LPF and MPL$_\omega$ — A Logical Comparison of VDM SL and COLD-K

C.A. Middelburg* and G.R. Renardel de Lavalette**

Abstract

This paper compares the finitary three-valued logic LPF and the infinitary two-valued logic MPL$_\omega$, the logics underlying VDM SL and COLD-K. These logics reflect different approaches to reasoning about partial functions and bringing recursive function definitions into proofs. The purpose of the comparison is to acquire insight into the relationship between these approaches. A natural translation from LPF to MPL$_\omega$ is given. It is shown that what can be proved remains the same after translation, in case strictness axioms are added to LPF or removed from MPL$_\omega$. The translation from LPF to MPL$_\omega$ is extended to recursive function definitions and this translation is next used to justify some ways of bringing the definitions of partial functions into proofs using LPF.

1 Introduction

Functions specified in VDM SL [Jon90, JS90] or COLD-K [Jon89] are generally partial functions. Partial functions give rise to non-denoting terms. This makes reasoning about partial functions problematic in classical first-order logic. The underlying logics of VDM SL and COLD-K, viz. LPF [BCJ84, Che86] and MPL$_\omega$ [KR89], have different approaches to solve this problem.

Classical first-order logic has been used fruitfully to describe and formalize mathematical concepts and theories. But there was always the problem of non-denoting terms: terms that do not refer to objects in the intended domain (they are also called undefined terms). The classical example is division by zero. In most formalizations, this problem is sidestepped by considering division not as a binary function but as a ternary relation D, where $D(x, y, z)$ means that x divided by y yields z, or by restriction of the language: allowing t/t' as a term only if $t' \neq 0$. In recursion theory, however, it became clear that partial functions are essential in any decent theory of recursive functions.

*Dept. of Computer Science, PTT Research, Dr. Neher Laboratories, P.O. Box 421, 2260 AK Leidschendam, The Netherlands; e-mail: CA.Middelburg@pttrnl.nl.

**Software Engineering Research Centre, P.O. Box 424, 3500 AK Utrecht, The Netherlands and Section Applied Logic, University of Utrecht, P.O. Box 80126, 3508 TC Utrecht, The Netherlands; e-mail: renardel@serc.nl.

Scott provided a formal basis for reasoning about partial functions in [Sco67], later extended to intuitionistic logic with sheaf semantics in [Sco79]. Semantically, the situation is as follows: the intended domain is extended with an object $\perp$ (undefined) to which the undefined terms refer. This idea has been adopted by, for example, Beeson in [Bee85] and the second author in [Ren84]. This approach has also been followed in MPL_ω, the logic underlying COLD-K. It can be characterized as follows:

- there is a definedness predicate $\downarrow$, where $t\downarrow$ means that t is denoting (or is defined, exists, refers to a defined object);
- if an atomic formula is true, then all terms occurring in it are defined (the strictness property).

Another approach has been followed in LPF, the logic underlying VDM SL. Here the possibility of undefinedness is extended to the formulae by adding a truth value N (neither-true-nor-false), so terms and formulae are in this respect treated on an equal footing. This makes LPF a non-classical logic with three truth values. So the definition of the logical connectives has to be extended. This is done by taking the definitions obtained by extending the classical truth-conditions and falsity-conditions with a clause yielding N for the other cases. In other words, Kleene's strong three-valued connectives [Kle52] are taken. For example, we have for any propositions A and A':

$$A \wedge A' = \begin{array}{ll} \mathsf{T} & \text{if } A = \mathsf{T} \textit{ and } A' = \mathsf{T}, \\ \mathsf{F} & \text{if } A = \mathsf{F} \textit{ or } A' = \mathsf{F}, \\ \mathsf{N} & \text{otherwise.} \end{array}$$

This extension to the three-valued case yields monotonic operators with respect to the ordering $\mathsf{N} \preceq \mathsf{T}, \mathsf{F}$, the ordering of information contents. Furthermore, there is only a weak strictness property:

> if an atomic formula $t = t'$ is true or false, then both t and t' are defined.

One might expect the following strong version here:

> if an atomic formula is true or false, then all terms occurring in it are defined.

There are reasons to think that such a version of strictness is an intended property of LPF. It is strongly suggested in Section 3.3 of [Jon86, Jon90], as is shown in [Tho89] (see Section 7 of the current paper for LPF with strictness axioms).

In this paper we consider these two ways of dealing with undefinedness more closely, in a comparison of the underlying logics of the specification languages VDM SL and COLD-K. In Section 2, these logics are described in broad outline. Some other logics which handle partial functions are discussed in Section 3. Sections 4–8 constitute the body of this paper. In Section 4 we present $\text{ML}^=$, many-sorted classical logic with equality, by giving definitions of the language, the proof system, and the interpretation of $\text{ML}^=$. It is used as the starting point for the presentations of LPF and MPL_ω in Sections 5 and 6. LPF is embedded into MPL_ω in Section 7. Section 8 is concerned with recursive function definitions in LPF and MPL_ω. Conclusions and final remarks are given in Section 9.

In [CJ90], Cheng and Jones compare the usability of different logics in handling partial functions. Their brief treatment of logics based on Scott's idea and their way of bringing

the definitions of partial functions into proofs using LPF provided an important stimulus to write this paper.

2 Overview of LPF and MPL$_\omega$

LPF and MPL$_\omega$ reflect different approaches to deal with non-denoting terms in formulae. This section describes these logics informally and in broad outline.

2.1 LPF

LPF is a non-classical finitary first-order logic of partial functions with equality. Its typical features are obtained by rather drastic changes to classical first-order logic. Classical reasoning is invalidated on a large scale. A mathematically precise presentation of LPF is given by Cheng in [Che86].

The logic LPF adopts an approach to solve the problem with non-denoting terms in formulae, which does not stay within the realm of classical, two-valued logics. Atomic formulae that contain non-denoting terms may be logically neither-true-nor-false. Thus, the assumption of the "excluded middle" is given up. Yet, the classical truth-conditions and falsehood-conditions for logical connectives and quantifiers are adopted. The formula concerned is classified as neither-true-nor-false exactly when it cannot be classified as true or false by these conditions. Likewise, an equation $t_1 = t_2$ is classified as neither-true-nor-false exactly when t_1 or t_2 is non-denoting.

This approach leaves open the treatment of free variables. In contrast with MPL$_\omega$, free variables are always denoting — just as bound variables.

A definedness connective Δ guarantees expressive completeness of the connectives of LPF for three-valued truth functions. $\Delta\, A$ is true if the formula A can be classified as either true or false, and false otherwise.

LPF lacks countably infinite conjunctions, which we shall see in MPL$_\omega$. Neither is there another feature in LPF that allows recursive or inductive definitions to be expressed as formulae. LPF lacks descriptions, which would also give rise to non-denoting terms, as well.

The formation rules for LPF are the usual formation rules with an additional rule for formulae of the form $\Delta\, A$. The proof system of LPF presented in [Che86] is a Gentzen-type sequent calculus that does not resemble a classical one. Even so, adding one axiom schema — stating the assumption of the excluded middle — would make it a complete proof system for classical first order logic with equality.

The formulae that contain only function and predicate symbols from a certain set Σ constitute the language of LPF over Σ. The structures used for interpretation of the language of LPF over Σ consist of an interpretation of every symbol in Σ as well as an interpretation of the equality symbol that is in accordance with the above-mentioned treatment of non-denoting terms in equations. The connectives and quantifiers are always interpreted according to the outlined classification of formulae. The interpretation of the language in the structures concerned is sound with respect to the proof system. The proof

system is complete with respect to the interpretation of the language in these structures.

The outlined version of LPF is the one-sorted version presented in [Che86]. In the current paper, that version is generalized to a many-sorted version of LPF. For convenience, the many-sorted version is compared with MPL_ω.

2.2 MPL_ω

MPL_ω is a many-sorted infinitary first-order logic of partial functions with equality. Its typical features are mainly obtained by additions to language and proof system of classical first-order logic. Classical reasoning is only invalidated on a small scale. The language, proof system and interpretation of MPL_ω are introduced by Koymans and Renardel de Lavalette in [KR89].

The logic MPL_ω adopts an approach to solve the problem with non-denoting terms in formulae, which stays within the realm of classical, two-valued logics. Atomic formulae that contain non-denoting terms are logically false. In this way, the assumption of the excluded middle does not have to be given up. When a formula cannot be classified as true, it is inexorably classified as false. No further distinction is made. However, denoting terms and non-denoting terms can be distinguished. $t =_S t$ means that t is denoting (for terms t of sort S), which is also written $t{\downarrow_S}$. There is a standard undefined constant symbol $\uparrow_S$ for every sort symbol S. $\uparrow_S$ is a non-denoting term of sort S.

If $A_0, A_1, A_2, \ldots$ are countably many formulae, then the formula $\bigwedge_n A_n$ can be formed. This allows a large class of recursive and inductive definitions of functions and predicates to be expressed as formulae of MPL_ω. This was first sketched in [KR89, Section 4] and later worked out in detail by Renardel de Lavalette in [Ren89].

If A is a formula, then the term $\iota x{:}\, S\,(A)$ can be formed which is called a description. Its intended meaning is the unique value x of sort S that satisfies A if such a unique value exists and undefined otherwise. This means that not every description will be denoting. Descriptions can be eliminated: it is possible to translate formulae containing descriptions into logically equivalent formulae without descriptions.

The formation rules for MPL_ω are the usual formation rules with an additional rule for descriptions and with the rule for binary conjunctions replaced by the rule for countably infinite conjunctions from L_ω [Kar64, Kei71] (classical first-order logic with countably infinite conjunctions). The proof system of MPL_ω presented in [KR89] is a Gentzen-type sequent calculus that resembles one for L_ω. Obviously, there are additional axioms for equality, undefined, and description.

The formulae that contain only sort, function and predicate symbols from a certain set Σ constitute the language of MPL_ω over Σ. The structures used for interpretation of the language of MPL_ω over Σ consist of an interpretation of every symbol in Σ as well as an interpretation of each of the equality symbols associated with the sort symbols in Σ. These interpretations have to be in accordance with the outlined treatment of non-denoting terms. The classical interpretation of the connectives and quantifiers is used. This means that, unlike free variables, bound variables are always denoting. The interpretation of the language in the structures concerned is sound with respect to the proof system. The proof system is complete with respect to the interpretation of the

language in these structures.

3 Other Logics for Partial Functions

There are other logics for partial function. They differ from LPF and MPL_ω in various ways. The key differences are discussed in this section.

3.1 Other Three-valued Logics

The key differences between the various three-valued logics are with respect to:

- what logical connectives and quantifiers are taken as basic,
- whether equality of some kind is taken as basic and if so which kind,
- what model-theoretic notion of logical consequence is taken to underlie the proof system.

Negation, disjunction, and existential quantifier are basic in the version of LPF presented in [Che86]. Each behaves according to its classical truth-condition and falsehood-condition; only if neither of them meets, it will yield neither-true-nor-false. This is Kleene's way of extending the classical connectives and quantifiers to the three-valued case [Kle52]. In addition, there are two basic connectives which have no classical counterparts: a nullary connective designating neither-true-nor-false and a unary connective for definedness of formulae. All possible connectives for three-valued logics are definable by the above-mentioned four basic connectives. The definable connectives include McCarthy's connectives [McC67] and Łukasiewicz's connectives [Łuk67]. McCarthy's quantifiers are also definable. Quantifiers which bind variables that may be non-denoting are not definable, but such quantifiers are uncommon in a three-valued setting.

There are several ways of extending classical equality to the three-valued case. Admissible kinds of equality only differ in their treatment of non-denoting terms:

- *weak equality*: if either t_1 or t_2 is non-denoting, then $t_1 = t_2$ is neither-true-nor-false;
- *strong equality*: if either t_1 or t_2 is non-denoting, then $t_1 = t_2$ is true whenever both t_1 and t_2 are non-denoting and false otherwise;
- *existential equality*: if either t_1 or t_2 is non-denoting, then $t_1 = t_2$ is false.

Weak equality is basic in LPF. Due to the presence of the definedness connective, strong equality and existential equality are also definable. Thus, LPF encompasses most three-valued logics with respect to their connectives, quantifiers and equality predicates.

The following are the intuitive ideas that underlie sensible notions of logical consequence for three-valued logics:

- from premises that are not false, one can draw conclusions that are not false;
- from premises that are true, one can draw conclusions that are not false;
- from premises that are true, one can draw conclusions that are true;
- from premises that are true, one can draw conclusions that are true, and conclusions that are false must arise from premises that are false.

Note that "from premises that are not false, one can draw conclusions that are true" does not correspond to a sensible notion of logical consequence, since not every formula will be a consequence of itself.

Logical consequence for three-valued logics according to the first idea amounts to logics in which the definedness of any formula (i.e. the property that the formula is either true or false) can be treated as true. It means that a separate proof is needed to establish the definedness. For formulae formed with Kleene's or McCarthy's connectives and Kleene's quantifiers, logical consequence for three-valued logics according to the second idea reduces to classical logical consequence for two-valued logics. For formulae formed with Kleene's connectives and Kleene's quantifiers, logical consequence for three-valued logics according to the third idea coincides with classical logical consequence for two-valued logics except for the absence of what depends anyhow on the excluded middle. That is, it is lacking exactly what does not leave room for formulae which are neither true nor false. This is what one naturally expects from a three-valued logic where the additional truth value is interpreted as neither true nor false. The last idea actually combines the first idea and the third idea. The corresponding notion of logical consequence seems too strong for a logic for the formal specification and verified design of software systems, which is only concerned with drawing true conclusions from true premises.

For LPF, the underlying notion of logical consequence is the third one of the above-mentioned notions. The first notion underlies Owe's weak logic [Owe84]. The second notion underlies the logic PFOL which is presented in [GL90] together with reflections on the semantic options for three-valued logics and their relationships. Amongst other things, it is shown in [GL90] that logical consequence according to the first or second notion can be defined over logical consequence according to the third notion (in the latter case, it can only be defined provided that the definedness connective is available). Similar reflections are also presented in [KTB88], but there a definedness connective is not mentioned. In [KTB88], attention is also paid to McCarthy's connectives. The last notion of logical consequence underlies Blamey's partial logic [Bla86]. It follows immediately that the above-mentioned definability results extend to this notion. Because of these definability results for other notions of logical consequence, LPF also encompasses most three-valued logics with respect to their underlying notion of logical consequence.

3.2 Other Two-valued Logics

The various two-valued logics for partial functions are quite similar with respect to their connectives and their underlying notion of logical consequence. The quantifiers differ slightly in the treatment of the bound variables.

The key differences between the various two-valued logics are with respect to:

- the treatment of free variables and bound variables,
- whether equality of some kind is taken as basic and if so which kind,
- whether two levels of truth values are used.

In MPL_ω, free variables may not denote but bound variables always do. The same happens in Scott's free logic [Sco67], Plotkin's PFL (Partial Function Logic) [Plo85] and other free logics (for a discussion of free logics, see [Ben86]). Due to this treatment of free and bound variables, frequent reasoning about the definedness of terms can be avoided. This contrasts with Scott's LCF (Logic of Computable Functions), which was renamed PPλ [GMW79, Pau87], where bound variables may as well not denote: frequent reasoning about undefined is customary. The treatment of variables in PPλ is a consequence of the decision not to differ from classical first-order logic with respect to logical axioms and inference rules. Quantifications, where the bound variable may be non-denoting, can be treated as abbreviations in MPL_ω. Both free variables and bound variables always denote in Beeson's LPT (Logic of Partial Terms) [Bee88]. Thus, LPT is kept closer to classical first-order logic than the free logics.

MPL_ω is a two-valued logic with existential equality. Strong equality is definable. Note that weak equality makes no sense in two-valued logics. All of this is the same as in Scott's free logic and Beeson's LPT. But PPλ is a logic with strong equality. This means that it does not differ from classical logic with respect to its kind of equality as well.

PPλ is the only one of the above-mentioned logics which is classical, but nevertheless it handles partial functions. That result is reached by adopting a layered approach which give rise to two levels of truth values. Terms, which are intended to represent computable objects, can be undefined (as computations commonly involve applications of partial functions). Thus, three "computational truth values", including an undefined truth value, is made provision for. Formulae, which are meant to be assertions about computable objects, must be either true or false (which corresponds to the truth or falsehood, respectively, of the assertions). So there are only two "logical truth values".

Roughly speaking, there are two main approaches of handling partial functions in a two-valued logic: the approach adopted by free logics and the layered approach adopted by PPλ. MPL_ω adopts the former approach; it encompasses most free logics with respect to their treatment of free and bound variables and their predicates concerning equality. Although MPL_ω does not adopt the latter approach, it is shown in the current paper that it allows for a layered approach: formulae of LPF can be treated as terms of MPL_ω.

4 Many-sorted Classical Logic with Equality

MPL_ω and LPF are quite different. Nevertheless, $\mathrm{ML}^=$, many-sorted classical logic with equality, provides a convenient starting point for a presentation of both logics. Hence, language, proof system and interpretation of $\mathrm{ML}^=$ are presented first.

4.1 Signatures

A language of $\mathrm{ML}^{=}$ is constructed with sort symbols, function symbols and predicate symbols that belong to a certain set, called a signature. For a given signature, say Σ, the language concerned is called the language of $\mathrm{ML}^{=}$ over signature Σ or the language of $\mathrm{ML}^{=}(\Sigma)$. The corresponding proof system and interpretation are analogously called the proof system of $\mathrm{ML}^{=}(\Sigma)$ and the interpretation of $\mathrm{ML}^{=}(\Sigma)$, respectively.

We assume a set $SORT$ of *sort symbols*, a set $FUNC$ of *function symbols*, and a set $PRED$ of *predicate symbols*. f, g range over $FUNC$, P, Q range over $PRED$, and $S, S_1, S_2, \ldots$ range over $SORT$. Every $f \in FUNC$ has a *function type* $S_1 \times \cdots \times S_n \to S_{n+1}$ and every $P \in PRED$ has a *predicate type* $S_1 \times \cdots \times S_n$ ($S_1, \ldots, S_{n+1} \in SORT$). To indicate this, we use the notation $f\colon S_1 \times \cdots \times S_n \to S_{n+1}$ and $P\colon S_1 \times \cdots \times S_n$. Function symbols of function type $\to S$ are also called *constant symbols* of sort S. For every $S \in SORT$, there is a standard predicate symbol $=_S\colon S \times S$, called *equality*.

A *signature* Σ is a finite subset of $SORT \cup FUNC \cup PRED$ such that

for all $f \in \Sigma, f\colon S_1 \times \cdots \times S_n \to S_{n+1} \;\Rightarrow\; S_1, \ldots, S_{n+1} \in \Sigma$;
for all $P \in \Sigma, P\colon S_1 \times \cdots \times S_n \;\Rightarrow\; S_1, \ldots, S_n \in \Sigma$.

We write $\mathrm{S}(\Sigma)$ for $\Sigma \cap SORT$, $\mathrm{F}(\Sigma)$ for $\Sigma \cap FUNC$, $\mathrm{P}(\Sigma)$ for $\Sigma \cap PRED$, $\mathrm{SP}(\Sigma)$ for $\{=_S |\ S \in \mathrm{S}(\Sigma)\}$. $SIGN$ denotes the set of all signatures for $\mathrm{ML}^{=}$.

We also assume a set VAR of *variable symbols*. $x, y, z, x_1, x_2, \ldots$ range over VAR. Every $x \in VAR$ has a *sort* S ($S \in SORT$).

We write $\mathcal{V}$ for $SORT \cup FUNC \cup PRED \cup VAR$. We write $w \equiv w'$, where $w, w' \in \mathcal{V}$, to indicate that w and w' are identical symbols.

Furthermore, it is assumed that $SORT$, $FUNC$, $PRED$, and VAR are four disjoint sets and $=_S \notin \mathcal{V}$ for all $S \in SORT$.

4.2 Language of $\mathrm{ML}^{=}(\Sigma)$

Terms and Formulae

The language of $\mathrm{ML}^{=}(\Sigma)$ contains terms and formulae. They are constructed according to the formation rules which are given below. $t, t', t_1, t_1', t_2, t_2', \ldots$ range over terms and $A, A_0, A_0', A_1, A_1', \ldots$ range over formulae.

The terms of $\mathrm{ML}^{=}(\Sigma)$ are inductively defined by the following formation rules:

1. variable symbols of sort S are terms of sort S, for any $S \in \mathrm{S}(\Sigma)$;
2. if $f \in \mathrm{F}(\Sigma)$, $f\colon S_1 \times \cdots \times S_n \to S_{n+1}$ and $t_1, \ldots, t_n$ are terms of sorts $S_1, \ldots, S_n$, respectively, then $f(t_1, \ldots, t_n)$ is a term of sort S_{n+1}.

The formulae of $\mathrm{ML}^{=}(\Sigma)$ are inductively defined by the following formation rules:

1. $\top$ and $\bot$ are formulae;
2. if $P \in \mathrm{P}(\Sigma) \cup \mathrm{SP}(\Sigma)$, $P\colon S_1 \times \cdots \times S_n$ and $t_1, \ldots, t_n$ are terms of sorts $S_1, \ldots, S_n$, respectively, then $P(t_1, \ldots, t_n)$ is a formula;

3. if A is formula, then $\neg A$ is a formula;
4. if A_1 and A_2 are formulae, then $A_1 \wedge A_2$ is a formula;
5. if A is a formula and x is a variable of sort S, $S \in \mathrm{S}(\Sigma)$, then $\forall x{:}\, S\,(A)$ is a formula.

The string representation of formulae as suggested by these formation rules can lead to syntactic ambiguities. Parentheses are used to avoid such ambiguities.

For every set Γ of formulae of $\mathrm{ML}^{=}(\Sigma)$, $sig(\Gamma)$, the signature of Γ, is the smallest signature such that for every formula $A \in \Gamma$, A is a formula of $\mathrm{ML}^{=}(sig(\Gamma))$.

$\mathcal{T}_{ML^{=}}(\Sigma)$ denotes the set of all terms of $\mathrm{ML}^{=}(\Sigma)$.
$\mathcal{L}_{ML^{=}}(\Sigma)$ denotes the set of all formulae of $\mathrm{ML}^{=}(\Sigma)$.

Notational Conventions

Constant symbols may be used as terms, i.e. in terms of the form $f(t_1, \ldots, t_n)$ the parentheses may be omitted whenever $n = 0$.

The equality symbols ($=_S$) are used in infix notation. Moreover, they are used without subscript when this causes no ambiguity.

Sometimes $\forall x_1{:}\, S_1\,(\cdots \forall x_n{:}\, S_n\,(A)\cdots)$ is simply written as $\forall x_1{:}\, S_1, \ldots, x_n{:}\, S_n\,(A)$.

Absent from the language of $\mathrm{ML}^{=}(\Sigma)$ are disjunction, existential quantification, etc. They are defined as abbreviations:

$$\begin{array}{ll} A_1 \vee A_2 & := \neg(\neg A_1 \wedge \neg A_2), \\ A_1 \rightarrow A_2 & := \neg A_1 \vee A_2, \\ A_1 \leftrightarrow A_2 & := (A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1), \\ \exists x{:}\, S\,(A) & := \neg \forall x{:}\, S\,(\neg A). \end{array}$$

The need to use parentheses in the string representation of formulae is reduced by ranking the precedence of the logical symbols $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$. The enumeration presents this order from the highest precedence to the lowest precedence.

Free Variables and Substitution

For a term or formula E of $\mathrm{ML}^{=}(\Sigma)$, *free*(E) denotes the set of *free variable* of E, which is defined as usual. We write *free*(Γ), where Γ is a set of formulae, for $\bigcup\{\mathit{free}(A) \mid A \in \Gamma\}$. A variable symbol x is called *free in* Γ if $x \in \mathit{free}(\Gamma)$.

Substitution for variables is also defined as usual. Let x be a variable symbol, t be a term (x and t of the same sort) and E be a term or formula. Then $[x := t]E$ is the result of replacing the term t for the free occurrences of the variable symbol x in E, avoiding that free variables in t become bound by means of renaming of bound variables.

4.3 Proof System of $\mathrm{ML}^{=}(\Sigma)$

Sequents

The proof system of $\mathrm{ML}^{=}(\Sigma)$ is formulated as a Gentzen-type sequent calculus.

A *sequent* is an expression of the form $\Gamma \vdash \Delta$, where Γ and Δ are finite sets of formulae of $\mathrm{ML}^{=}(\Sigma)$. Instead of $\{\,\} \vdash \Delta$ we write $\vdash \Delta$, and instead of $\Gamma \vdash \{\,\}$ we write $\Gamma \vdash$.

The intended meaning of the sequent $\Gamma \vdash \Delta$ is that the conjunction of the formulae in Γ entails the disjunction of the formulae in Δ. A sequent is proved by a derivation obtained by using the axiom schemas and rules of inference given below.

Axiom Schemas and Rules of Inference

$\Gamma, \Delta, \Gamma', \Delta', \ldots$ stand for arbitrary finite sets of formulae of $\mathrm{ML}^{=}(\Sigma)$.
A, A_1, A_2 stand for arbitrary formulae of $\mathrm{ML}^{=}(\Sigma)$.
t, t_1, t_2 stand for arbitrary terms (of appropriate sorts) of $\mathrm{ML}^{=}(\Sigma)$.
x, y, z stand for arbitrary variable symbols (of appropriate sorts).
S stands for an arbitrary sort symbol in Σ.

We write Γ, Δ for $\Gamma \cup \Delta$ and A for $\{A\}$.

The proof system of $\mathrm{ML}^{=}(\Sigma)$ is defined by the following axiom schemas and rules of inference:

Logical Axioms:

($\top$) $\quad \vdash \top$

($\bot$) $\quad \bot \vdash$

(taut) $\quad A \vdash A$

Non-logical Axioms:

(eqv) $\quad \vdash \forall x{:}\, S\,(x = x) \wedge \forall x{:}\, S, y{:}\, S, z{:}\, S\,(x = y \wedge x = z \to y = z)$

(sub) $\quad t_1 = t_2, [x := t_1]A \vdash [x := t_2]A$

Rules of Inference:

($\neg L$) $\quad \dfrac{\Gamma \vdash \Delta, A}{\Gamma, \neg A \vdash \Delta}$ $\qquad$ ($\neg R$) $\quad \dfrac{\Gamma, A \vdash \Delta}{\Gamma \vdash \Delta, \neg A}$

($\wedge L$) $\quad \dfrac{\Gamma, A_i \vdash \Delta}{\Gamma, A_1 \wedge A_2 \vdash \Delta}$ for $i = 1, 2$ $\qquad$ ($\wedge R$) $\quad \dfrac{\Gamma \vdash \Delta, A_1 \quad \Gamma \vdash \Delta, A_2}{\Gamma \vdash \Delta, A_1 \wedge A_2}$

($\forall L$) $\quad \dfrac{\Gamma, [x := t]A \vdash \Delta}{\Gamma, \forall x{:}\, S\,(A) \vdash \Delta}$ $\qquad$ ($\forall R$) $\quad \dfrac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, \forall x{:}\, S\,(A)}$

(cut) $\quad \dfrac{\Gamma \vdash \Delta, A \quad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$ $\qquad$ (weak) $\quad \dfrac{\Gamma \vdash \Delta}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$

Restriction on the rule ($\forall R$): x not free in $\Gamma \cup \Delta$.

Multiple instances of axiom schema (eqv) for the same sort symbol are superfluous. In $\mathrm{ML}^{=}$, the axiom schema (sub) is equivalent to $\vdash \forall x{:}\, S, y{:}\, S\,(x = y \wedge A \to [x := y]A)$. The rule (weak) becomes a derived rule when the axiom schemas are replaced by weakened versions (axiom schemas $\vdash A$ are replaced by axiom schemas $\Gamma \vdash \Delta, A$, etc.).

The deduction theorem holds in $\mathrm{ML}^{=}$:

$$\frac{A_1 \vdash A_2}{\vdash A_1 \to A_2} \quad \text{is a derived rule.}$$

LPF does not have this property in common with $\mathrm{ML}^{=}$.

Derivations

A *derivation* (or *proof*) is a finitely branching tree with branches of finite length, where the nodes are labelled with sequents in such a way that the labels of terminal nodes are instances of axiom schemas and the label of any non-terminal node is obtained from the labels of its immediate descendants by applying an inference rule.

A sequent $\Gamma \vdash \Delta$ is *derivable* if there exists a derivation with its root labelled by $\Gamma \vdash \Delta$. We write $\mathrm{ML}^{=}(\Sigma)\colon \Gamma \vdash \Delta$ (and sometimes just $\Gamma \vdash \Delta$ without more ado) to indicate that $\Gamma \vdash \Delta$ is derivable.

4.4 Interpretation of $\mathrm{ML}^{=}(\Sigma)$

Structures

The structures used for interpretation of terms and formulae of $\mathrm{ML}^{=}(\Sigma)$ consist of an interpretation of every symbol in the signature Σ as well as an interpretation of the equality symbols.

A structure $\mathbf{A}$ with signature Σ consists of:

1. for every $S \in \mathrm{S}(\Sigma)$, a non-empty set $S^{\mathbf{A}}$;
2. for every $f \in \mathrm{F}(\Sigma)$, $f\colon S_1 \times \cdots \times S_n \to S_{n+1}$, a total map $f^{\mathbf{A}}\colon S_1^{\mathbf{A}} \times \cdots \times S_n^{\mathbf{A}} \to S_{n+1}^{\mathbf{A}}$;
3. for every $P \in \mathrm{P}(\Sigma)$, $P\colon S_1 \times \cdots \times S_n$, a total map $P^{\mathbf{A}}\colon S_1^{\mathbf{A}} \times \cdots \times S_n^{\mathbf{A}} \to \{\mathsf{T}, \mathsf{F}\}$;
4. for every $S \in \mathrm{S}(\Sigma)$, a total map $=_S^{\mathbf{A}}\colon S^{\mathbf{A}} \times S^{\mathbf{A}} \to \{\mathsf{T}, \mathsf{F}\}$ such that

$$\textit{for all } d, d' \in S^{\mathbf{A}},\ =_S^{\mathbf{A}}(d, d') = \begin{cases} \mathsf{T} & \text{if } d = d', \\ \mathsf{F} & \text{if } d \neq d'. \end{cases}$$

Instead of $w^{\mathbf{A}}$ we write w when it is clear from the context that the interpretation of symbol w in structure $\mathbf{A}$ is meant.

Assignment

An assignment in a structure $\mathbf{A}$ with signature Σ assigns to variables of sorts in Σ elements of the corresponding domains in $\mathbf{A}$. The interpretation of terms and formulae of $\mathrm{ML}^{=}(\Sigma)$ in $\mathbf{A}$ is given with respect to an assignment α in $\mathbf{A}$.

Let $\mathbf{A}$ be a structure with signature Σ. Then an *assignment* in $\mathbf{A}$ is a function α which maps variables of sort $S \in \mathrm{S}(\Sigma)$ to elements of $S^{\mathbf{A}}$.

For every assignment α in $\mathbf{A}$, variable symbol x of sort $S \in \mathrm{S}(\Sigma)$ and element $d \in S^{\mathbf{A}}$, we write $\alpha(x \to d)$ for the assignment α' such that $\alpha'(y) = \alpha(y)$ if $y \not\equiv x$ and $\alpha'(x) = d$.

Interpretation

The interpretation of terms is given by a function mapping term t of sort S, structure $\mathbf{A}$ and assignment α in $\mathbf{A}$ to the element of $S^{\mathbf{A}}$ that is the value of t in $\mathbf{A}$ under assignment α. Similarly, the interpretation of formulae is given by a function mapping formula A, structure $\mathbf{A}$ and assignment α in $\mathbf{A}$ to the element of $\{\mathsf{T}, \mathsf{F}\}$ that is the truth value of A in $\mathbf{A}$ under assignment α. We write $[\![t]\!]^{\mathbf{A}}_{\alpha}$ and $[\![A]\!]^{\mathbf{A}}_{\alpha}$ for these interpretations. The superscripts are omitted when it is clear from the context which structure is meant.

The interpretation functions for terms and formulae are inductively defined by:

$$
\begin{array}{lll}
[\![x]\!]_\alpha & = \alpha(x), & \\
[\![f(t_1,\ldots,t_n)]\!]_\alpha & = f([\![t_1]\!]_\alpha,\ldots,[\![t_n]\!]_\alpha), & \\
 & & \\
[\![\top]\!]_\alpha & = \mathsf{T}, & \\
[\![\bot]\!]_\alpha & = \mathsf{F}, & \\
[\![P(t_1,\ldots,t_n)]\!]_\alpha & = P([\![t_1]\!]_\alpha,\ldots,[\![t_n]\!]_\alpha), & \\
[\![t_1 = t_2]\!]_\alpha & = =_S([\![t_1]\!]_\alpha,[\![t_2]\!]_\alpha), & \\
[\![\neg\, A]\!]_\alpha & = \mathsf{T} & \text{if } [\![A]\!]_\alpha = \mathsf{F}, \\
 & \mathsf{F} & \text{if } [\![A]\!]_\alpha = \mathsf{T}, \\
[\![A_1 \wedge A_2]\!]_\alpha & = \mathsf{T} & \text{if } [\![A_1]\!]_\alpha = \mathsf{T} \textit{ and } [\![A_2]\!]_\alpha = \mathsf{T}, \\
 & \mathsf{F} & \text{if } [\![A_1]\!]_\alpha = \mathsf{F} \textit{ or } [\![A_2]\!]_\alpha = \mathsf{F}, \\
[\![\forall x\colon S\,(A)]\!]_\alpha & = \mathsf{T} & \text{if } \textit{for all } d \in S,\ [\![A]\!]_{\alpha(x\to d)} = \mathsf{T}, \\
 & \mathsf{F} & \text{if } \textit{for some } d \in S,\ [\![A]\!]_{\alpha(x\to d)} = \mathsf{F}\,.
\end{array}
$$

We write $\mathbf{A} \models A[\alpha]$ for $[\![A]\!]^{\mathbf{A}}_{\alpha} = \mathsf{T}$.

For finite sets Γ and Δ of formulae of $\mathrm{ML}^{=}(\Sigma)$, Δ is a *consequence* of Γ, written $\Gamma \models \Delta$, iff for all structures $\mathbf{A}$ with signature Σ, for all assignments α in $\mathbf{A}$, if $\mathbf{A} \models A[\alpha]$ for all $A \in \Gamma$ then $\mathbf{A} \models A'[\alpha]$ for some $A' \in \Delta$.

$\mathrm{ML}^{=}$ has the following soundness and completeness properties:

soundness: if $\Gamma \vdash \Delta$, then $\Gamma \models \Delta$; *completeness*: if $\Gamma \models \Delta$, then $\Gamma \vdash \Delta$.

LPF and MPL_ω have these properties in common with $\mathrm{ML}^{=}$.

5 Many-sorted Partial Infinitary Logic

MPL_ω is introduced by Koymans and Renardel de Lavalette in [KR89], where a mathematically precise presentation is given. In this section, language, proof system and interpretation of MPL_ω are presented by describing the differences with language, proof system and interpretation of $\mathrm{ML}^{=}$. These differences are consequences of the features that MPL_ω has in addition to those of $\mathrm{ML}^{=}$, viz. coverage of undefinedness, descriptions and countably infinite conjunctions. The assumptions and definitions about symbols and signatures for $\mathrm{ML}^{=}$ also apply to MPL_ω, except that additional standard symbols — needed for the coverage of undefinedness — are assumed.

5.1 Additional Standard Symbols for MPL_ω

For every $S \in SORT$, there is a standard function symbol $\uparrow_S : \rightarrow S$, called *undefined*. We write $\mathrm{SF}(\Sigma)$ for $\{\uparrow_S |\ S \in \mathrm{S}(\Sigma)\}$. It is assumed that $\uparrow_S \notin \mathcal{V}$ for all $S \in SORT$.

5.2 Language of $\mathrm{MPL}_\omega(\boldsymbol{\Sigma})$

The language of $\mathrm{MPL}_\omega(\Sigma)$ differs from the language of $\mathrm{ML}^=(\Sigma)$. The language of $\mathrm{ML}^=(\Sigma)$ requires adaptations to each of the additional features of MPL_ω.

Terms and Formulae

The terms and formulae of $\mathrm{MPL}_\omega(\Sigma)$ are simultaneously and inductively defined by the formation rules which are obtained from the formation rules for terms and formulae of $\mathrm{ML}^=(\Sigma)$ as follows:

a. replace formation rule 2 for terms by the following rule to adapt for undefinedness:

   2′. if $f \in \mathrm{F}(\Sigma) \cup \mathrm{SF}(\Sigma)$, $f : S_1 \times \cdots \times S_n \rightarrow S_{n+1}$ and $t_1, \ldots, t_n$ are terms of sorts $S_1, \ldots, S_n$, respectively, then $f(t_1, \ldots, t_n)$ is a term of sort S_{n+1};

b. add the following formation rule for terms to adapt for descriptions:

   3′. if A is a formula and x is a variable of sort S, $S \in \mathrm{S}(\Sigma)$, then $\iota x : S\,(A)$ is a term of sort S;

c. replace the formation rule 4 for formulae by the following rule to adapt for countably infinite conjunctions:

   4′. if $\langle A_n \rangle_{n<\omega} = \langle A_0, A_1, \ldots \rangle$ are formulae, then $\bigwedge_n A_n$ is a formula.

So we have countable conjunctions instead of binary conjunctions and descriptions as additional terms. Similar to the treatment of standard predicate symbols in the formation of (atomic) formulae, the standard function symbols are treated in the formation of terms in the same way as the function symbols from the signature Σ.

The terms and formulae of $\mathrm{MPL}_\omega(\Sigma)$, that can be constructed from the signature Σ according to the formation rules which are given above, include ill-formed terms and formulae. Only terms and formulae with a finite number of free variables are well-formed. In what follows, terms and formulae are always assumed to be well-formed.

A term or formula E of $\mathrm{MPL}_\omega(\Sigma)$ is *well-formed* iff *free*(E) is finite.

$\mathcal{T}_{MPL_\omega}(\Sigma)$ denotes the set of all well-formed terms of $\mathrm{MPL}_\omega(\Sigma)$.
$\mathcal{L}_{MPL_\omega}(\Sigma)$ denotes the set of all well-formed formulae of $\mathrm{MPL}_\omega(\Sigma)$.

Notational Conventions

The undefined symbols $(\uparrow_S)$ are used without subscript when this causes no ambiguity.

Countable disjunctions, binary conjunctions, definedness, and non-existential equality are defined as abbreviations:

$$\begin{array}{ll} \bigvee_n A_n & := \neg \bigwedge_n \neg A_n, \\ A_1 \wedge A_2 & := \bigwedge_n A'_n, \text{where } A'_0 = A_1 \text{ and } A'_n = A_2 \text{ for } 0 < n < \omega, \\ t{\downarrow_S} & := t =_S t \\ t_1 \simeq_S t_2 & := (t_1{\downarrow_S} \vee t_2{\downarrow_S}) \to t_1 =_S t_2. \end{array}$$

Binary disjunction, existential quantification, etc. are defined as abbreviations as for $\mathrm{ML}^{=}$.

The definedness symbols ($\downarrow_S$) and non-existential equality symbols ($\simeq_S$) are used without subscript when this causes no ambiguity.

In [KR89], definedness of terms is taken as a primitive notion: the definedness symbols are additional standard symbols. Here, definedness of t is defined as abbreviation of a formula. This difference is not essential, since $\vdash t{\downarrow} \leftrightarrow t = t$ holds.

5.3 Proof System of $\mathrm{MPL}_\omega(\Sigma)$

The proof system of $\mathrm{MPL}_\omega(\Sigma)$ differs from the proof system of $\mathrm{ML}^{=}(\Sigma)$. The proof system of $\mathrm{ML}^{=}(\Sigma)$ also requires adaptations to each of the additional features of MPL_ω.

The proof system of $\mathrm{MPL}_\omega(\Sigma)$ is defined by the axiom schemas and rules of inference which are obtained from the axiom schemas and rules of inference of $\mathrm{ML}^{=}(\Sigma)$ as follows:

a. add the following non-logical axiom schemas to adapt for undefinedness:

$(S{\uparrow})$ $\vdash \neg({\uparrow_S}{\downarrow})$

$(f{\downarrow})$ $\vdash f(t_1, \ldots, t_n){\downarrow} \to t_1{\downarrow} \wedge \ldots \wedge t_n{\downarrow}$

$(P{\downarrow})$ $\vdash P(t_1, \ldots, t_n) \to t_1{\downarrow} \wedge \ldots \wedge t_n{\downarrow}$

$(={\downarrow})$ $\vdash t_1 = t_2 \to t_1{\downarrow} \wedge t_2{\downarrow}$;

b. replace the rules of inference $(\forall L)$ and $(\forall R)$ by the following rules to adapt for undefinedness:

$$(\forall L) \quad \frac{\Gamma \vdash \Delta, t{\downarrow_S} \quad \Gamma, [x := t]A \vdash \Delta}{\Gamma, \forall x{:}\, S\,(A) \vdash \Delta} \quad (\forall R) \quad \frac{\Gamma, x{\downarrow_S} \vdash \Delta, A}{\Gamma \vdash \Delta, \forall x{:}\, S\,(A)} \ ;$$

c. add the following non-logical axiom schema to adapt for descriptions:

(ι) $\vdash \forall y{:}\, S\,(y = \iota x{:}\, S\,(A) \leftrightarrow \forall x{:}\, S\,(A \leftrightarrow x = y))$;

d. replace the rules of inference $(\wedge L)$ and $(\wedge R)$ by the following rules to adapt for countably infinite conjunctions:

$$(\textstyle\bigwedge L) \quad \frac{\Gamma, A_i \vdash \Delta}{\Gamma, \bigwedge_n A_n \vdash \Delta} \text{ for all } i \quad (\textstyle\bigwedge R) \quad \frac{\langle \Gamma \vdash \Delta, A_n \rangle_{n<\omega}}{\Gamma \vdash \Delta, \bigwedge_n A_n} \ .$$

Restriction on the axiom schema (ι): y not free in A. The restriction on the rule $(\forall R)$, viz. x not free in $\Gamma \cup \Delta$, remains.

Furthermore, the sets of formulae in a sequent $\Gamma \vdash \Delta$ may be countably infinite, but only finitely many different variables may occur free, i.e. $\mathit{free}(\Gamma)$ and $\mathit{free}(\Delta)$ are finite. Derivations may be infinitely (but countably) branching.

This sequent calculus resembles a Gentzen-type sequent calculus for infinitary classical first-order logic with equality. There are additional non-logical axiom schemas concerning definedness for the function and predicate symbols (including the undefined and equality symbols). There is also an axiom schema for descriptions. The slightly adapted rules for the universal quantifier are due to the treatment of free and bound variables: free variables may not denote but bound variables always do.

In [KR89], a proof system of $\mathrm{MPL}_\omega(\Sigma)$ is presented whose axiom schemas are weakened versions of equivalent axiom schemas for the axiom schemas of the proof system presented above (axiom schemas $\Gamma \vdash \Delta, A$ instead of axiom schemas $\vdash A$, etc.) and whose rules of inference do not include the weakening rule. However, the weakening rule is a derived rule. The rules of the proof system presented above include the weakening rule, because it is no longer a derived rule. On the other hand, when the weakening rule is included, the weakened versions of the axiom schemas become derivable. In other words, the proof systems are equivalent.

5.4 Interpretation of $\mathrm{MPL}_\omega(\Sigma)$

The interpretation of $\mathrm{MPL}_\omega(\Sigma)$ differs from the interpretation of $\mathrm{ML}^=(\Sigma)$. The interpretation of $\mathrm{ML}^=(\Sigma)$ requires adaptations to each of the additional features of MPL_ω. This includes the structures used for interpretation.

Structures

The structures used for MPL_ω differ from the structures used for $\mathrm{ML}^=$. The differences (for a structure $\mathbf{A}$ with signature Σ) are:

- for every $S \in \mathrm{S}(\Sigma)$, $S^{\mathbf{A}}$ is such that $\bot \in S^{\mathbf{A}}$;
- for every $S \in \mathrm{S}(\Sigma)$, $=_S^{\mathbf{A}}$ is such that
  $\mathit{for\ all}\ d, d' \in S^{\mathbf{A}},\ =_S^{\mathbf{A}}(d, d') = \begin{cases} \mathsf{T} & \text{if } d \neq \bot\ \mathit{and}\ d' \neq \bot\ \mathit{and}\ d = d', \\ \mathsf{F} & \text{if } d = \bot\ \mathit{or}\ d' = \bot\ \mathit{or}\ d \neq d'; \end{cases}$
- for every $f \in \mathrm{F}(\Sigma)$, $f\colon S_1 \times \cdots \times S_n \to S_{n+1}$, $f^{\mathbf{A}}$ is such that
  $\mathit{for\ all}\ d_1 \in S_1^{\mathbf{A}}, \ldots, d_n \in S_n^{\mathbf{A}},\ d_1 = \bot\ \mathit{or}\ \ldots\ \mathit{or}\ d_n = \bot \Rightarrow f^{\mathbf{A}}(d_1, \ldots, d_n) = \bot;$
- for every $P \in \mathrm{P}(\Sigma)$, $P\colon S_1 \times \cdots \times S_n$, $P^{\mathbf{A}}$ is such that
  $\mathit{for\ all}\ d_1 \in S_1^{\mathbf{A}}, \ldots, d_n \in S_n^{\mathbf{A}},\ d_1 = \bot\ \mathit{or}\ \ldots\ \mathit{or}\ d_n = \bot \Rightarrow P^{\mathbf{A}}(d_1, \ldots, d_n) = \mathsf{F}.$

In other words, the interpretations of sort symbols must be sets containing a special element $\bot$. When a term is non-denoting, $\bot$ is used as its interpretation. The interpretation of every symbol concerned is in accordance with the following treatment of non-denoting terms: atomic formulae that contain non-denoting terms are logically false.

Interpretation

The interpretation functions for terms and formulae are simultaneously and inductively defined by the interpretation rules obtained from the interpretation rules of $\mathrm{ML}^{=}$ as follows:

a. add the following interpretation rule to adapt for undefinedness:

$$[\![\uparrow_S]\!]_\alpha \;=\; \bot,$$

b. replace the interpretation rule for universal quantifications by the following rule to adapt for undefinedness:

$$\begin{array}{lcll} [\![\forall x{:}\, S\,(A)]\!]_\alpha & = & \mathsf{T} & \text{if } \textit{for all } d \in S - \{\bot\},\ [\![A]\!]_{\alpha(x\to d)} = \mathsf{T}, \\ & & \mathsf{F} & \text{if } \textit{for some } d \in S - \{\bot\},\ [\![A]\!]_{\alpha(x\to d)} = \mathsf{F}; \end{array}$$

c. add the following interpretation rule to adapt for descriptions:

$$\begin{array}{lcll} [\![\iota x{:}\, S\,(A)]\!]_\alpha & = & \textit{the unique } d \in S - \{\bot\} \textit{ such that } [\![A]\!]_{\alpha(x\to d)} = \mathsf{T} & \text{if it exists,} \\ & & \bot & \text{otherwise;} \end{array}$$

d. replace the interpretation rule for binary conjunctions by the following rule to adapt for countably infinite conjunctions:

$$\begin{array}{lcll} [\![\bigwedge_n A_n]\!]_\alpha & = & \mathsf{T} & \text{if } [\![A_n]\!]_\alpha = \mathsf{T} \textit{ for all } n < \omega, \\ & & \mathsf{F} & \text{if } [\![A_n]\!]_\alpha = \mathsf{F} \textit{ for some } n < \omega. \end{array}$$

So the interpretation of binary conjunctions is extended to countable conjunctions and the interpretation of descriptions is added. The interpretation of universal quantifications is slightly adapted in order to guarantee that bound variables always denote.

LPF needs predicate symbols to be interpreted as truth-valued functions and the interpretation of formulae to be given by a truth-valued function. This is already anticipated in the presentation of $\mathrm{ML}^{=}$ and MPL_ω. In this way, the essential differences between LPF and MPL_ω can be emphasized.

In [KR89], structures for $\mathrm{MPL}_\omega(\Sigma)$ are used whose interpretations of predicate symbols are n-ary relations. In the structures used here, these relations can be thought of as being replaced by the truth-valued functions that are their characteristic functions. This view agrees with the interpretation of formulae of the forms $P(t_1, \ldots, t_n)$ and $t_1 = t_2$ in such structures, which is given by the interpretation function described here. That is, the truth-valued functions used as interpretations of predicate symbols are consistently identified with the relations for which they are the characteristic functions. In other words, the truth of formulae is invariant under the replacement. Furthermore, the interpretation of formulae of $\mathrm{MPL}_\omega(\Sigma)$ is given in [KR89] by an interpretation relation. The interpretation function described here is the characteristic function of that relation. So it gives essentially the same interpretation of formulae.

5.5 Properties of MPL_ω

Some interesting properties of this logic, are presented below.

1. The deduction theorem holds in MPL_ω:

   $$\frac{A_1 \vdash A_2}{\vdash A_1 \to A_2} \quad \text{is a derived rule.}$$

2. MPL_ω is weaker than $\mathrm{ML}^{=}_{\omega}$, $\mathrm{ML}^{=}$ with binary conjunctions replaced by countable conjunctions:

   for every sequent $\Gamma \vdash \Delta$ of $\mathrm{ML}^{=}_{\omega}(\Sigma)$: $\quad \mathrm{MPL}'_\omega(\Sigma)\colon \Gamma \vdash \Delta \quad$ iff $\quad \mathrm{ML}^{=}_{\omega}(\Sigma)\colon \Gamma \vdash \Delta.$

   $\mathrm{MPL}'_\omega(\Sigma)$ is $\mathrm{MPL}_\omega(\Sigma)$ extended with the axiom schema $\vdash t{\downarrow}$ where t is an arbitrary term without occurrences of undefined symbols ($\uparrow_S$).

   That is, adding the assumption of definedness to MPL_ω yields a complete proof system for $\mathrm{ML}^{=}_{\omega}$. The proof is obvious.

   MPL_ω is even strictly weaker than $\mathrm{ML}^{=}_{\omega}$: $\vdash t{\downarrow}$, where t is an arbitrary term without occurrences of undefined symbols, is in general not derivable in MPL_ω.

3. MPL_ω can be reduced to $\mathrm{L}^{=}_{\omega}$, classical first-order logic with equality and countably infinite conjunctions:

   $\mathrm{MPL}_\omega(\Sigma)$: $\quad \vdash A \quad$ iff $\quad \mathrm{L}^{=}_{\omega}(\Sigma^*)\colon \mathrm{Ax}(\Sigma) \vdash A^*$

   for appropriate mappings $\bullet^*$ (for signatures and formulae) and an appropriate mapping Ax (mapping signatures of MPL_ω to sets of formulae of $\mathrm{L}^{=}_{\omega}$).

   A proof is given in [Ren89, Appendix C.1].

6 Logic for Partial Functions

LPF is introduced by Barringer, Cheng and Jones in [BCJ84], where it is presented in broad outline. A mathematically precise presentation is given by Cheng in [Che86]. In this section, language, proof system and interpretation of LPF are presented by describing the differences with language, proof system and interpretation of $\mathrm{ML}^{=}$. These differences are consequences of the feature that LPF has in addition to the features of $\mathrm{ML}^{=}$, viz. coverage of undefinedness with three truth values. The assumptions and definitions about symbols and signatures for $\mathrm{ML}^{=}$ also apply to LPF.

In [Che86], the connectives $*, \Delta, \neg, \vee$ and the quantifier $\exists$ are considered to be basic. In this section, $\top, \bot, *, \Delta, \neg, \wedge$ and $\forall$ are considered to be basic. In either case, all non-basic connectives and quantifiers can be defined with the basic ones. In [Che86], a one-sorted version of LPF is presented. In order to make a comparison with MPL_ω easier, a many-sorted version of LPF is presented in this section.

6.1 Language of LPF(Σ)

The language of LPF(Σ) differs from the language of ML$^=$(Σ). The language of ML$^=$ requires adaptations to the additional feature of LPF.

Terms and Formulae

The formation rules for terms of LPF(Σ) and formation rules for terms of ML$^=$(Σ) are the very same. The formulae of LPF(Σ) are inductively defined by the formation rules which are obtained from the formation rules for formulae of ML$^=$(Σ) as follows:

a. replace formation rule 1 by the following rule:

   $1''$. $\top$, $\bot$, and $*$ are formulae;

b. add the following formation rule:

   $6''$. if A is formula, then $\Delta\, A$ is a formula.

So we have $*$ and Δ as additional connectives (nullary and unary, respectively).

$\mathcal{T}_{LPF}(\Sigma)$ denotes the set of all terms of LPF(Σ).
$\mathcal{L}_{LPF}(\Sigma)$ denotes the set of all formulae of LPF(Σ).

Notational Conventions

Several kinds of definedness and equality are defined as abbreviations:

$$
\begin{array}{ll}
\delta\, A & := A \vee \neg\, A, \\
E_S(t) & := t =_S t, \\
t{\downarrow_S} & := \Delta\, E_S(t), \\
t_1 \stackrel{\exists}{=}_S t_2 & := t_1 =_S t_2 \wedge \Delta(E_S(t_1) \wedge E_S(t_2)), \\
t_1 ==_S t_2 & := (t_1{\downarrow_S} \vee t_2{\downarrow_S}) \rightarrow t_1 \stackrel{\exists}{=}_S t_2.
\end{array}
$$

Binary disjunction, existential quantification, etc. are defined as abbreviations as for ML$^=$.

The definedness and equality symbols E_S, $\downarrow_S$, $\stackrel{\exists}{=}_S$ and $==_S$ are used without subscript when this causes no ambiguity.

Both $E(t)$ and $t\downarrow$ mean that t is denoting. If t is non-denoting, then $E(t)$ is neither-true-nor-false but $t\downarrow$ is false as in MPL$_\omega$. If either t_1 or t_2 is non-denoting, then $t_1 \stackrel{\exists}{=} t_2$ is either true or false and $t_1 == t_2$ is false. $\stackrel{\exists}{=}$ is existential equality and $==$ is strong equality (see Section 3.1). $==$ is essentially the same as $\simeq$ in MPL$_\omega$.

6.2 Proof System of LPF(Σ)

The proof system of LPF(Σ) differs from the proof system of ML$^=$(Σ). The proof system of ML$^=$ also requires adaptations to the additional feature of LPF. The differences are great.

The proof system of $\mathrm{LPF}(\Sigma)$ is defined by the axiom schemas and rules of inference which are obtained from the axiom schemas and rules of inference of $\mathrm{ML}^{=}(\Sigma)$ as follows:

a. add the following logical axiom schemas:

$(*)$ $\quad * \vdash$

$(\neg *)$ $\quad \neg * \vdash$

(contr) $\quad A, \neg A \vdash$;

b. add the following non-logical axiom schemas:

(var) $\quad \vdash E(x)$

$(=E)$ $\quad t_1 = t_2 \vdash E(t_1) \wedge E(t_2)$

$(\neg =E)$ $\quad \neg(t_1 = t_2) \vdash E(t_1) \wedge E(t_2)$

(comp) $\quad E(t_1), E(t_2) \vdash t_1 = t_2, \neg(t_1 = t_2)$;

c. add the following rules of inference:

$$(\Delta L) \quad \frac{\Gamma, \neg A \vdash \Delta \quad \Gamma, A \vdash \Delta}{\Gamma, \Delta\, A \vdash \Delta} \qquad (\Delta R) \quad \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, \Delta\, A} \quad \frac{\Gamma \vdash \Delta, \neg A}{\Gamma \vdash \Delta, \Delta\, A}$$

$$(\neg \Delta L) \quad \frac{\Gamma \vdash \Delta, A}{\Gamma, \neg \Delta\, A \vdash \Delta} \quad \frac{\Gamma \vdash \Delta, \neg A}{\Gamma, \neg \Delta\, A \vdash \Delta} \qquad (\neg \Delta R) \quad \frac{\Gamma, \neg A \vdash \Delta \quad \Gamma, A \vdash \Delta}{\Gamma \vdash \Delta, \neg \Delta\, A}$$;

d. replace all rules of inference except (cut) and (weak) by the following rules:

$$(\neg \neg L) \quad \frac{\Gamma, A \vdash \Delta}{\Gamma, \neg \neg A \vdash \Delta} \qquad (\neg \neg R) \quad \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, \neg \neg A}$$

$$(\wedge L) \quad \frac{\Gamma, A_i \vdash \Delta}{\Gamma, A_1 \wedge A_2 \vdash \Delta} \text{ for } i = 1, 2 \qquad (\wedge R) \quad \frac{\Gamma \vdash \Delta, A_1 \quad \Gamma \vdash \Delta, A_2}{\Gamma \vdash \Delta, A_1 \wedge A_2}$$

$$(\neg \wedge L) \quad \frac{\Gamma, \neg A_1 \vdash \Delta \quad \Gamma, \neg A_2 \vdash \Delta}{\Gamma, \neg(A_1 \wedge A_2) \vdash \Delta} \qquad (\neg \wedge R) \quad \frac{\Gamma \vdash \Delta, \neg A_i}{\Gamma \vdash \Delta, \neg(A_1 \wedge A_2)} \text{ for } i = 1, 2$$

$$(\forall L) \quad \frac{\Gamma \vdash \Delta, E_S(t) \quad \Gamma, [x := t]A \vdash \Delta}{\Gamma, \forall x\colon S\,(A) \vdash \Delta} \qquad (\forall R) \quad \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, \forall x\colon S\,(A)}$$

$$(\neg \forall L) \quad \frac{\Gamma, \neg A \vdash \Delta}{\Gamma, \neg \forall x\colon S\,(A) \vdash \Delta} \qquad (\neg \forall R) \quad \frac{\Gamma \vdash \Delta, E_S(t) \quad \Gamma \vdash \Delta, \neg[x := t]A}{\Gamma \vdash \Delta, \neg \forall x\colon S\,(A)}$$.

The restriction on the rule $(\forall R)$, viz. x not free in $\Gamma \cup \Delta$, remains and also applies to the rule $(\neg \forall L)$.

This sequent calculus does not resemble a Gentzen-type sequent calculus for classical first-order logic. Adding the axiom schema $\vdash A, \neg A$, corresponding to the law of the excluded middle, would make it a complete proof system for $\mathrm{ML}^{=}$. Because this law does not hold, the rules $(\neg L)$ and $(\neg R)$ are replaced by the axiom schema (contr) and the special rules

concerning negation for each of the basic connectives and the universal quantifier. The additional axiom schemas for equality are due to the extension of equality to the three-valued case. The axiom schema $(= E)$ is similar to the axiom schema $(=\downarrow)$ of MPL_ω. Axiom schemas and rules concerning the connectives without a classical counterpart are added. The usual rules for the universal quantifier are slightly adapted, because bound variables always denote. The adapted rules also differ from the corresponding rules of MPL_ω, because, unlike there, free variables always denote in LPF. The axiom schema (var) expresses that free variables always denote. Note that the rules $(\wedge L)$ and $(\wedge R)$ are not changed.

In [Che86], a proof system of LPF is presented with rules concerning disjunction and existential quantification instead of rules concerning conjunction and universal quantification as in the proof system presented above, since different connectives and quantifiers are taken as basic there. In either case, the other rules are derived rules. The axiom schemas of the proof system presented above also include the additional axiom schemas $(\top)$ and $(\bot)$. However, these axiom schemas are derivable. In other words, the proof systems are equivalent.

6.3 Interpretation of LPF(Σ)

The interpretation of $\mathrm{LPF}(\Sigma)$ differs from the interpretation of $\mathrm{ML}^=(\Sigma)$. The interpretation of $\mathrm{ML}^=$ requires adaptations to the additional feature of LPF. This includes the structures used for interpretation.

Structures

The structures used for LPF differ from the structures used for $\mathrm{ML}^=$. The differences (for a structure $\mathbf{A}$ with signature Σ) are:

- for every $S \in \mathrm{S}(\Sigma)$, $S^{\mathbf{A}}$ is such that $\bot \in S^{\mathbf{A}}$ and $S^{\mathbf{A}} - \{\bot\}$ is non-empty;
- for every $P \in \mathrm{P}(\Sigma) \cup \mathrm{SP}(\Sigma)$, $P^{\mathbf{A}}$ maps to $\{\mathsf{T}, \mathsf{F}, \mathsf{N}\}$;
- for every $S \in \mathrm{S}(\Sigma)$, $=_S^{\mathbf{A}}$ is such that

$$\textit{for all } d, d' \in S^{\mathbf{A}},\ =_S^{\mathbf{A}}(d, d') = \begin{cases} \mathsf{T} & \text{if } d \neq \bot \textit{ and } d' \neq \bot \textit{ and } d = d', \\ \mathsf{F} & \text{if } d \neq \bot \textit{ and } d' \neq \bot \textit{ and } d \neq d', \\ \mathsf{N} & \text{otherwise.} \end{cases}$$

In other words, the interpretations of sort symbols are extended with a special element $\bot$ and the domain of truth values is extended with another special element N. When a term is non-denoting, $\bot$ is used as its interpretation. Analogously, when a formula is neither T (*true*) nor F (*false*), N is used as its interpretation. We can think of N as corresponding to the classification *neither-true-nor-false*. The interpretation of every symbol concerned is in accordance with the treatment of non-denoting terms where equations $t_1 = t_2$ are logically neither-true-nor-false when t_1 or t_2 is non-denoting.

In contrast with MPL_ω, the interpretation of sort symbols may not be sets that only contain $\bot$. Functions and predicates may be non-strict. However, these are minor differences. The main difference is that predicates may yield the truth value N.

Assignment

The definition of assignment is also changed. Let $\mathbf{A}$ be a structure with signature Σ. Then an *assignment* in $\mathbf{A}$ is a function α which maps variables of sort $S \in \mathrm{S}(\Sigma)$ to elements of $S^{\mathbf{A}} - \{\bot\}$. This means that variables are never mapped to $\bot$. This restriction is in accordance with the treatment of free and bound variables: both free and bound variables always denote. Without the redefinition of assignment, a redefinition of consequence would have been necessary.

Interpretation

The interpretation function for terms and formulae are inductively defined by the interpretation rules obtained from the interpretation rules of $\mathrm{ML}^{=}$ as follows:

a. add the following interpretation rules:

$$\begin{array}{lcll} [\![*]\!]_\alpha, & = & \mathsf{N}, & \\ [\![\Delta\, A]\!]_\alpha & = & \mathsf{T} & \text{if } [\![A]\!]_\alpha = \mathsf{T} \textit{ or } [\![A]\!]_\alpha = \mathsf{F}, \\ & & \mathsf{F} & \text{otherwise;} \end{array}$$

b. replace the interpretation rules for negations, binary conjunctions, and universal quantifications by the following rules:

$$\begin{array}{lcll} [\![\neg\, A]\!]_\alpha & = & \mathsf{T} & \text{if } [\![A]\!]_\alpha = \mathsf{F}, \\ & & \mathsf{F} & \text{if } [\![A]\!]_\alpha = \mathsf{T}, \\ & & \mathsf{N} & \text{otherwise,} \\ [\![A_1 \wedge A_2]\!]_\alpha & = & \mathsf{T} & \text{if } [\![A_1]\!]_\alpha = \mathsf{T} \textit{ and } [\![A_2]\!]_\alpha = \mathsf{T}, \\ & & \mathsf{F} & \text{if } [\![A_1]\!]_\alpha = \mathsf{F} \textit{ or } [\![A_2]\!]_\alpha = \mathsf{F}, \\ & & \mathsf{N} & \text{otherwise,} \\ [\![\forall x\colon S\,(A)]\!]_\alpha & = & \mathsf{T} & \text{if } \textit{for all } d \in S - \{\bot\},\ [\![A]\!]_{\alpha(x \to d)} = \mathsf{T}, \\ & & \mathsf{F} & \text{if } \textit{for some } d \in S - \{\bot\},\ [\![A]\!]_{\alpha(x \to d)} = \mathsf{F}, \\ & & \mathsf{N} & \text{otherwise.} \end{array}$$

So the classical T/F-conditions for connectives and quantifiers are adopted; only the assumption is given up that all formulae under all assignment have to be either true or false. The interpretation of the new connectives is added.

6.4 Properties of LPF

Some properties, that are relevant to the use of this logic, are presented here.

1. LPF has the following expressiveness property:

   > All functions of type $\{\mathsf{T}, \mathsf{F}, \mathsf{N}\}^n \to \{\mathsf{T}, \mathsf{F}, \mathsf{N}\}$ are definable by the basic connectives of LPF.

   A proof is given in [Che86, Section 3.1].

2. A weak version of the deduction theorem holds in LPF:

$$\frac{A_1 \vdash A_2 \quad \vdash \delta(A_1)}{\vdash A_1 \to A_2} \text{ is a derived rule.}$$

A proof is given in [Jon90, Section 1.3].

3. LPF is weaker than $\mathrm{ML}^{=}$:

   for every sequent $\Gamma \vdash \Delta$ of $\mathrm{ML}^{=}(\Sigma)$: $\quad \mathrm{LPF}'(\Sigma): \Gamma \vdash \Delta \quad$ iff $\quad \mathrm{ML}^{=}(\Sigma): \Gamma \vdash \Delta$.

   $\mathrm{LPF}'(\Sigma)$ is $\mathrm{LPF}(\Sigma)$ extended with the axiom schema $\vdash A, \neg A$.

   That is, adding the law of the excluded middle to LPF yields a complete proof system for $\mathrm{ML}^{=}$. A proof for propositional LPF without equality (in the one-sorted case) is given in [Avr88, Section 4.1]. That proof extends directly to the general case.

   LPF is even strictly weaker than $\mathrm{ML}^{=}$: $\vdash A, \neg A$ is in general not derivable in LPF.

7 Reducing LPF to MPL_ω

The relationship between LPF and $\mathrm{ML}^{=}$ is clearly characterized by property (3) of Section 6.4. The relationship between MPL_ω and $\mathrm{ML}^{=}_\omega$ is similarly characterized by property (2) of Section 5.5. Besides, the relationship between MPL_ω and the one-sorted fragment of $\mathrm{ML}^{=}_\omega$, i.e. $\mathrm{L}^{=}_\omega$, is characterized by property (3) of Section 5.5. That property is a reducibility result.

In this section, the relationship between LPF and MPL_ω will also be characterized by a reducibility result. The mappings concerned provide a uniform embedding of LPF into MPL_ω.

7.1 Translation

The translation given below simplifies the translation of the logical expressions of VVSL (which are roughly the formulae of LPF) in [Mid90].

We write:

$$\begin{array}{llll} \mathcal{T}_{LPF} & \text{for } \bigcup\{\mathcal{T}_{LPF}(\Sigma) \mid \Sigma \in SIGN\}, & \mathcal{L}_{LPF} & \text{for } \bigcup\{\mathcal{L}_{LPF}(\Sigma) \mid \Sigma \in SIGN\}, \\ \mathcal{T}_{MPL_\omega} & \text{for } \bigcup\{\mathcal{T}_{MPL_\omega}(\Sigma) \mid \Sigma \in SIGN\}, & \mathcal{L}_{MPL_\omega} & \text{for } \bigcup\{\mathcal{L}_{MPL_\omega}(\Sigma) \mid \Sigma \in SIGN\}. \end{array}$$

For the translation of formulae two mappings are used:

$$\langle\!\langle \bullet \rangle\!\rangle: \mathcal{L}_{LPF} \to \mathcal{T}_{MPL_\omega}, \quad \langle\!\langle \bullet \rangle\!\rangle: \mathcal{T}_{LPF} \to \mathcal{T}_{MPL_\omega}.$$

For a formula A of LPF, the formula $\langle\!\langle A \rangle\!\rangle = tt$ is the translation of A to MPL_ω. Intuitively, $\langle\!\langle A \rangle\!\rangle = tt$ is a formula of MPL_ω stating that the formula A of LPF is true in LPF. Likewise, $\langle\!\langle A \rangle\!\rangle = ff$ is a formula of MPL_ω stating that the formula A of LPF is false in LPF. In case both $\langle\!\langle A \rangle\!\rangle = tt$ and $\langle\!\langle A \rangle\!\rangle = ff$ are false in MPL_ω, A is neither-true-nor-false in LPF.

The syntactic variables that are used in the definition of these mappings, range over syntactic objects as follows (subscripts and primes are not shown):

S ranges over $SORT$, f ranges over $FUNC$, P ranges over $PRED$,
x ranges over VAR, t ranges over $\mathcal{T}_{LPF}$, A ranges over $\mathcal{L}_{LPF}$.

It is assumed that $\mathbb{B} \in SORT$, $tt, ff \in FUNC$, $tt, ff : \to \mathbb{B}$, and $b \in VAR$, b of sort $\mathbb{B}$.

We assume a total mapping from $\mathcal{V}$ to $\mathcal{V}$; for each $w \in \mathcal{V}$, we write w for the symbol to which w is mapped. Furthermore, the mapping is assumed to be injective and such that

each sort symbol S is mapped to a sort symbol S,
each function symbol $f : S_1 \times \cdots \times S_n \to S_{n+1}$ is mapped to a function symbol
$f : S_1 \times \cdots \times S_n \to S_{n+1}$,
each predicate symbol $P : S_1 \times \cdots \times S_n$ is mapped to a function symbol
$P : S_1 \times \cdots \times S_n \to \mathbb{B}$,
each variable symbol x of sort S is mapped to a variable symbol x of sort S,
the set $\{\mathbb{B}, tt, ff, b\}$ and its image are disjoint.

The translation mappings are inductively defined by:

$$
\begin{array}{ll}
\langle\!| x |\!\rangle & = x, \\
\langle\!| f(t_1, \ldots, t_n) |\!\rangle & = f(\langle\!| t_1 |\!\rangle, \ldots, \langle\!| t_n |\!\rangle), \\
 & \\
\langle\!| * |\!\rangle & = \uparrow, \\
\langle\!| P(t_1, \ldots, t_n) |\!\rangle & = P(\langle\!| t_1 |\!\rangle, \ldots, \langle\!| t_n |\!\rangle), \\
\langle\!| t_1 = t_2 |\!\rangle & = \iota\, b : \mathbb{B}(\ \langle\!| t_1 |\!\rangle\downarrow \wedge \langle\!| t_2 |\!\rangle\downarrow \to (\langle\!| t_1 |\!\rangle = \langle\!| t_2 |\!\rangle \ \leftrightarrow\ b = tt)\ ), \\
\langle\!| \Delta\, A |\!\rangle & = \iota\, b : \mathbb{B}(\ \langle\!| A |\!\rangle = tt \vee \langle\!| A |\!\rangle = ff \ \leftrightarrow\ b = tt\ ), \\
\langle\!| \neg A |\!\rangle & = \iota\, b : \mathbb{B}(\ (\langle\!| A |\!\rangle = tt \ \leftrightarrow\ b = ff) \wedge \\
 & \qquad (\langle\!| A |\!\rangle = ff \ \leftrightarrow\ b = tt)\ ), \\
\langle\!| A_1 \wedge A_2 |\!\rangle & = \iota\, b : \mathbb{B}(\ (\langle\!| A_1 |\!\rangle = tt \wedge \langle\!| A_2 |\!\rangle = tt \ \leftrightarrow\ b = tt) \wedge \\
 & \qquad (\langle\!| A_1 |\!\rangle = ff \vee \langle\!| A_2 |\!\rangle = ff \ \leftrightarrow\ b = ff)\ ), \\
\langle\!| \forall x : S\,(A) |\!\rangle & = \iota\, b : \mathbb{B}(\ (\forall x : S(\langle\!| A |\!\rangle = tt) \ \leftrightarrow\ b = tt) \wedge \\
 & \qquad (\exists x : S(\langle\!| A |\!\rangle = ff) \ \leftrightarrow\ b = ff)\ ).
\end{array}
$$

These translation rules strongly resemble the interpretation rules of LPF that are given in Section 6.3. Only the translation rule for equations does not resemble the corresponding interpretation rule. The interpretation rule concerned gives explicitly a truth-condition and a falsehood-condition. The corresponding translation rule, unlike the other ones, could be simplified by leaving this style.

A translation for sequents of LPF(Σ) can also be devised:

$$\langle\!| \Gamma \vdash \Delta |\!\rangle \ :=\ \mathrm{Ax}(\Sigma) \cup \mathrm{Bax} \cup \{\langle\!| A |\!\rangle = tt \mid A \in \Gamma\} \vdash \{\langle\!| A' |\!\rangle = tt \mid A' \in \Delta\},$$

$$
\begin{array}{lll}
\text{where} & \mathrm{Ax}(\Sigma) & = \{\exists y_S : S(y_S\downarrow) \mid S \in \mathrm{S}(\Sigma)\}, \\
 & \mathrm{Bax} & = \{\forall\, b : \mathbb{B}(b = tt \vee b = ff), \neg(tt = ff)\}.
\end{array}
$$

Here, it is assumed that y_S is a variable symbol of sort S for all $S \in SORT$.

7.2 Reducibility

Roughly speaking, LPF can be reduced to MPL_ω in the sense that what can be proved in LPF remains the same after translation. Unfortunately, MPL_ω only deals with strict functions and predicates whilst LPF deals with non-strict functions and predicates as well. Therefore "MPL_ω without strictness axioms" and "LPF with strictness axioms" are

introduced.

MPL_ω^- is MPL_ω with its proof system restricted by removing the axiom schema $(f\!\downarrow)$ and the axiom schema $(P\!\downarrow)$.

LPF^+ is LPF with its proof system extended by adding the axiom schema $(f\!\downarrow)$ and the following axiom schema:

$$(P\!\downarrow') \quad \vdash \Delta\, P(t_1, \ldots, t_n) \to t_1\!\downarrow \wedge \ldots \wedge t_n\!\downarrow \ .$$

Structures for LPF with strict maps as interpretations of function and predicate symbols are called strict structures.

Theorem

1. LPF^+ can be reduced to MPL_ω, i.e.:

   $$\mathrm{LPF}^+(\Sigma)\colon\ \Gamma \vdash \Delta \quad \text{iff} \quad \mathrm{MPL}_\omega(\Sigma \cup \{\mathbb{B}, \mathit{tt}, \mathit{ff}\})\colon\ \langle\!|\Gamma \vdash \Delta|\!\rangle;$$

2. LPF can be reduced to MPL_ω^-, i.e.:

   $$\mathrm{LPF}(\Sigma)\colon\ \Gamma \vdash \Delta \quad \text{iff} \quad \mathrm{MPL}_\omega^-(\Sigma \cup \{\mathbb{B}, \mathit{tt}, \mathit{ff}\})\colon\ \langle\!|\Gamma \vdash \Delta|\!\rangle.$$

Proof:
1. $\Rightarrow$ is proved by induction over the length of a derivation of $\Gamma \vdash \Delta$. For $\Leftarrow$, it suffices to show that for some strict structure $\mathbf{A}$ of LPF with signature Σ that is a counter-model for $\Gamma \vdash \Delta$, there exists a structure $\mathbf{A}^*$ of MPL_ω with signature $\Sigma \cup \{\mathbb{B}, \mathit{tt}, \mathit{ff}\}$ that is a counter-model for $\langle\!|\Gamma \vdash \Delta|\!\rangle$.
2. is proved similarly. □

It is assumed that the translation of sequents is extended to inference rules in the obvious way.

Corollary
The translation of the inference rules of LPF are derived rules in MPL_ω.

8 Recursively Defined Functions

In the previous section, LPF is embedded into MPL_ω. Besides, recursive function definitions can be represented in MPL_ω. All that allows the rules that are used to reason about recursively defined functions in LPF to become derived rules of MPL_ω.

8.1 LPF and Recursive Function Definitions

The logic LPF is used in VDM [Jon86, Jon90, JS90] to reason about recursively defined functions. The feature for recursive function definitions in VDM is made precise below by defining a conservative extension of LPF, referred to by LPF^R.

The following additional formation rule for terms is required:

3″. if A is a formula and t_1 and t_2 are terms of sort S, then $\mathsf{if}\ A\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$ is a term of sort S, for any $S \in \mathrm{S}(\Sigma)$.

Terms of this form are called *conditionals*. In [BCJ84, Jon86], conditionals are also regarded as terms of an extension of LPF.

The following additional formation rule for formulae is required:

7″. if $f \in \mathrm{F}(\Sigma)$, $f\colon S_1 \times \ldots \times S_n \to S_{n+1}$, $x_1, \ldots, x_n$ are distinct variables of sorts $S_1, \ldots, S_n$, respectively, and t is a term of sort S_{n+1} with $\mathit{free}(t) \subseteq \{x_1, \ldots, x_n\}$, then $f(x_1, \ldots, x_n) \triangleq t$ is a formula.

Formulae of this form are called *recursive function definitions*.

A recursive function definition $f(x_1, \ldots, x_n) \triangleq t$ defines f directly in terms of a defining term t in which the function being defined may be recursively used.

The following additional inference rules are needed:

$$\text{(if-E)} \qquad \frac{\vdash A \quad \vdash t == \mathsf{if}\ A\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2}{\vdash t == t_1} \qquad \frac{\vdash \neg A \quad \vdash t == \mathsf{if}\ A\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2}{\vdash t == t_2}$$

$$(\triangleq{\to}{=}{=}) \qquad \frac{}{f(x_1, \ldots, x_n) \triangleq t \vdash \forall x_1\colon S_1, \ldots, x_n\colon S_n\, (f(x_1, \ldots, x_n) == t)}\ .$$

Restriction on the rule $(\triangleq{\to}{=}{=})$: $f\colon S_1 \times \cdots \times S_n \to S_{n+1}$.

These rules for conditionals and recursive function definitions are essentially the same as those postulated in [BCJ84]. However, recursive function definitions are not treated as formulae there.

The following substitution rules for reasoning about recursively defined functions are introduced by Jones in [Jon86]:[1]

$$\text{($\triangleq$-subs)} \quad \frac{\vdash t = t \quad \vdash [y := [x := t]t']A}{\vdash [y := f(t)]A} \qquad \text{given a recursive function definition } f(x) \triangleq t'$$

$$\text{(if-subs)} \quad \frac{\dfrac{\vdash A \quad \vdash E(t) \quad \vdash [y := [x := t]t_1]A'}{\vdash [y := f(t)]A'} \qquad \dfrac{\vdash \neg A \quad \vdash E(t) \quad \vdash [y := [x := t]t_2]A'}{\vdash [y := f(t)]A'}}{} \qquad \text{given a recursive function definition } f(x) \triangleq \mathsf{if}\ A\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$$

These rules can be regarded as derived rules of LPF^R. They permit proofs to avoid use of and reasoning about strong equality, provided that there are no nested conditionals involved.

In [Jon90], Jones informally explains how a recursive definition of a partial function can be rendered into inference rules. The inference rules concerned resemble the appropriate rules of an inductive definition of the function (for partial functions, such rules usually need to be of a particular form). Given the recursive definition, the inference rules can also be regarded as derived rules of LPF^R.

In structures for LPF, a partial function is modelled by a total map whose argument domains and result domain contain a special element $\bot$. An argument tuple is mapped to $\bot$ if the function concerned is undefined for that argument tuple. This suggests the following definition, which is used in the additional interpretation rules given below.

[1] Only the simple case of unary functions is shown; extension to the general case of n-ary functions is straightforward.

For total maps F and G, $F, G: \mathcal{S}_1 \times \cdots \times \mathcal{S}_n \to \mathcal{S}_{n+1}$ $(\bot \in \mathcal{S}_1, \ldots, \mathcal{S}_{n+1})$, F is called *less defined than* G iff

$$\textit{for all } d_1 \in \mathcal{S}_1, \ldots, d_n \in \mathcal{S}_n, F(d_1, \ldots, d_n) \neq \bot \Rightarrow F(d_1, \ldots, d_n) = G(d_1, \ldots, d_n).$$

The following additional interpretation rules are required:

$$\begin{array}{lcll}
[\![\mathsf{if}\ A\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2]\!]^{\mathbf{A}}_{\alpha} & = & [\![t_1]\!]^{\mathbf{A}}_{\alpha} & \text{if } [\![A]\!]^{\mathbf{A}}_{\alpha} = \mathsf{T}, \\
 & & [\![t_2]\!]^{\mathbf{A}}_{\alpha} & \text{if } [\![A]\!]^{\mathbf{A}}_{\alpha} = \mathsf{F}, \\
 & & \bot & \text{otherwise}, \\
[\![f(x_1, \ldots, x_n) \triangleq t]\!]^{\mathbf{A}}_{\alpha} & = & \mathsf{T} & \text{if } f^{\mathbf{A}} \text{ is the least defined } F: S_1^{\mathbf{A}} \times \cdots \times S_n^{\mathbf{A}} \to S_{n+1}^{\mathbf{A}} \text{ such} \\
 & & & \text{that } \textit{for all } d_1 \in S_1^{\mathbf{A}} - \{\bot\}, \ldots, d_{n+1} \in S_{n+1}^{\mathbf{A}} - \{\bot\}, \\
 & & & [\![t]\!]^{\mathbf{A}'}_{\alpha(x_1 \to d_1) \cdots (x_n \to d_n)} = d_{n+1} \Rightarrow F(d_1, \ldots, d_n) = d_{n+1}, \\
 & & \mathsf{F} & \text{otherwise};
\end{array}$$

assuming that $f: S_1 \times \cdots \times S_n \to S_{n+1}$ and where $\mathbf{A}'$ is the structure with signature Σ such that $w^{\mathbf{A}'} = w^{\mathbf{A}}$ if $w \not\equiv f$ and $f^{\mathbf{A}'} = F$ $(w \in \Sigma \cup \mathrm{SP}(\Sigma))$.

It can easily be checked that the least defined F satisfying a condition of the given form always exists.

The extended interpretation is sound with respect to the extended proof system. However, the extended proof system is not complete with respect to the extended interpretation. Because the inference rules do not cover the "leastness" of recursively defined functions, they are not sufficient to prove all properties that hold for those functions. Their undefinedness properties cannot be proved.

8.2 MPL_ω and Recursive Function Definitions

Just like terms of LPF, conditionals can be mapped to terms of MPL_ω. That is, the definition of the translation mappings for LPF can be extended to LPF with conditionals by the addition of the following translation rule for conditionals:

$$\langle\!|\mathsf{if}\ A\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2|\!\rangle = \iota y: S((\langle\!|A|\!\rangle = tt \to y = \langle\!|t_1|\!\rangle) \wedge (\langle\!|A|\!\rangle = ff \to y = \langle\!|t_2|\!\rangle)),$$

assuming that $\mathsf{if}\ A\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$ is a term of sort S and where y is a variable symbol of sort S not free in $\langle\!|A|\!\rangle$, $\langle\!|t_1|\!\rangle$, and $\langle\!|t_2|\!\rangle$.

Just like formulae of LPF, recursive function definitions can be mapped to terms of MPL_ω. That is, the definition of the translation mappings for LPF with conditionals can further be extended to LPF^R by the addition of the following translation rule for recursive function definitions:

$$\begin{array}{l}
\langle\!|f(x_1, \ldots, x_n) \triangleq t|\!\rangle = \\
\quad \iota\, b: \mathbb{B}(f \stackrel{\mathrm{I}}{:=} \forall x_1: S_1, \ldots, x_n: S_n, y: S_{n+1}(\langle\!|t|\!\rangle = y \to f(x_1, \ldots, x_n) = y) \leftrightarrow b = tt),
\end{array}$$

assuming that $f: S_1 \times \cdots \times S_n \to S_{n+1}$ and where y is a variable symbol of sort S_{n+1} not free in $\langle\!|t|\!\rangle$.

For a recursive function definition $f(x_1, \ldots, x_n) \triangleq t$, the formula $\langle\!|f(x_1, \ldots, x_n) \triangleq t|\!\rangle = tt$ is its translation to MPL_ω. Intuitively, $\langle\!|f(x_1, \ldots, x_n) \triangleq t|\!\rangle = tt$ is a formula of MPL_ω stating that f is the least defined function such that the value of that function for arguments $x_1, \ldots, x_n$ is the value that is yielded by evaluation of t when f denotes that function.

This translation simplifies the translation of the explicit function definitions of VVSL in [Mid90].

For function symbol $f \in \mathrm{F}(\Sigma)$, $f\colon S_1 \times \cdots \times S_n \to S_{n+1}$, and formula A of $\mathrm{MPL}_\omega(\Sigma)$, $f \stackrel{\mathrm{I}}{:=} A$ is defined as an abbreviation of a formula of MPL_ω by

$$f \stackrel{\mathrm{I}}{:=} A \;:=\; \forall x_1\colon S_1, \ldots, x_n\colon S_n\, (f(x_1, \ldots, x_n) \simeq (\delta f.A)(x_1, \ldots, x_n)).$$

$(\delta f.A)(t_1, \ldots, t_n)$ abbreviates a term of $\mathrm{MPL}_\omega(\Sigma)$. It can be regarded as function application, with $(\delta f.A)$ denoting the least defined function f such that A (under the conditions that A is admissible for f and A is functionality preserving for f). This notation and the accompanying conditions are defined in [Mid90, Section 4.7]. It can be defined as abbreviations because MPL_ω is a logic with countably infinite conjunctions.

The meaning of $\delta f.A$ is reflected in the following derived rules, which are proved in [Ren89, Appendix D.4]:

$$\frac{\vdash \mathrm{Adm}(f, A)}{\vdash \mathrm{Func}(f, A) \to [f := \delta f.A]A} \qquad \frac{\vdash \mathrm{Adm}(f, A)}{\vdash \mathrm{Func}(f, A) \to (A \to \delta f.A \subseteq f)}\,.$$

$\mathrm{Adm}(f, A)$ and $\mathrm{Func}(f, A)$ abbreviate formulae of MPL_ω stating the above-mentioned conditions concerning admissibility and preservation of functionality. $\delta f.A \subseteq f$ abbreviates a formula stating that $\delta f.A$ is less (or equally) defined than f.

The rules ($\stackrel{\triangle}{=}\!\!\to\!\!=\!=$) and (if-E) of LPF^R become derived rules of MPL_ω after translation. Consequently, the same holds for the substitution rules ($\stackrel{\triangle}{=}$-subs) and (if-subs) as well as the rules generated from recursive function definitions according to [Jon90].

Theorem
The translation of the rules ($\stackrel{\triangle}{=}\!\!\to\!\!=\!=$) and (if-E) are derived rules of MPL_ω.
Proof:
Straightforward. □

9 Conclusions and Final Remarks

Classical reasoning can be used in the positive fragment of LPF. Generally, classical reasoning cannot be used out of the positive fragment. In particular, the deduction theorem does not hold in LPF. The departures from classical reasoning are consequences of giving up the assumption of the excluded middle.

In MPL_ω, reasoning only differs from classical reasoning with respect to variables and equality. The differences are direct consequences of embodying non-denoting terms. Such differences are also present in LPF. But unlike formulae of LPF, formulae of MPL_ω do not inherit the possibility of being non-denoting.

Free variables and bound variables are treated the same in LPF (variables always denote). The different treatment of free variables and bound variables in MPL_ω is usual in free logics, e.g. in Scott's free logic. Owing to the different treatment of free variables and bound variables, many references to undefined (i.e. non-denoting terms) can be avoided.

In both logics, equality is such that non-denoting terms are not equal, not even when they are identical (for non-denoting terms t, $t = t$ is neither-true-nor-false in LPF and

false in MPL_ω). But the kind of equality that equates all non-denoting terms can be expressed in both logics. The substitution rules ($\triangleq$-subs) and (if-subs), which can be regarded as derived rules of LPF extended with recursive function definitions, permit proofs of properties concerning simple recursively defined functions to avoid use of two kinds of equality. Similar derived rules can be devised for MPL_ω.

In this paper, it is shown that LPF can be reduced to MPL_ω and that recursive function definitions in VDM style can be represented in MPL_ω. Formulae of LPF can be treated as terms of MPL_ω. Thus, LPF-formulae's truth, falsehood or either lack can be expressed in MPL_ω. Because recursive definitions of partial functions can also be expressed in MPL_ω, reasoning about these functions can be done within MPL_ω. Even properties that depend upon the leastness of the functions can be proved.

First of all, these results demonstrate that three-valued logics such as LPF are not necessary to deal with partial functions: formulae can be translated to formulae of a two-valued logic and what can be proved remains the same after translation. What is more, the results demonstrate that MPL_ω allows for a layered approach to handle partial functions: formulae of LPF can be regarded as abbreviations of terms of MPL_ω. Thereby, it becomes possible to switch between reasoning in LPF and reasoning in MPL_ω. In that case, reasoning in LPF should be taken for being derived from reasoning in MPL_ω.

LPF has to be extended for reasoning about recursively defined function. The results about recursive definitions of partial functions justify the additional rules ($\triangleq{\to}{=}{=}$) and (if-E) (thus, they also justify the generation of rules from recursive function definitions according to [Jon90]). LPF has also to be extended for reasoning about the data types used in VDM (e.g. natural numbers, finite sets, composite objects), see also [Jon90]. The rules concerned can be justified in the same vein. The required translations are already worked out in [Mid90].

References

[Avr88] A. Avron. Foundations and proof theory of 3-valued logics. LFCS Report ECS-LFCS-88-48, University of Edinburgh, Department of Computer Science, 1988.

[BCJ84] H. Barringer, H. Cheng, and C.B. Jones. A logic covering undefinedness in program proofs. *Acta Informatica*, 21:251–269, 1984.

[Bee85] M.J. Beeson. *Foundations of Constructive Mathematics*. Springer Verlag, 1985.

[Bee88] M.J. Beeson. Towards a computation system based on set theory. *Theoretical Computer Science*, 60:297–340, 1988.

[Ben86] E. Bencivenga. Free logics. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic, Volume III*, chapter III.6. D. Reidel Publishing Company, 1986.

[Bla86] S. Blamey. Partial logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic, Volume III*, chapter III.1. D. Reidel Publishing Company, 1986.

[Che86] J.H. Cheng. *A Logic for Partial Functions*. PhD thesis, University of Manchester, Department of Computer Science, 1986. Technical Report UMCS-86-7-1.

[CJ90] J.H. Cheng and C.B. Jones. On the usability of logics which handle partial functions. Technical Report UMCS-90-3-1, University of Manchester, Department of Computer Science, 1990.

[GL90] A. Gavilanes-Franco and F. Lucio-Carrasco. A first order logic for partial functions. *Theoretical Computer Science*, 74:37–69, 1990.

[GMW79] M.J.C. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF*. Springer Verlag, LNCS 78, 1979.

[Jon86] C.B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall, first edition, 1986.

[Jon89] H.B.M. Jonkers. An introduction to COLD-K. In M. Wirsing and J.A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications*, pages 139–205. Springer Verlag, LNCS 394, 1989.

[Jon90] C.B. Jones. *Systematic Software Development Using VDM*. Prentice-Hall, second edition, 1990.

[JS90] C.B. Jones and R.C.F. Shaw. *Case Studies in Systematic Software Development*. Prentice-Hall, 1990.

[Kar64] C. Karp. *Languages with Expressions of Infinite Length*. North-Holland, 1964.

[Kei71] H.J. Keisler. *Model Theory for Infinitary Logic*. North-Holland, 1971.

[Kle52] S.C. Kleene. *Introduction to Metamathematics*. North-Holland, 1952.

[KR89] C.P.J. Koymans and G.R. Renardel de Lavalette. The logic MPL_ω. In M. Wirsing and J.A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications*, pages 247–282. Springer Verlag, LNCS 394, 1989.

[KTB88] B. Konikowska, A. Tarlecki, and A. Blikle. A three-valued logic for software specification and validation. In R. Bloomfield, L. Marshall, and R. Jones, editors, *VDM '88*, pages 218–242. Springer Verlag, LNCS 328, 1988.

[Łuk67] J. Łukasiewicz. On three-valued logic. In S. McCall, editor, *Polish Logic 1920–1939*. Oxford University Press, 1967.

[McC67] J. McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 33–70. North-Holland Publishing Company, 1967.

[Mid90] C.A. Middelburg. *Syntax and Semantics of VVSL — A Language for Structured VDM Specifications*. PhD thesis, University of Amsterdam, September 1990. Available from PTT Research, Dr. Neher Laboratories.

[Owe84] O. Owe. An approach to program reasoning based on a first-order logic of partial functions. Computer Science Technical Report CS-081, University of California, San Diego, 1984.

[Pau87] L.C. Paulson. *Logic and Computation*. Cambridge University Press, Cambridge Tracts in Theoretical Computer Science 2, 1987.

[Plo85] G.D. Plotkin. Partial function logic. Lectures at Edinburgh University, 1985.

[Ren84] G.R. Renardel de Lavalette. Descriptions in mathematical logic. *Studia Logica*, 43:281–294, 1984.

[Ren89] G.R. Renardel de Lavalette. COLD-K^2, the static kernel of COLD-K. Report RP/mod-89/8, SERC, 1989.

[Sco67] D.S. Scott. Existence and description in formal logic. In R. Schoenman, editor, *Bertrand Russell, Philosopher of the Century*, pages 181–200. Allen & Unwin, 1967.

[Sco79] D.S. Scott. Identity and existence in intuitionistic logic. In M.P. Fourman, C.J. Mulvey, and D.S. Scott, editors, *Applications of Sheaves*, pages 660–696. Springer Verlag, Lecture Notes in Mathematics 753, 1979.

[Tho89] S. Thompson. A logic for Miranda. *Formal Aspects of Computing*, 1(4):339–365, 1989.