntax for the Vanilla meta-interpreter (cf. Kalsbeek [Kal]), in which can occur in term positions but not vice versa, and in which predicates and functions are always symbols with a fixed arity. It also extends Prolog's syntax (cf. Apt and Teusink [AT]), which shares with full ambivalent syntax the full atoms-terms ambivalence, and the free arity of functions and predicates, but in which predicates and functions are always parameters.

To obtain versions of ambivalent syntax which generate languages that are adapted to a particular purposes such as the above, the definitions for full

ambivalent syntax can be adapted and specialised. In Section 2.2 we discuss several of these refinements.

2.1 Full ambivalent syntax

A fully ambivalent language $L_{\mathcal{G}}$ is generated by a set of non-logical constants (parameters) $\mathcal{G}$, an infinite set of variables $x, y, z, \ldots$, and the usual logical connectives and quantifiers of first-order logic.

Definition 2.1 (Expressions)

1. The variables $x, y, z, \ldots$ and individual constants $a, b, c, \ldots \in \mathcal{G}$ are expressions.
2. If t, $t_1, \ldots, t_n$ are expressions, then $(t)(t_1, \ldots, t_n)$ is an expression.
3. If A and B are expressions, then so are $\neg A$, $A \wedge B$, $A \vee B$, $A \leftarrow B$, $A \rightarrow B$ and $A \leftrightarrow B$.
4. If A is an expression and x is a variable, then $\forall x(A)$ and $\exists x(A)$ are expressions. □

The above ambivalent syntax thus extends standard syntax for first order predicate logic in several ways. The usual requirement is dropped which states that the sets of predicate symbols and function symbols are disjoint. In addition, the usual requirement is dropped that predicates and functions have a fixed arity. As an example, $p(p(p,p))$ is a well-formed ambivalent expression. In addition, variables may occur in formula positions. As an example, the Prolog clause $solve(x) \leftarrow x$ is a well-formed expression in full ambivalent syntax. Also, 'second order' quantification is allowed. That is, expressions like $\exists x.x(c)$ are well-formed in full ambivalent syntax. (We will see in Section 3 that, semantically, quantification over predicates will not be interpreted as second order quantification.) Moreover, not only parameters, but also more complex expressions are allowed as predicates and functions. An example is the generic closure predicate discussed in the introduction. This feature allows for the formation of new predicates, which can be useful in databases. Other examples are: $p(x) \wedge q(x) \rightarrow (p \wedge q)(x)$ and $\forall x \forall z(p(x,z) \rightarrow (\exists y.p(y))(x))$. The latter example shows that quantified expressions are allowed in predicate places.

In many cases, we will omit brackets if this does not lead to confusion. For example, we will write $c(x)$ instead of $(c)(x)$, and $\forall x p(x)$ (or also $\forall x.p(x)$) instead of $\forall x(p(x))$. In various cases, however, brackets cannot be omitted without altering an expression. For instance, $(p \vee q)(t,s)$ is an atomic expression, while $p \vee q(t,s)$ is a disjuctive expression. Similarly, we distinguish between the expressions $p((x)(a))$ and $(p(x))(a)$. In the former, the symbol p predicates over the expression (x)(a), while in the latter, $p(x)$ predicates over a.

Definition 2.2 (Atomic expressions)
An *atomic expression* is either a constant, or a variable, or an expression of the form $(t)(t_1, \ldots, t_n)$.
Atomic expressions of the form $(t)(t_1, \ldots, t_n)$ are *functional atoms.* □

The set $FV(t)$ of free variables of an expression t is defined analogous to the definition for standard syntax, except that it is additionally defined for expressions in predicate and function places.

Definition 2.3 (Free Variables) Let x be a variable, $c \in \mathcal{G}$, and let A, B, and $(t)(t_1, \ldots, t_n)$ be expressions. The set of free variables occurring in an expression is defined as follows:

$FV(x) = x$
$FV(c) = \emptyset$
$FV((t)(t_1, \ldots, t_n)) = FV(t) \cup FV(t_1) \cup .. \cup FV(t_n)$
$FV(A \wedge B) = FV(A) \cup FV(B)$
$FV(\neg A) = FV(A)$
$FV(\exists x A) = FV(A) \setminus \{x\}$
Similarly for the other binary connectives and the universal quantifier.

A variable x occurring in an expression t is *bound* in t if $x \notin FV(t)$; otherwise we say x occurs free in t. We write $A\{x/t\}$ to denote the result of replacing every free occurrence of x in A by t. □

Definition 2.4 An expression t is *closed* (or a *sentence*) if $FV(t) = \emptyset$

By Definition 2.3, x occurs free in $(p(x))(a)$. Also, x is not free in $p(\exists x p(x))$ because x is not free in the argument $\exists x p(x)$. In this sense, quantifiers in term-positions behave like ordinary quantifiers.

The above definition can be refined in a standard way to distinguish between various occurrences. For example, in $p(x) \vee \exists x f(x)$, the first occurrence of x is free, while the second occurrence is bound by the existential quantifier. The scope of quantifiers can be defined in the usual way. For example, in $\exists x q(p(x) \wedge \exists x f(x))$, the first occurrence of x is bound by the outermost existential quantifier, while the second occurrence is bound by the rightmost quantifier.

In Section 3 we will need the following standard notion.

Definition 2.5 *An expression t is* free for *a variable x in an expression A iff t does not contain any free variable that is bound by some quantifier in A when every free occurrence of x in A is replaced by t.* □

Due to the nature of full ambivalent syntax, the role of an expression can be determined only by the context of the expression. For example, consider

the expression $(q(d))(a, q(a)) \wedge q(c)$. It can be evaluated both as a term and as formula, depending on the context. In the latter case, q(d) serves as a predicate, q(a) as a term, and q(c) as a formula.

Clearly, the Unique Reading Lemma does not hold for ambivalent syntax, as every AL expression can be both a formula and a term. We can, however, define in some cases which role a subexpression assumes in the context of an expression in which it appears. In some cases, it is trivial to determine the contextual role of subexpressions. For example, $a \vee b$ occurs as a term in $a(a \vee b)$, independent of whether the latter is evaluated as a term or as a formula. But we have the choice whether or not to consider a as a term in $c(a \vee b)$. The choice we make is inspired by the semantics we define in the next section. In this semantics, there will be no connection between the interpretation of the 'term' a and the interpretation of the 'term' $a \vee b$. Therefore, we will not consider a as a subterm in $c(a \vee b)$. Similar considerations lead to the following definitions of the notions of subformula, term, function, and predicate. The definition of the notion of subformula is standard.

Definition 2.6 (Subformula)

1. Every expression is a subformula of itself.
2. Let A be an expression. Then every subformula of A is a subformula of $\neg A$, $\exists x A$, and $\forall x A$.
3. Let A and B be expressions. If E is a subformula of A or B, then E is a subformula of $A \vee B$, $A \wedge B$, $A \rightarrow B$, $A \leftarrow B$, and $A \leftrightarrow B$. □

By this definition, a does not occur as a subformula in $c(a \vee b)$.

Definition 2.7 (Occurrence as a term)

1. In an expression of the form $(t)(t_1, \ldots, t_n)$, $t_1, \ldots, t_n$ occur as terms.
2. If t occurs as a term in A and A is a subformula of F, then t occurs as a term in F.
3. If t occurs as a term in s and s occurs as a term in F, then t occurs as a term in F. □

By the above definition, a does not occur as a term in $c(a \vee b)$, while $a \vee b$ does. Also, c does not occur as a term in $c(a \vee b)$.

Definition 2.8 (Occurrence as a function)

1. t occurs as a function in an expression $(t)(t_1, \ldots, t_n)$.
2. If t occurs as a function in s and s occurs as a term in F, then t occurs as a function in F. □

Definition 2.9 (Occurrence as a predicate)

1. t occurs as a predicate in an expression $(t)(t_1, \ldots, t_n)$.
2. If t occurs as a predicate in A and A is a subformula of F, then t occurs as a predicate in F. □

By the above definitions, t occurs both as a predicate and as a function in $(t)(t_1, \ldots, t_n)$. In $\forall x(t(x))$, t occurs as a predicate, but not as a function. In $p(\forall x(t(x)))$, t occurs neither as a predicate nor as a function.

All of the above definitions can be refined to distinguish between the various occurrences. For example, in $a(a) \vee a$, the first occurrence of a is as a predicate (but not as a function), the second occurrence is as a term, while the third occurrence is as a subformula.

It is useful to make the following distinction between two kinds of occurrences of quantifiers. The first kind of quantifier, which we will call 'outside quantifier', occurs in places where they are also allowed in FOL formulas. The second kind occurs only in 'ambivalent' places.

Definition 2.10 Let $Qx.s$, where Q is a quantifier $\forall$ or $\exists$, be an occurrence of a subexpression of an AL expression t. Q is an *outside quantifier* if it is an occurrence as a subformula of t. Otherwise, Q is an *inside quantifier*.
Inside and outside connectives are defined similarly. □

For example, in $\forall x.(x \vee \exists y.f(y))$, the first quantifier is an outside quantifier, while the second is an inside quantifier. Observe that both inside and outside quantifiers do bind variables in their scope. In Section 3, where we discuss semantics for AL, we will see that the outside quantifiers will be interpreted as real quantifiers, ranging over elements of the domain; this in contrast to inside quantifiers.

2.2 Refinements

For many purposes, the full ambivalent syntax defined in the previous subsection admits too many expressions. Refined versions of ambivalent syntax, well-tuned to particular domains of application, can be obtained by specialising one or more of the clauses in Definition 2.1, and by the use of sets of special constants in addition to the set of generating constants. In defining special versions of full ambivalent syntax, it is sometimes useful to introduce new syntactic categories.

We will give some examples of specialisation and the use of special constant sets.

- A version of ambivalent syntax which does not admit all its expressions to occur as predicates and functions, but only its constants, can be obtained by modifying clause (2) of Definition 2.1 as follows: If $t_1, \ldots, t_n$ are expressions and $c \in \mathcal{G}$, then $c(t_1, \ldots, t_n)$ is an expression. In this particular

version, variables and more complex expressions do not occur as predicates and functions, and as a result there is no 'second order' quantification.

- It may be useful to select special roles for some of the parameters. As an example, in some domains, it may be useful to have one or more symbols that occur only as predicates. The negation predicate in Prolog is an example of such a special predicate (see below). In that case, a set of special symbols $\mathcal{S}$ is added to the signature of the language, and an extra syntactic category is introduced: special expressions. The following modification of Definition 2.1 is used:

  1 Variables x and parameters $c \in \mathcal{G}$ are expressions.

  2 If $t, t_1, \ldots, t_n$ are expressions, then $(t)(t_1, \ldots, t_n)$ is an expression.

  2' If $p \in \mathcal{S}$, and $t_1, \ldots, t_n$ are expressions, then $p(t_1, \ldots, t_n)$ is a special expression.

  3' If A is an expression, then $\neg A$ is an expression; If A is a special expression, then $\neg A$ is a special expression.
  $A \vee B$ is a special expression if both A and B are special expressions or if one among them is a special expression and the other is an expression. $A \vee B$ is an expression if both A and B are expressions. Similarly for the other binary connectives.

  4' $\forall x(A)$ is an expression if A is an expression; it is a special expression if A is a special expression.

  In particular, special expressions do not occur as terms, predicates, or functions, in expressions and special expressions. Special parameters only occur as predicates in special expressions.

  Special definitional clauses can also introduce parameter symbols which only occur with fixed arities.

- The syntax of Prolog is a specialisation of full ambivalent syntax, and shares some of the properties of the above specialisations. In Prolog's syntax, only parameters are allowed in predicate and function positions. In addition, Prolog has a special predicate *not*, which is only allowed to occur in predicate positions. A representative part of Prolog's syntax can be described as follows:

  1 variables x and parameters $c \in G$ are expressions.

  2 $c(t_1, \ldots, t_n)$ is an expression if $c \in G$ and $t_1, \ldots, t_n$ are expressions.

  3 $not(t)$ is a special expression if t is an expression.

  4 $fail$ and ! are special expressions.

  5 every expression is a special expression.

6 $t \leftarrow t_1, \ldots, t_n$ is a Prolog clause if t is an expression and $t_1, \ldots, t_n$ are expressions.

7 every expression that is not a variable is a Prolog clause.

Other features of Prolog's syntax can be incorporated in this framework, such as the Prolog built in predicates *assert* and *retract*, which take Prolog clauses as arguments.

- The ambivalent syntax described in Kalsbeek [Kal] is a specialised version of full ambivalent syntax. It differs from full ambivalent syntax in the following ways:
  1. Only atomic expressions are allowed to occur in term positions.
  2. Only parameters are allowed to occur as functions and predicates.

  In particular, variables are not allowed to occur in formula positions. A full description can be found in Kalsbeek [Kal].

3 Semantics

We have chosen to develop a closed term semantics for AL for the purpose of this paper. All of the definitions we give can be adapted to special versions of AL such as Prolog's syntax.

Formally, a structure $\mathcal{M}$ for an ambivalent language $L_{\mathcal{G}}$ (the *underlying language* of $\mathcal{M}$) is a tuple (D, T), where

1. D, the domain of $\mathcal{M}$, is the set of *closed* expressions of $L_{\mathcal{G}}$;
2. T, the truth set of $\mathcal{M}$, is a subset of the set of *closed atomic* expressions of $L_{\mathcal{G}}$.

The satisfiability relation for closed expressions in a structure $\mathcal{M} = (D, T)$ is inductively defined as follows:

Definition 3.1

$\mathcal{M} \models_{AL} A$ iff $A \in T$ where A is a closed atomic expression

$\mathcal{M} \models_{AL} A \wedge B$ iff $\mathcal{M} \models_{AL} A$ and $\mathcal{M} \models_{AL} B$

$\mathcal{M} \models_{AL} \neg A$ iff $\mathcal{M} \not\models_{AL}$ A

$\mathcal{M} \models_{AL} \forall x A$ iff for all $d \in D$, $\mathcal{M} \models_{AL}$ A{x/d}

Disjunction, implications, equivalence and existential quantification are defined in the standard way. □

Several things are worth noting at this stage.

- Closed expressions can be evaluated as sentences in a model. At the same time, they constitute elements of the domain.

- A closed expression, evaluated as a sentence, in a model, has a unique truth value. Thus, while there is syntactic ambivalence, there is no semantic ambiguity.

- The truth values of atoms are, in principle, not related to the structure of expressions which occur in them as subterms. That is, for example, the truth value of an atom $p(s \wedge v)$ is in principle not related to the respective truth values of $p(s)$ and $p(t)$. A similar distinction holds for the relation between outside and inside quantifiers: The truth value of $\forall x.p(f(x))$ is independent of the truth value of $p(\forall x.f(x))$. The latter is decided by a checking whether the atom $p(\forall x.f(x))$ belongs to the truth set, while the former is decided by checking whether $p(f(d))$ belongs to the truth set, for each element d in the domain. In certain applications, such as formalisations of knowledge and belief, it might be desirable to have explicit relations between inside and outside quantifiers, or between outside and inside connectives such as the above. The soundness and strong completeness theorem 5.1 that we will prove in Section 5 shows that such relations may be implemented by axioms.

- It should be noted that 'similar' expressions like, for example, $\forall x.f(x)$ and $\forall y.f(y)$ constitute different, and unrelated, objects in the domains of models. That is, the truth values of the closed expressions $t(\forall x.f(x))$ and $t(\forall y.f(y))$ need not be the same. This may be counterintuitive, and even undesirable in some applications. Extra assumptions on the truth sets T may impose that expressions which are similar under appropriate renaming of the bound variables, behave similarly as elements of the domain. This in its turn should then be matched with an appropriate rule in the derivational calculus. In contrast, for outside quantifiers the following relation, familiar from FOL, holds: $\mathcal{M} \models \forall x A$ iff $\mathcal{M} \models \forall y A\{x/y\}$, if y is free for x in A.

- It should be observed that in the above interpretation of the quantifiers (the substitution interpretation), there is no real semantic second order quantification: the quantifiers range over object in domains, and not over subsets of domains. Thus, while syntactically AL allows for second order features such as quantification over functions and predicates, its closed term semantics is first order.

We will use the following standard notions.

Definition 3.2 Let A be a closed expression and let S be a set of AL sentences. S is *satisfiable* in AL iff there exists an interpretation $\mathcal{M}$ such that $\mathcal{M} \models_{AL} \phi$ for all $\phi \in S$. $\mathcal{M}$ is called a *model* of S in this case.

A is *valid* in AL, denoted by $\models_{AL} A$, iff $\neg A$ is not satisfiable in AL.
We say that A is a *logical consequence* of S, denoted by $S \models_{AL} A$, iff A is true in every model of S. □

Definition 3.3 A structure $\mathcal{M}$ is a *Herbrand model* for a theory S ($\mathcal{M} \models^h_{AL} S$) if $\mathcal{M}$ is a structure for S and the underlying language of $\mathcal{M}$ is the underlying language of S (that is, if D coincides with the set of closed expressions of the full ambivalent syntax generated by S). □

In addition, we can define semantics for all expressions, using assignments (Definition 3.5). We need the following standard definitions.

Definition 3.4 Let σ be a substitution $\{x_1/t_1, \ldots, x_n/t_n\}$. Then its *domain* $dom(\sigma)$ and *range* $ran(\sigma)$ are defined as follows:
$dom(\sigma) = \{x_1, \ldots, x_n\}$,
$ran(\sigma) = \{t_1, \ldots, t_n\}$
The σ-image of a variable x, denoted as $x\sigma$, is given as follows:
$x_i\sigma = t_i$, for $i \in [1, n]$, and
$x\sigma = x$, for $x \notin dom(\sigma)$.
For expressions t and substitutions σ, $t\sigma$ is the result of replacing each free occurrence of x in t by the σ-image of x. □

Definition 3.5 A substitution σ is an *assignment* for an expression s and an interpretation $\mathcal{M} = \langle D, T \rangle$, if $dom(\sigma) \supseteq FV(s)$ and $ran(\sigma) \subseteq D$. □

Definition 3.6 Let σ be an assignment for t and $\mathcal{M}$. We say that $\mathcal{M}$ *satisfies* t *with* σ ($\mathcal{M}, \sigma \models t$) if $\mathcal{M} \models t\sigma$. □

As a *derivational calculus* for Ambivalent Logic we take some standard system of natural deduction. The resulting derivability relation will, as usual, be indicated by $\vdash$. The deduction rules are defined in a standard way. Consider the usual elimination rule for the universal quantifier, $\forall x.t/t\{x/s\}$. It has a side condition, which prevents unintended bindings resulting from the substitution of s for x. In the case of AL, unintended bindings can result, not only from quantifiers in standard places, but also from inside quantifiers. As an example, $\forall x p(\exists y f(x, y))/p(\exists y f(y, y))$ is not a correct instance of the rule for elimination of the universal quantifier: y is not free for x in $p(\exists y f(x, y))$.

Example 3.7 The ambivalent expression $p(a)$ can, among others, be considered as either $p(x)\{x/a\}$ or $x(a)\{x/p\}$. The former corresponds to the following (standard) instance of the introduction rule for the existential quantifier: $p(a)/\exists x(p(x))$. The latter in its turn corresponds to $p(a)/\exists x(x(a))$.

We remind the reader of the following two notions of completeness.

Definition 3.8 Let L be a logic with semantic consequence relation $\models_L$ and derivational consequence relation $\vdash_L$.
L is *weakly complete* if, for any sentence B, $\models_L B$ implies $\vdash_L B$.
L *strongly complete* if, for any sentence B and any set of sentences S, $S \models_L B$ implies $S \vdash_L B$. □

In the case of first order logic with standard syntax, a restriction to Herbrand semantics results in the weakening of some of the standard results. A well-known example is the loss of strong completeness in its full sense. While strong completeness w.r.t. Herbrand models holds for finite and infinitely extendible theories, it does not hold for theories in which every closed term of the underlying language occurs in the theory. The same phenomenon occurs in the context of Ambivalent Logic, as witnessed by the following proposition.

Proposition 3.9 *AL is not strongly complete for infinite theories with respect to the Herbrand semantics.*

Proof: Let $t_1, t_2, t_3, \ldots$ be an enumeration of all the closed expressions of an ambivalent language L. Then, by the definition of validity in Herbrand models, $t_1, t_2, t_3, \ldots \models^h_{AL} \forall x.x$. However, $t_1, t_2, t_3, \ldots \not\vdash \forall x.x$ □

While a restriction to Herbrand semantics is sensible in the case of logic programming (logic programs being finite theories), in a broader context it is, in view of the above proposition, sensible to consider the more general closed term semantics as the appropriate semantics for Ambivalent Logic.

A more general semantics for AL, with arbitrary domains and, consequently, non-identity interpretation functions, will not be considered in the context of the present paper. The reason is that, (as remarked in Chen et al. [CKW93]) interpretation of quantified terms using a non-identity interpretation function requires something like lambda-abstraction, which considerably complicates the construction of models. In the next section we argue that the restriction to closed term semantics is not a severe limitation.

4 Equality and Identity

The usual semantics for FOL with standard syntax is different from the closed term semantics we described above. In particular, in the semantics for FOL, any set can in principle serve as the domain of a model, and the interpretation functions, mapping closed terms of the language on elements of the domain, are not necessarily identity functions. In this semantics, equality is usually interpreted as identity on the domains, although the equality axioms in themselves do not force this interpretation. In contrast, closed term models are not appropriate for the identity interpretation of equality.

Closed term models, however, can be used to represent other models. Hence they play an important, albeit hidden, role in FOL. Consider the usual proof of

the completeness theorem for FOL, using the Henkin method. The proof in fact consists of two separate stages, each the proof of an independent lemma. In the first stage, a consistent theory T is extended to a maximally consistent Henkin theory T^*, for which a Herbrand model H is constructed. H is a closed term model for the theory T, and, in presence of equality in the language, equality is interpreted in H as an equivalence relation which is also a congruence with respect to the predicates and functions. The second stage of the proof is merely motivated by the convention to interpret equality as identity. From H, a model M for T^* is constructed, the domain of which consists of the equivalence classes in H under equality. The valuation function of M is 'inherited' from H. This second stage can be interpreted as the proof of a representation lemma, which expresses that for every closed term model H there exists an elementary equivalent model M in which equality is interpreted as identity. This representation lemma can be reversed. Let $M = \langle D_M, V_M \rangle$ be a model. A closed term model H is now constructed as follows. Define a function f from the closed terms of the language to be interpreted to the domain of M, which is defined by $\underline{f(t)} := t^M$. Now define the valuation function V_H as follows: $V_H(P(\bar{t})) := V_M(P(\underline{f(t)}))$. Now H and M are elementary equivalent, and equality is interpreted as just another binary predicate in H, satisfying properties dictated by the equality axioms.

The two sides of this representation result together in fact show that there is a bijection between the class of closed term models and the class of general models (modulo isomorphism). It also shows that there is no inherent need to interpret identity as equality. The interpretation of equality as identity forces the choice of the usual semantics of FOL — if this restriction is abandoned, closed term models are a sound and complete semantics for first order logic with equality, provided the truth sets satisfy the properties dictated by the equality axioms. What we have argued here are several things. First, the usual axioms for equality do in themselves not force the identity interpretation. Second, if the usual identity interpretation of equality is abandoned, we can restrict ourselves to consideration of closed term models without loss of generality.

In particular, we can incorporate equality in Ambivalent Logic without changing the style of the semantics for Ambivalent Logic. In the closed term semantics, equality will not correspond to real identity on the domains.

Let us consider, in some more detail, the usual equality theory ET for FOL. It can be axiomatised (abstracting from arities) as follows:

(I) $\forall x.x = x$
$\forall x \forall y(x = y \rightarrow y = x)$
$\forall x \forall y \forall z(x = y \rightarrow (y = z \rightarrow x = z))$
(II) $\forall x \forall y \forall t(x = y \rightarrow t = t\{x/y\})$
(III) $\forall x \forall y \forall t(x = y \rightarrow (t \rightarrow t\{x/y\}))$

We obtain an appropriate closed term semantics for AL extended with these equality axioms, by additionally restricting the truth sets to satisfy the cor-

responding properties. In particular, ET(I) requires truth sets on which $=$ is an equivalence relation. ET(I,II) requires truth sets T (say, with underlying language $L_{\mathcal{G}}$) which additionally satisfy the following: if $d = c \in T$, and t is an $L_{\mathcal{G}}$-expression with only x free, then $t\{x/d\} = t\{x/c\} \in T$. ET as a whole requires truth sets T which in addition also satisfies the following property: for all closed $L_{\mathcal{G}}$-expressions d and c, and for all $L_{\mathcal{G}}$-expressions t with only x free, if $d = c \in T$, then $t\{x/d\} \in T$ iff $t\{x/c\} \in T$.

Observe that, in the context of the closed term semantics style, we are not committed to ET as a whole. In particular, there are interesting differences between AL + ET(I,II) and AL + ET, which we will more closely consider in the next section.

In the sequel, when we consider AL, we explicitly mean AL without equality. If we consider either of AL + ET(I), AL + ET(II), and AL + ET, we will implicitly assume that the semantics satisfies the corresponding conditions mentioned above.

5 Formal results

In the present section, we set out to develop some basic theory for Ambivalent Logic. First, we prove soundness and completeness of AL w.r.t. the closed term semantics.

Theorem 5.1
AL is sound and strongly complete with respect to the closed term semantics; AL is strongly complete for infinitely extendible theories with respect to Herbrand semantics.

Proof: Soundness is left to the reader. The completeness proof follows the standard completeness proof for FOL.

Let S be a consistent AL theory with underlying language $L = L_{\mathcal{G}}$. First extend S to a Henkin theory S^*, as follows.
Extend the set of generating constants $\mathcal{G}$ with a new constant d, and let $L' = L_{\mathcal{G}\cup\{d\}}$.

Let $\phi_1, \phi_2, \phi_3, \ldots$ be an enumeration of all the expressions of L' with only x free. Let $d_1, d_2, d_3, \ldots$ be an enumeration of all the closed expressions of L' that are not expressions of L. A carefully chosen subset of these expressions will serve as Henkin constants. Define C_n as the set of closed expressions that occur as terms in $\phi_1 \wedge \ldots \wedge \phi_n$.

Define

$$
\begin{array}{lcl}
S_0 & := & S, \\
m_1 & := & min\{k | d_k \notin C_1\}, \\
S_1 & := & S_0 \cup \{\exists x.\phi_1(x) \rightarrow \phi_1(d_{m_1}), \\
m_{n+1} & := & min\{k > m_n | d_k \notin C_{n+1}\}, \\
S_{n+1} & := & S_n \cup \{\exists x.\phi_{n+1}(x) \rightarrow \phi_{n+1}(d_{m_{n+1}}), \\
S^* & := & \cup S_n.
\end{array}
$$

S^* is a Henkin theory. Conservativity of S_{n+1} over S_n can be proven in the usual way. Therefore, S^* is conservative over S and thus consistent.

Next, extend S^* to a maximally consistent theory S^m, by the Lindenbaum lemma. The language of S^m is L', and S^m is a Henkin theory. A model $\mathcal{M} = \langle D, T \rangle$ for S^m is obtained as follows. Let D consist of all the closed expressions of L'. Let T consist of the closed atomic expressions that belong to S^m. By induction on the complexity of sentences, $\mathcal{M}$ is a Herbrand model for S^m. By conservativity of S^m over S, $\mathcal{M}$ is a closed term model for S.

For infinitely extendible theories a model can be constructed in a slightly more elegant way. The closed expressions of L that do not occur as terms in the theory can be used as the Henkin constants. The resulting model is a Herbrand model for the theory. □

We leave it to the reader to check that the above result also goes through for AL + ET(I), AL + ET(I,II), and AL + ET.

Proposition 5.2 *Let S be a finite set of AL expressions, in which at least one parameter occurs. Then S is infinitely extendible.*

Proof: Let g be a parameter occurring in S. Then $g, g(g), g(g(g)), \ldots$ is an infinite set of closed expressions in the language underlying S. In contrast, the number of closed expressions of this language that occur as terms in expressions of S, is finite, as S is a finite set. □

An immediate consequence of the above theorem and proposition is the following.

Corollary 5.3 *Every satisfiable AL sentence has a Herbrand model.*

The following notion of normal form is the appropriate version for AL.

Definition 5.4 An expression F is in *normal form* if $F \equiv Q_1 x_1, \ldots, Q_n x_n . t$, where

1. The Q_i are quantifiers $\forall$ or $\exists$.
2. t is built up from atomic expressions and the connectives $\neg$, $\vee$, and $\wedge$; in particular, in any subexpression of t of the form $Qy.s$, where Q is a quantifier, Q is an inside quantifier.
3. The variables x_i are mutually distinct.
4. If $Qy.s$ is a subexpression of t, where Q is an (inside) quantifier, then y is distinct from any of the x_i. □

Lemma 5.5 *For every formula F there is an equivalent formula F', such that F' is in normal form.*

Proof: By the usual methods. □

The *Skolem form* F^s of a normal form expression F can be obtained by the usual methods (cf., for instance, Schöning [Sch89] for an algorithm to obtain Skolem forms). Observe that Skolemisation does leave the inside quantifiers and connectives intact.

Next we prove an AL version of the usual s-equivalence theorem. The proof differs from the usual proof in a crucial aspect. In the usual proof, essential use is made of the option to use non-trivial interpretation functions on domains. In particular, a model $\mathcal{M}$ for a normal form formula F is extended to a model for its Skolem form by extending the signature of $\mathcal{M}$ with functions on its domain that interpret witnesses of the existential quantifiers. In the case of Ambivalent Logic, this construction cannot be used, as we do not allow for non-trivial interpretation functions. For clarity of this exposition, assume that $\mathcal{M}$ is a Herbrand model for F. The introduction of the Skolem functions then results in a proper extension of the domain of $\mathcal{M}$. As a result, instead of obtaining a model $\mathcal{M}'$ for the Skolem form by a proper extension the interpretation valuation functions of $\mathcal{M}$, a new model has to be constructed. The upshot is that this new model $\mathcal{M}'$ is a Herbrand model for the Skolem form F^s, in this sense the result below is stronger than the corresponding usual result for FOL. However, the construction of $\mathcal{M}'$ is a bit more involved in the ambivalent case. The truth set T' of $\mathcal{M}'$ is obtained by projecting the expressions of the extended language on the expressions of the original language, and by the taking T' as the pre-image of T under this projection. The projection (translation) is the identity function on the original language.

The proof we give here is given for the case of AL, but can be generalised to FOL and intermediate cases.

Theorem 5.6 (s-equivalence) *For every closed expression F in normal form, F is satisfiable iff its Skolem form is satisfiable.*

Proof: Let $F = \forall x_1 \exists y_1 \ldots \forall x_n \exists y_n . \phi$, where all outside quantifiers are indicated. Let $L_{\mathcal{G}}$ be the underlying language of F.
Let $f_1, \ldots, f_n$ be new parameters not occurring in ϕ.
Let $L' = L_{\mathcal{G} \cup \{f_1, \ldots, f_n\}}$.
Let $F^s = \forall x_1 \ldots \forall x_n . \phi\{y_1/f_1(x_1), \ldots, y_n/f_n(x_1, \ldots, x_n)\}$.
Let $\phi' = \phi\{y_1/f_1(x_1), \ldots, y_n/f_n(x_1, \ldots, x_n)\}$, that is, $F^s = \forall x_1 \ldots \forall x_n . \phi'$.

The proof of the left-to-right direction is as usual. For the converse direction, let, (by Corollary 5.3), $\mathcal{M} = \langle D, T \rangle$ be a Herbrand model for F; here, D consists of the closed expressions of $L_{\mathcal{G}}$. We can now, as usual, by the axiom of choice, define (external) functions $v_1, \ldots, v_n$ on D, such that for all $d_1, \ldots, d_n \in D$

$$\mathcal{M}, \{x_1/d_1, y_1/v_1(d_1), \ldots, x_n/d_n, y_n/v_n(d_1, \ldots, d_n)\} \models \phi. \qquad \text{(I)}$$

We construct, using $\mathcal{M}$ and the functions $v_1, \ldots, v_n$, a Herbrand model $\mathcal{M}' = \langle D', T' \rangle$, where D' consists of all the closed expressions in L', such that $\mathcal{M}' \models$

F^s. Observe that D is a strict subset of D'.

We will define the truth-set T' using the following translation (or projection) $(\cdot)^*$ of expressions of L' into expressions of $L_{\mathcal{G}}$.

- For all $c \in \mathcal{G}$, $c^* = c$
- For all variables x, let $x^* = x$
- For f_i, choose a $c_i \in \mathcal{G}$, and let $f_i^* = c_i$
- If $t = f_i$ and $n = i$, let $(t)(t_1, \ldots, t_n)^* = v_i(t_1^*, \ldots, t_i^*)$;
- Otherwise, $(t)(t_1, \ldots, t_n)^* = (t^*)(t_1^*, \ldots, t_i^*)$.
- $(A \vee B)^* = A^* \vee B^*$ and similarly for other binary connectives
- $(\neg A)^* = \neg A^*$
- $(\forall x(A))^* = \forall x(A^*)$
- $(\exists x(A))^* = \exists x(A^*)$.

Now for all expressions t in $L_{\mathcal{G}}$, $t^* = t$. In particular, $(\phi)^* = \phi$. Also, let t be an expression in $L_{\mathcal{G}}$, and let $\sigma = \{z_1/s_1, \ldots, z_k/s_k\}$ be a substitution such that the s_i are L'-expressions. We leave it to the reader to check that $(t\sigma)^* = t^*\{z_1/s_1^*, \ldots, z_k/s_k^*\} = t\{z_1/s_1^*, \ldots, z_k/s_k^*\}$. Therefore, the following equation (II) holds:

$$\begin{aligned}(\phi'\{x_1/d_1), \ldots, x_n/d_n\})^* \quad &= \\ &= (\phi\{x_1/d_1, y_1/f_1(d_1), \ldots, x_n/d_n, y_n/f_n(d_1, \ldots, d_n)\})^* \\ &= \phi^*\{x_1/d_1^*, y_1/(f_1(d_1))^*, \ldots, x_n/d_n^*, y_n/(f_n(d_1, \ldots, d_n))^*\} \qquad \text{(II)} \\ &= \phi\{x_1/d_1^*, y_1/v_1(d_1^*), \ldots, x_n/d_n^*, y_n/v_n(d_1^*, \ldots, d_n^*)\}.\end{aligned}$$

Now define the truth set T' as follows:

$$T' = \{t \in L' : t^* \in T\}$$

By induction it follows that for all sentences A in $L_{\mathcal{G}}$,

$$\mathcal{M}' \models A \quad \text{iff} \quad \mathcal{M} \models A^*.$$

In particular, by (I) and (II), $\mathcal{M}' \models F^s$. □

Another folklore result is Herbrand's theorem:

Theorem 5.7 *A closed formula in Sklolem form with matrix F is unsatisfiable iff there is a finite subset of the Skolem expansion of F which is unsatisfiable.*

The usual proof of the above theorem goes through, without modification, for Ambivalent Logic.

All of the above results (5.2 to 5.7) also hold for extensions of AL with any of the above-mentioned equality theories.

We have proven some of the essential ingredients to prove soundness and completeness of resolution for Ambivalent Logic: the construction of Skolem forms, the s-equivalence theorem and the Herbrand theorem. The missing ingredients for the completeness of resolution are the lifting lemma (which allows a transformation of resolution refutation on clauses in propositional logic to a resolution refutation on clauses in predicate logic), and decidability of unification for Ambivalent Logic. The traditional proof of the lifting lemma goes through, without modification, for Ambivalent Logic. Unification theory for Ambivalent Logic will be discussed in Section 6. The traditional proof of soundness of resolution for FOL (see for instance Schöning [Sch89]) goes through, without modification, for Ambivalent Logic. We conclude that, modulo the results on unification in the next section, we have the following result:

Theorem 5.8 *Resolution is a sound and complete inference method for AL.*

We conclude this section by proving one more result. Given the fact that AL is a syntactic extension of FOL, and has the same derivational calculus, the question of conservativity of AL over FOL arises, that is: if a formula ϕ in standard syntax is derivable in AL, is it then also derivable in FOL — and vice versa? The answer, as the following theorem shows, is affirmative. This result shows that AL is an extension of FOL. The proof uses both directions of the above s-equivalence theorem 5.6. The technique used in the proof is similar to that used in the proof of Theorem 5.6: from a model, a new, ambivalent model is defined, the truth-set of which is a pre-image of a syntactic projection to the truth-set of the original model. In this case, a Herbrand model for a language with standard syntax, is extended to an elementary equivalent Herbrand model for the associated ambivalent language.

Theorem 5.9 *Let ϕ be a formula in standard syntax. Then $\models_{FOL} \phi$ iff $\models_{AL} \phi$.*

Proof: The left-to-right direction follows from the fact that every derivation in FOL is also a derivation in AL, combined with completeness of FOL, and soundness of AL. For the converse direction (conservativity of AL over FOL), it suffices, by contraposition, to show that if ϕ is satisfiable in FOL, then it is also satisfiable as an AL formula.

So let ϕ be satisfiable in FOL. Without loss of generality we can assume that ϕ is in normal form. By the s-equivalence theorem for FOL, the Skolem form ϕ^s is also satisfiable. In particular, there is a Herbrand model $\mathcal{M} = \langle D, T \rangle$ satisfying ϕ^s. Using $\mathcal{M}$, we will construct an ambivalent Herbrand model $\mathcal{M}' = \langle D', T' \rangle$ satisfying ϕ^s.

Let C, F, and P be the set of constants, respectively functions, respectively relation symbols occurring in ϕ^s. Then the universe D of $\mathcal{M}$ is generated by $\langle C, F \rangle$. Let D' be the ambivalent universe generated by the parameter set $C \cup F \cup P$. In order to define the truth set T', we will make use of a (total and surjective) function $(\cdot)^*$ from D' to D. (The definition of * is inspired by the abstraction function defined in [Kal]; here, we use, instead of fresh constants, some element of D.)

Choose an arbitrary $d \in D$. Define $^* : D' \to D$ as follows:

1. For $c \in C \cup F \cup P$,
   $c^* := c$ if $c \in C$
   $c^* := d$ otherwise.

2. for closed terms $(t)(t_1, \ldots, t_n)$,
   $((t)(t_1, \ldots, t_n))^* := t(t_1{}^*, \ldots, t_n{}^*)$ if t is an n-ary predicate in P
   $(t)(t_1, \ldots, t_n)^* := d$ otherwise.

3. For all other $t \in D'$, $t^* := d$.

Observe that, for every $t \in D$, $t^* = t$.

Using the function *, the truth set T' of $\mathcal{M}'$ can now be defined as follows:

$$T' = \{t \in D' : t^* \in T\}.$$

Now let $\phi^s = \forall x_1 \ldots \forall x_n . A$, where A is a conjunctive normal form.

$\mathcal{M} \models_{FOL} \forall x_1 \ldots \forall x_n . A$
$\Longrightarrow$ {by definition of the satisfaction relation}
for all $d_1, \ldots, d_n \in D$ $\mathcal{M} \models_{FOL} A\{x_1/d_1, \ldots, x_n/d_n\}$
$\Longrightarrow$ {by definition of * and by the form of A}
for all $d_1, \ldots, d_n \in D'$, $\mathcal{M} \models_{FOL} A\{x_1/d_1{}^*, \ldots, x_n/d_n{}^*\}$
$\Longrightarrow$ {by definition of T'}
for all $d_1, \ldots, d_n \in D'$, $\mathcal{M}' \models_{AL} A\{x_1/d_1, \ldots, x_n/d_n\}$
$\Longrightarrow$ {by definition of the satisfaction relation}
$\mathcal{M}' \models_{AL} \forall x_1 \ldots \forall x_n . A$.

From the above implication and the easy side of the s-equivalence theorem 5.6, it immediately follows that $\mathcal{M}'$ is a closed term model satisfying ϕ. □

The soundness direction of the above theorem generalises to AL + ET(I), AL + ET(I,II), and AL + ET. The completeness side however, trivially does not hold for either of AL + ET(I) and AL + ET(I,II), as FOL incorporates all of ET.

Although AL + EQ(I,II) is not conservative over FOL for formulas with equality, this logic is of some interest in the context of intensional logic. By the soundness and completeness w.r.t. the appropriate semantics, it does not

satisfy ET(III) — therefore, it is opaque w.r.t. substitutions of equal terms. In addition, by the ambivalent syntax of AL, which allows for all expressions to occur in term positions, AL + EQ(I,II) has identity naming, that is, expressions can be represented by themselves. In the domain of knowledge and belief this is a useful property. In contrast, AL + ET is, by the above theorem, a conservative extension of FOL.

6 Unification

In the present section we show how the Martelli-Montanari unification algorithm can be adapted to the case of ambivalent syntax. We show that the adapted algorithm (which is defined on page 23) has all the desired properties, in particular, termination and, in case of successful termination, generation of unifiers which are most general within an appropriate class of unifiers. We follow the outlines of the theory for unification as given in Doets [Doe94], to which we also refer the reader for the usual definitions of unifier and most general unifier.

There are several differences between unification for standard syntax and unification for full ambivalent syntax.

1. In ambivalent syntax, functions and predicates are arityless. Unification of two atoms with different arities has to be excluded. An extra action (9) is sufficient: halt with failure on an equation $(t)(t_1, \ldots, t_n) = s(s_1, \ldots, s_m)$, if $n \neq m$.

2. In the Martelli-Montanari algorithm for standard syntax, there are two actions for functional atoms:
   halting with failure on $f(t_1, \ldots, t_n) = g(s_1, \ldots, s_n)$, if f is unequal to g;
   replacement with $t_1 = s_1, \ldots, t_n = s_n$, otherwise.
   In ambivalent syntax, all expressions can occur in function positions. The above two actions are replaced by one single action (8), accounting for the unification of the expressions in the function positions as well as the arguments. An extra action (10) is needed to prevent unification of functional atoms with expressions that are not functional atoms or variables. In addition, two extra actions (6) and (7) deal with unification of parameters.

3. In ambivalent syntax, conjunctive, disjunctive, implicational, and negated expressions occur as terms and as subexpressions, thus they are candidates for unification. For example, $x \vee y$ and $a \vee c(z)$ are unified by $\{x/a, y/c(z)\}$. This is accounted for in the unification algorithm by the actions (11) and (13). Appropriate failure conditions are reflected by the actions (12) and (14).

4. Likewise, quantified expressions have to be dealt with. For closed quantified expressions, semantical identity coincides with syntactical identity. However, quantified expressions in general are still liable for unification.

First, they can unify with variables. For example, $\exists x(x)$ and y are unified by $\{y/\exists x(x)\}$. Further, quantified expressions can contain free variables, which are candidates for unification. As an example, $\exists x(x(y))$ and $\exists x(x(c))$ are unified by $\{y/c\}$. In contrast, $\exists x(x)$ and $\exists y(y)$ cannot be unified, as both are closed expressions.

Unification of quantified expressions is partly engineered by action (15), which eliminates identical quantifiers: $\exists x.t = \exists x.s$ is repaced by $t = s$, and similarly for the universal quantifier. In contrast, the semantic distinction between $\exists x.t$ and $\exists y.t\{x/y\}$ is reflected in action (16), which halts with failure on equations $\exists x.t = \exists y.s$, if x and y are different (and likewise for the universal quantifier).

However, the (necessary) action (15) might lead to incorrect results, as it releases bound variables, which subsequently become, incorrectly, candidates for non-trivial unification. As an example, action (15) replaces the unsolvable equation $\exists x.x = \exists x.c$ with the solved equation $x = c$. (Below we will formally define the notion of solved equation.) This problem is solved in two ways.

First, before the algorithm is run on a set of equations, all the bound variables should be renamed to marked variants. An appropriate renaming function will be given in Definition 6.1 below. This enables us to keep track of the origin of variables during execution of the algorithm.

Second, the algorithm should treat marked and unmarked variables differently. In particular, marked variables, which should be thought of as bound variables, only unify with themselves and with unmarked variables. Trivial unification of marked and unmarked variables is dealt with in action (1). Non-trivial unification of marked variables with anything but unmarked variables is excluded by action (2). However, unification of unmarked variables with expressions in which marked variables occur free, should be possible. For example, consider the expressions $\exists x(p(x) \vee q(x))$ and $\exists x(y \vee q(x))$, where y is an unmarked variable. These expressions are unified by $\{y/p(x)\}$. That is, unification of unmarked variables with expressions in which unmarked variables occur free, should be allowed. This is taken care of by action (4). The usual actions (3) and (4) only apply if the left-hand variables are unmarked.

The appropriate marking of bound variables is obtained by applying a marking function $(\cdot)^m$, which replaces all occurrences of bound variables with marked copies of these variables. The effect of the marking function is a renaming of the bound variables in an expression, with the effect that no variable occurs both bound and free after marking.

Before we define the marking function, we introduce some notation. Recall that $(t)(t_1, \ldots, t_n)$ is a functional atom with n-ary predicate (or function) t, and

Martelli-Montanari unification algorithm for ambivalent syntax

1	$x = x$	where x is a (marked or unmarked) variable
	remove	
2	$x' = t$	where x' is a marked variable, t is not an unmarked variable, and t is different from x'
	halt with failure	
3	$x = t$	where x is unmarked, t is different from x, and x occurs in t
	halt with failure	
4	$x = t$	where x is unmarked, t is different from x, and x does not occur in t
	replace x by t in all other equations	
5	$t = x$	where t is not an unmarked variable
	replace by $x = t$	
6	$c = c$	where c is a parameter
	remove	
7	$c = t$	where c is a parameter, t is not a variable and c and t are different
	halt with failure	
8	$(t)(t_1, \ldots, t_n) = (s)(s_1, \ldots, s_n)$	
	replace by $t = s, t_1 = s_1, \ldots, t_n = s_n$	
9	$(t)(t_1, \ldots, t_n) = (s)(s_1, \ldots, s_m)$	where $n \neq m$
	halt with failure	
10	$(t)(t_1, \ldots, t_n) = s$	where s is not a variable or a functional atom
	halt with failure	
11	$\neg t = \neg s$	
	replace by $t = s$	
12	$\neg t = s$	where s is not a variable or a negated expression
	halt with failure	
13	$t_1 \diamond t_2 = s_1 \diamond s_2$	where $\diamond$ is one of the binary connectives
	replace by $t_1 = s_1, t_2 = s_2$	
14	$t_1 \diamond t_2 = s$	where s is not a variable or of the form $s_1 \diamond s_2$
	halt with failure	
15	$Qx'(t) = Qx'(s)$	where Q is one of the quantifiers $\exists$ and $\forall$
	replace by $t = s$	
16	$Qx'(t) = s$	where Q is one of the quantifiers $\exists$ and $\forall$, s is not a variable and s is not of the form $Qx'(v)$
	halt with failure	

arguments t_i. We will use the notation $t(x_1, \ldots, x_n)$ to indicate an expression t for which $FV(t) \subseteq \{x_1, \ldots, x_n\}$.

Definition 6.1 (Marking bound variables)

$$
\begin{array}{ll}
c^m = c & \textit{for parameters } c, \\
x^m = x & \textit{for unmarked variables } x, \\
x'^m = x' & \textit{for marked variables } x', \\
((t)(t_1, \ldots, t_n))^m = (t^m)(t_1^m, \ldots, t_n^m) &
\end{array}
$$

$(\exists x(t))^m = \exists x'(t\{x/x'\})^m$ *where x is an unmarked variable,*
$(\exists x'(t))^m = \exists x'(t^m)$ *where x' is a marked variable,*
and similarly for universally quantified expressions.
In addition, $(\cdot)^m$ commutes with the logical connectives. □

The unification algorithm for AL will only yield the desired results if applied to terms in which all the bound variables are marked.

We will use the following terminology:

Definition 6.2 An expression t is *clean* if $t^m = t$. A set of equations $\{t_1 = s_1, \ldots, t_n = s_n\}$ is clean if all of the t_i and s_i are clean. A pair of sequents $\langle t_1, \ldots, t_n\rangle$, $\langle s_1, \ldots, s_n\rangle$ is clean if the associated set of equations $\{t_1 = s_1, \ldots, t_n = s_n\}$ is clean. □

In particular, marked variables can occur free in clean expressions, but in contrast, the latter do not contain bound occurrences of unmarked variables.

In the correctness proof of the algorithm we need the following notions of ambivalent substitution and ambivalent unifier:

Definition 6.3 σ is an *ambivalent substitution* if no marked variables occur in the domain of σ. □

Definition 6.4 Let t and s be expressions. σ is an *ambivalent unifier* for t and s, if σ is an ambivalent substitution and $t\sigma = s\sigma$. □

Observe that not every unifier is also an ambivalent unifier. For example, $\{x'/f(y)\}$ unifies $g(x')$ with $g(f(y))$, while no ambivalent unifiers exist for this tuple. In contrast, every ambivalent unifier is a unifier.

Proposition 6.5 *Let $L = \langle t_1, \ldots, t_n\rangle$ and $R = \langle s_1, \ldots, s_n\rangle$ be two sequents of clean expressions such that no marked variables occur free in any of the t_i and s_i. Then L and R are unifiable iff there is an ambivalent unifier for L and R.*

Proof: Suppose σ unifies L and R. Now let τ be the restriction of σ to $FV(L) \cup FV(R)$. Then, by assumption, τ is an ambivalent unifier of L and R. For the other direction, notice that every ambivalent unifier is a unifier. □

We relativise the notion of most general unifier to the class of ambivalent unifiers.

Definition 6.6 An ambivalent unifier θ for a set of (clean) expressions E is a *most general ambivalent unifier* for E (or, in short, an m.g.a.u.) if for every ambivalent unifier σ for E there is an ambivalent substitution τ such that $\sigma = \theta\tau$. An m.g.a.u. θ for E is *strong* if for every ambivalent unifier σ of E, $\sigma = \theta\sigma$. □

An m.g.a.u. is not necessarily an m.g.u., as the following counterexample witnesses.

Counterexample 6.7 Let $\theta = \{y/f(x')\}$. It is easy to check that θ is an m.g.a.u. for $E = \{p(\exists x'.f(x')) = p(\exists x'.y)\}$. Also, $\sigma = \{y/f(x'), x'/y\}$ is a unifier of E. However, suppose there is a substitution τ such that $\sigma = \theta\tau$. Then $f(x')\tau = f(x')$, and thus $x' \notin dom(\tau)$. But this contradicts $\{x'/y\} \in \{y/f(x')\}\tau$.

In addition, we need to adapt the notions of solved equation and equivalence between sets of equations.

Definition 6.8 A set of equations $\{t_1 = s_1, \ldots, t_n = s_n\}$ is *solved* if
1. the t_i are pairwise different, unmarked variables, and
2. no t_i occurs in any of the s_j. □

A solved set of equations $E = \{x_1 = s_1, \ldots, x_n = s_n\}$ determines a most general ambivalent unifier $\{x_1/s_1, \ldots, x_n/s_n\}$ for E (or, more precisely, for the associated pair of sequences $\langle x_1, \ldots, x_n\rangle$ and $\langle s_1, \ldots, s_n\rangle$). We need one more definition.

Definition 6.9 Two sets of equations of ambivalent terms are *equivalent* if they have the same ambivalent unifiers. □

Now we are in position to prove correctness of the unification algorithm.

Theorem 6.10 *The Martelli-Montanari unification algorithm for ambivalent syntax, when applied to a finite set of clean equations, results in a solved set of equations, determining a strong most general ambivalent unifier for the associated sequences in case an ambivalent unifier exists, and terminates with failure otherwise.*

Proof: The theorem follows from the following claims.

Claim 1 *Every non-halting action applied to a set of clean equations produces an equivalent set of clean equations.*

Proof: None of the actions 1, 4, 5, 6, 8, 11, 13, and 15 introduces a bound, unmarked variable. Therefore any of these actions transforms a set of clean equations into a new set of clean equations.

Preservation of equivalence is trivial for the actions 1, 5, 6, 8, 11, and 13. For action 4, preservation of equivalence is proven as usual.
For action 15, let σ be an ambivalent substitution. Then the following holds:

$(\exists x'(t))\sigma = (\exists x'(s))\sigma \iff \quad \{x' \notin dom(\sigma)\}$
$\exists x'(t\sigma) = \exists x'(s\sigma) \iff t\sigma = s\sigma.$ □

Claim 2 *The algorithm terminates.*

Proof: Consider the lexicographic order $<_3$ on $\boldsymbol{N}^3$. That is,

$$(n_1, n_2, n_3) <_3 (m_1, m_2, m_3)$$

iff

$$\begin{array}{ll} & n_1 < m_1 \\ \text{or} & n_1 = m_1 \ \& \ n_2 < m_2 \\ \text{or} & n_1 = m_1 \ \& \ n_2 = m_2 \ \& \ n_3 < m_3. \end{array}$$

Given a set of equations E, we call an unmarked variable x *solved in* E if, for some expression t, $x = t \in E$, and this is the only free occurrence of x in E. We call a variable x *unsolved in* E if x is unmarked and x is not solved in E.

With each set of clean equations E we now associate the following three functions:

$uns(E)$:= the number of unsolved variables in E,

$lfun(E)$:= the total number of occurrences of parameter symbols on the left hand side of the equations in E,

$lsym(E)$:= the total number of symbols (including brackets) occurring on the left hand side of equations of E.

We claim that each of the non-halting actions of the algorithm strictly reduces the triple $(uns(E), lfun(E), lsym(E))$.

Indeed, action 4 decreases $uns(E)$ by 1, while none of the successful actions increase $uns(E)$. Also, none of the other successful actions increase $lfun(E)$. Action 5 decreases $lfun(E)$ by 1 if t is a parameter, and decreases $lsym(E)$ by at least 1 otherwise. The actions 1, 6, 8, 11, 13, and 15 all decrease $lsym(E)$. Termination of the algorithm now follows from the well-foundedness of $<_3$.
□

Claim 3 *If the algorithm terminates successfully on a set of clean equations, then the final set of equations is solved.*

Proof: Suppose the algorithm terminates successfully, resulting in a set of clean equations E. Then none of the actions 5 – 16 applies to E, so the left hand expressions are all variables. Action 2 does not apply to E, so these are all unmarked variables. Finally, actions 1, 3, and 4 do not apply to E, so none of these variables occurs on the right hand side of any of the final equations. □

Claim 4 *If the algorithm halts with failure on a clean set of equations, then this set does not have an ambivalent unifier.*

Proof: Suppose the algorithm halts on the clean set E. If failure is the result of action 2, then the equation $x' = t$, where t is different from x' and t is not a variable, is a member of E. Clearly there are no ambivalent unifiers for $x' = t$. If failure is the result of action 3, then $x = t$ is an element of E, and there is no unifier σ for x and t, as $x\sigma$ is a proper subterm of $t\sigma$. If failure is the result of action 7, then $c = t$ is an element of E, and there is no unifier σ for c and t, because $c\sigma = c$, and $t\sigma$ is either a parameter different from c or an expression that is neither a variable nor a parameter. In the other cases, similar arguments apply. □
This completes the proof of Theorem 6.10. □

Clearly, this result is less general than the corresponding result for the original version of the unification algorithm. First of all, it applies only to those equations in which the sets of free and bound variables are distinct. It is an open question whether this restriction can be dropped. Second, the unifiers generated are most general only within the class of ambivalent unifiers, that is, those unifiers for which the domain does not contain any variable that occurs bound in the original set of equations. Counterexample 6.7 above suggests that this restriction can not be dropped.

As a corollary of the above theorem we now have a decidable unification algorithm for Prolog syntax. Clearly, in the Prolog case, where quantified expressions do not occur in term positions, we do not have to deal with the renaming of bound variables. The relevant actions in the Prolog version of the unification algorithm are the *Prolog actions* 1, 3, 4, 5, 6, 7, 8, 9, and 10. We leave it to the reader to check that the following holds.

Theorem 6.11 *The Martelli Montanari unification algorithm for Prolog's syntax, consisting of the Prolog actions 1, 3, 4, 5, 6, 7, 8, 9, and 10 results in a solved set of equations, determining a strong most general unifier in case a unifier exists, and terminates with failure otherwise.*

7 Comparison with Hilog

Hilog (Chen et al. [CKW93]) was developed as a language for higher order Logic Programming. In the discussion below, we assume that the reader is familiar with Hilog. We will here mainly point out some of the differences and similarities between Hilog and AL.

AL syntax is an extension of the syntax of Hilog. While the syntax of AL is fully ambivalent, Hilog syntax is characterised by the occurrence of terms in formula-, function- and predicate positions. (For example, $(c(a))(c)$, $x \rightarrow p(x)$, and $\forall x \exists y.(x(p))(y)$ are Hilog formulas.) Other than AL, Hilog does not admit quantified expressions in any unusual positions. As an example, $p(\forall x p(x))$ is an AL expression, but not a well-formed Hilog expression. In either syntax, the parameters are arityless.

While syntactically AL can be considered an extension of Hilog, the two logics differ considerably in semantics. The distinguishing feature of the (first-order) Hilog semantics is that each parameter of the language has a unique intension—that is, the interpretation function associates with each individual parameter (regardless of its contextual role), exactly one object in the domain of a model. With each intension then, several extensions are associated that capture the different contextual roles of the parameter. This extends to general terms. As a consequence, the schema $\forall x \forall y.(x = y \rightarrow \phi(x) \leftrightarrow \phi(y))$, and also $\forall x \forall y \forall z.(x = y \rightarrow x(z) \leftrightarrow y(z))$. In contrast, in the context of AL we have a *choice* between validating the above schema or not, by either taking all of ET as the equality theory, or restricting the equality to ET(I,II). Thus, unlike AL, Hilog is not appropriate for intensional logics, where opaqueness is usually desirable. (In contrast, observe that equivalence of terms does not imply their equality in neither Hilog nor AL.)

Another difference between AL and Hilog is found in comparing their respective relations to FOL. As we have seen, AL is a conservative extension of FOL without equality, and AL with the usual equality theory ET is a conservative extension of FOL with equality (Theorem 5.9). In Hilog, conservativity over FOL is restricted to the classes of equality free formulas and definite clauses in which equality only occurs in body-atoms. A standard derivational calculus for FOL is sound and complete w.r.t. AL semantics (Theorem 5.1). In contrast, in accordance to the intension of developing Hilog as basis for Logic Programming, paramodulation is introduced in [CKW93] as a derivational calculus for Hilog, and its soundness and completeness with respect to the Hilog semantics is proven. It is also shown that Hilog can be encoded in FOL, and a standard derivational calculus for FOL can be soundly used for the derivation of encoded Hilog formulas.

It should be noted that, despite the second order aspects of Hilog and AL syntax, both are essentially first order theories. In higher order predicate logic, the second order variables range over all relations over the intended domain. That is, in higher order semantics, functions and predicates are identified with their domain. In contrast, in Hilog and AL, the 'second order' variables range over elements of the intended domain. As a result, relation comprehension does not generally hold in AL and Hilog. That is, $\exists p \forall x_1 \ldots \forall x_n (p(x_1, \ldots, x_n) \leftrightarrow \phi(x_1, \ldots, x_n))$ is not generally valid in Hilog and AL. For example, relation comprehension is not true in AL for $\exists z q(x, y, z)$. But, unlike in Hilog, both $\exists p \forall x (p(x) \leftrightarrow \neg q(x))$ and $\exists p \forall x (p(x) \leftrightarrow \exists u (q((u))(x))$ are valid in AL.

A consequence of the identification of functions and predicates with their extensions in higher-order logic, is that the undecidability of this extensional equality carries over to the unification problem. Both for AL and Hilog however, unification is decidable.

8 Conclusion

The results reported show that, with minor modifications, basic proof theoretic results for first order predicate logic also go through for Ambivalent Logic. In particular, unification for AL is decidable, and both a standard derivational calculus and resolution are sound and complete inference methods for AL. A conservativity result shows that AL should be considered as a (syntactic) extension of first order predicate logic. All of the results reported relativise to subsystems of Ambivalent Logic that are frequently used in practice, in particular Prolog syntax and the syntax(es) used in Vanilla meta-programming and data bases. In addition, various properties of AL, such as the optional opaqueness with respect to equality and the flexibility of its syntax, suggest that AL may provide an interesting format for the representation of knowledge and belief.

Acknowledgements

This work reported here was originally inspired by ideas of Bob Kowalski and has benefited from many fruitful discussions with Barry Richards. Discussions with Bern Martens, Krzysztof Apt, and John Lloyd have always been very constructive. Recent comments and suggestions by Pat Hill, Danny De Schreye, Sten-Ake Tarnlund, and Johan van Benthem, helped improve this paper. We especially want to thank the editors of this volume for their support.

Marianne Kalsbeek was supported by the Dutch Organisation for Scientific Research (N.W.O.).

References

[Apt93] K. R. Apt. Declarative programming in Prolog. In D. Miller, editor, *Proc. International Symposium on Logic Programming*, pages 11–35. MIT Press, 1993.

[AT] K.R. Apt and F. Teusink. Comparing negation in logic programming and in Prolog. This volume.

[CKW93] W. Chen, M. Kifer, and D.S. Warren. Hilog: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, 1993.

[Doe94] H. C. Doets. *From Logic to Logic Programming.* MIT Press, 1994.

[Gab92] D. Gabbay. Metalevel features in the object level: modal and temporal logic programming III. In L.Farinas del Cerro and M. Penttonen, editors, *Intensional logics for programming*, pages 85–124. Clarendon Press, 1992.

[Jia94a] Y. Jiang. Ambivalent logic as the semantic basis for metalogic programming: I. In P. Van Hentenryck, editor, *Proceedings of the International Conference on Logic Programming*, pages 387–401. MIT Press, June 1994.

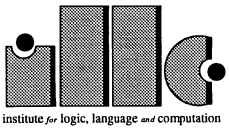
[Jia94b] Y. Jiang. Ambivalent logic as the semantic basis of metalogic programming: Theory and practice. Technical report, Imperial College, Dept. of Computing, 1994.

[Kal] M. Kalsbeek. Correctness of the Vanilla meta-interpreter and ambivalent syntax. This volume.

[KK91] R.A. Kowalski and J. Kim. A metalogic programming approach to multi-agent knowledge and belief. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 231–246. Academic Press, 1991.

[Ric74] B. Richards. A point of reference. *Synthese*, 28:431–445, 1974.

[Sch89] U. Schöning. *Logic for Computer Scientists*, volume 8 of *Progress in Computer Science and Applied Logic*. Birkhauser, 1989.



ILLC Research Reports and Technical Notes

Coding for Research Reports: *Series-Year-Number*, with LP = Logic, Philosophy and Linguistics; ML = Mathematical Logic and Foundations; CL = Compuational Linguistics; CT = Computation and Complexity Theory; X = Technical Notes.
All previous ILLC-publications are available from the ILLC bureau. For prepublications before 1994, contact the bureau.

ML-94-01 *Domenico Zambella, Notes on polynomially bounded arithmetic*
ML-94-02 *Domenico Zambella, End Extensions of Models of Linearly Bounded Arithmetic*
ML-94-03 *Johan van Benthem, Dick de Jongh, Gerard Renardel de Lavalette, Albert Visser, NNIL, A Study in Intuitionistic Propositional Logic*
ML-94-04 *Michiel van Lambalgen, Independence Structures in Set Theory*
ML-94-05 *V. Kanovei, IST is more than an Algorithm to prove ZFC Theorems*
ML-94-06 *Lex Hendriks, Dick de Jongh, Finitely Generated Magari Algebras and Arithmetic*
ML-94-07 *Sergei Artëmov, Artëm Chuprina, Logic of Proofs with Complexity Operators*
ML-94-08 *Andreja Prijatelj, Free Algebras Corresponding to Multiplicative Classical Linear Logic and some Extensions*
ML-94-09 *Giovanna D'Agostino, Angelo Montanari, Alberto Policriti, A Set-Theoretic Translation Method for Polymodal Logics*
ML-94-10 *Elena Nogina, Logic of Proofs with the Strong Provability Operator*
ML-94-11 *Natasha Alechina, On One Decidable Generalized Quantifier Logic Corresponding to a Decidable Fragment of First-Order Logic*
ML-94-12 *Victor Selivanov, Fine Hierarchy and Definability in the Lindenbaum Algebra*
ML-94-13 *Marco R. Vervoort, An Elementary Construction of an Ultrafilter on $\aleph_1$ Using the Axiom of Determinateness*

LP-94-01 *Dimitar Gelev, Introducing Some Classical Elements of Modal Logic to the Propositional Logics of Qualitative Probabilities*
LP-94-02 *Andrei Arsov, Basic Arrow Logic with Relation Algebraic Operators*
LP-94-03 *Jerry Seligman, An algebraic appreciation of diagrams*
LP-94-04 *Kazimierz Świrydowicz, A Remark on the Maximal Extensions of the Relevant Logic* R
LP-94-05 *Natasha Kurtonina, The Lambek Calculus: Relational Semantics and the Method of Labelling*
LP-94-06 *Johan van Benthem, Dag Westerståhl, Directions in Generalized Quantifier Theory*
LP-94-07 *Nataša Rakić, Absolute Time, Special Relativity and ML$^\nu$*
LP-94-08 *Daniel Osherson, Scott Weinstein, Dick de Jongh, Eric Martin, Formal Learning Theory*
LP-94-09 *Harry P. Stein, Linguistic Normativity and Kripke's Sceptical Paradox*
LP-94-10 *Harry P. Stein, The Hazards of Harmony*
LP-94-11 *Paul Dekker, Predicate Logic with Anaphora*
LP-94-12 *Paul Dekker, Representation and Information in Dynamic Semantics*
LP-94-13 *Jeroen Groenendijk, Martin Stokhof, Frank Veltman, This Might Be It*
LP-94-14 *Jeroen Groenendijk, Martin Stokhof, Frank Veltman, Update Semantics for Modal Predicate Logic*
LP-94-15 *Henk Zeevat, The Mechanics of the Counterpart Relation*
LP-94-16 *David Beaver, When Variables Don't Vary Enough*
LP-94-17 *David Beaver, Accommodating Topics*
LP-94-18 *Claire Gardent, Discourse Multiple Dependencies*

LP-94-19 *Renate Bartsch, The Relationship between Connectionist Models and a Dynamic Data-Oriented Theory of Concept Formation*

LP-94-20 *Renate Bartsch, The Myth of Literal Meaning*

LP-94-21 *Noor van Leusen, The Interpretation of Corrections*

LP-94-22 *Maarten Marx, Szabolcs Mikulás, István Németi, Taming Arrow Logic*

LP-94-23 *Jaap van der Does, Cut Might Cautiously*

LP-94-24 *Michiel Leezenberg, Metaphor and Literacy*

CT-94-01 *Harry Buhrman and Leen Torenvliet, On the Cutting Edge of Relativization: the Resource Bounded Injury Method*

CT-94-02 *Alessandro Panconesi, Marina Papatriantafilou, Philippas Tsigas, Paul Vitányi, Randomized Wait-Free Distributed Naming*

CT-94-03 *Ming Lee, John Tromp, Paul Vitányi, Sharpening Occam's Razor (extended abstract)*

CT-94-04 *Ming Lee and Paul Vitányi, Inductive Reasoning*

CT-94-05 *Tao Jiang, Joel I. Seiferas, Paul M.B. Vitányi, Two heads are Better than Two Tapes*

CT-94-06 *Guido te Brake, Joost N. Kok, Paul Vitányi, Model Selection for Neural Networks: Comparing MDL and NIC*

CT-94-07 *Charles H. Bennett, Péter Gács, Ming Li, Paul M.B. Vitányi, Wojciech H. Zurek, Thermodynamics of Computation and Information Distance*

CT-94-08 *Krzysztof R. Apt, Peter van Emde Boas and Angelo Welling, The STO-problem is NP-hard*

CT-94-09 *Klaus Ambos-Spies, Sebastiaan A. Terwijn, Zheng Xizhong, Resource Bounded Randomness and Weakly Complete Problems*

CT-94-10 *Klaus Ambos-Spies, Hans-Christian Neis, Sebastiaan A. Terwijn, Genericity and Measure for Exponential Time*

CT-94-11 *Natasha Alechina, Logic with Probabilistic Operators*

CT-94-12 *Marianne Kalsbeek, Gentzen Systems for Logic Programming Styles*

CT-94-13 *Peter Desain, Henkjan Honing, CLOSe to the edge? Advanced Object-Oriented Techniques in the Representation of Musical Knowledge*

CT-94-14 *Henkjan Honing, The Vibrato Problem. Comparing two Ways to Describe the Intraction between the Continuous Knowledge and Discrete Components in Music Representation Systems*

X-94-01 *Johan van Benthem, Two Essays on Semantic Modelling*

X-94-02 *Vladimir Kanovei, Michiel van Lambalgen, Another Construction of Choiceless Ultrapower*

X-94-03 *Natasha Alechina, Michiel van Lambalgen, Correspondence and Completeness for Generalized Quantifiers*

X-94-04 *Harry P. Stein, Primitieve Normen Linguïstische normativiteit in het licht van Kripke's sceptische paradox*

X-94-05 *Johan van Benthem, Logic and Argumentation*

X-94-06 *Natasha Alechina, Philippe Smets, A Note on Modal Logics for Partial Belief*

X-94-07 *Michiel Leezenberg, The Shabak and the Kakais: Dynamics of Ethnicity in Iraqi Kurdistan*

LP-95-01 *Marten Trautwein, Assessing Complexity Results in Feature Theories*

ML-95-01 *Michiel van Lambalgen, Randomness and Infinity*

CT-95-01 *Marianne Kalsbeek, A Vademecum of Ambivalent Logic*

Titles in the ILLC Dissertation Series:

1993-1 *Transsentential Meditations; Ups and downs in dynamic semantics*, Paul Dekker
1993-2 *Resource Bounded Reductions*, Harry Buhrman
1993-3 *Efficient Metamathematics*, Rineke Verbrugge
1993-4 *Extending Modal Logic*, Maarten de Rijke
1993-5 *Studied Flexibility*, Herman Hendriks
1993-6 *Aspects of Algorithms and Complexity*, John Tromp
1994-1 *The Noble Art of Linear Decorating*, Harold Schellinx
1994-2 *Generating Uniform User-Interfaces for Interactive Programming Environments*, Jan Willem Cornelis Koorn
1994-3 *Process Theory and Equation Solving*, Nicoline Johanna Drost
1994-4 *Calculi for Constructive Communication, a Study of the Dynamics of Partial States*, Jan Jaspars
1994-5 *Executable Language Definitions, Case Studies and Origin Tracking Techniques*, Arie van Deursen
1994-6 *Chapters on Bounded Arithmetic & on Provability Logic*, Domenico Zambella
1994-7 *Adventures in Diagonalizable Algebras*, V. Yu. Shavrukov
1994-8 *Learnable Classes of Categorial Grammars*, Makoto Kanazawa
1994-9 *Clocks, Trees and Stars in Process Theory*, Wan Fokkink
1994-10 *Logics for Agents with Bounded Rationality*, Zhisheng Huang
1995-1 *On Modular Algebraic Prototol Specification*, Jacob Brunekreef
1995-2 *Investigating Bounded Contraction* , Andreja Prijatelj
1995-3 *Algebraic Relativization and Arrow Logic*, Maarten Marx
1995-4 *Study on the Formal Semantics of Pictures*, Dejuan Wang
1995-5 *Generation of Program Analysis Tools*, Frank Tip