COMPUTING SCIENCE

A Semantic Analysis of Logics that Cope with Partial Terms

C. B. Jones, M. J. Lovert and L. J. Steggles

TECHNICAL REPORT SERIES

No. CS-TR-1310 January 2012

TECHNICAL REPORT SERIES

No. CS-TR-1310 January, 2012

A Semantic Analysis of Logics that Cope with Partial Terms

C.B. Jones, M.J. Lovert, L.J. Steggles

Abstract

Specifications of programs frequently involve operators and functions that are not defined over all of their (syntactic) domains. Proofs about specifications -and those to discharge proof obligations that arise in justifying steps of design- must be based on formal rules. Since classical logic deals only with defined values, some extra thought is required. There are several ways of handling terms that can fail to denote a value – this paper provides a semantically based comparison of three of the best known approaches. In addition, some pointers are given to further alternatives.

Printed and published by Newcastle University,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

JONES, C.B., LOVERT, M.J., STEGGLES, L.J.

A Semantic Analysis of Logics that Cope with Partial Terms
[By] C.B. Jones, M.J. Lovert, L.J. Steggles
Newcastle upon Tyne: Newcastle University: Computing Science, 2012.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1310)

Added entries

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series. CS-TR-1310

Abstract

Specifications of programs frequently involve operators and functions that are not defined over all of their (syntactic) domains. Proofs about specifications -and those to discharge proof obligations that arise in justifying steps of design- must be based on formal rules. Since classical logic deals only with defined values, some extra thought is required. There are several ways of handling terms that can fail to denote a value – this paper provides a semantically based comparison of three of the best known approaches. In addition, some pointers are given to further alternatives.

About the authors

Cliff B. Jones is currently Professor of Computing Science at Newcastle University. As well as his academic career, Cliff has spent over 20 years in industry. His 15 years in IBM saw among other things the creation –with colleagues in Vienna– of VDM which is one of the better known “formal methods”. Under Tony Hoare, Cliff wrote his doctoral thesis in two years. From Oxford, he moved directly to a chair at Manchester University where he built a world-class Formal Methods group which –among other projects– was the academic lead in the largest Software Engineering project funded by the Alvey programme (IPSE 2.5 created the “mural” (Formal Method) Support Systems theorem proving assistant). He is now applying research on formal methods to wider issues of dependability. Until 2007 his major research involvement was the five university IRC on “Dependability of Computer-Based Systems” of which he was overall Project Director. He is also PI on an EPSRC-funded project “AI4FM” and coordinates the “Methodology” strand of the EU-funded DEPLOY project. He also leads the ICT research in the ITRC Program Grant. Cliff is a Fellow of the Royal Academy of Engineering (FREng), ACM, BCS, and IET. He has been a member of IFIP Working Group 2.3 (Programming Methodology) since 1973 (and was Chair from 1987-96).

Matthew is a PhD student at Newcastle University under the supervision of Prof. Cliff Jones and Dr. Jason Steggles. He is undertaking research on mechanised proof support tools for the Logic of Partial Functions. Matthew gained his BSc in Computing Science with First Class Honours from Newcastle University in 2008, during which time he was awarded with a BCS prize, a Scott Logic prize and a British Airways prize for outstanding performance in each stage of his degree course.

Dr L. Jason Steggles is a lecturer in the School of Computing Science, University of Newcastle. His research interests lie in the use of formal techniques to develop correct computing systems. In particular, he has worked extensively on using algebraic methods for specifying, prototyping and validating computing systems.

Suggested keywords

PARTIAL TERMS
PARTIAL FUNCTIONS
CLASSICAL LOGIC
RELATIONAL OPERATORS
LOGIC OF PARTIAL FUNCTIONS
SEMANTIC MODELS

A Semantic Analysis of Logics that Cope with Partial Terms

C. B. Jones, M. J. Lovert, and L. J. Steggles

School of Computing Science, Newcastle University, NE1 7RU, UK
{cliff.jones,matthew.lovert,l.j.steggles}@ncl.ac.uk

Abstract. Specifications of programs frequently involve operators and functions that are not defined over all of their (syntactic) domains. Proofs about specifications –and those to discharge proof obligations that arise in justifying steps of design– must be based on formal rules. Since classical logic deals only with defined values, some extra thought is required. There are several ways of handling terms that can fail to denote a value — this paper provides a semantically based comparison of three of the best known approaches. In addition, some pointers are given to further alternatives.

1 Introduction

Terms such as the head of an empty sequence (**hd** $[\,]$), applying a mapping outside its actual domain ($\{1 \mapsto 1\}(2)$), or even the obvious $7/0$ can be considered to fail to denote values. Of course, it would be perverse to write such naked terms deliberately but the fact is that they arise as sub-terms of quite innocent expressions. What some people call "undefined terms" are ubiquitous in reasoning about realistic program specifications and designs.

In some uses, it is tempting to try to "guard" dangerous applications by writing expressions such as:[1]

$$\forall i{:}\,\mathbb{Z} \cdot i \neq 0 \;\Rightarrow\; i/i = 1 \tag{1}$$

But there are other expressions that cannot be rewritten with such guards; consider:

$$\forall i{:}\,\mathbb{Z} \cdot (i/i = 1) \vee ((i-1)/(i-1) = 1) \tag{2}$$

Although also verging on the contrived, disjunctions where either term can be undefined –but only in the case where the other disjunct is true– arise quite naturally in specifications. The same can be said of conditions under which conjunctions and implications come into contact with "undefinedness".

The issue of reasoning about such partial terms in program development has long been recognised; certainly [McC67] discusses the problem and the issue has since been tackled in a variety of approaches [Owe85, Che86, Ten87, Bli88, KTB88, Jer88, Spi88,

[1] Assume that x/y represents integer division and that it does not yield a defined result with a zero divisor.

Jon90, CJ91, MS97, GSE95, Jon06, Fit07, Sch11]. The topic is discussed by logicians such as in [Łuk20, Wan61, vF66, Kol76, Kol81, Hoo87, Avr88, Far90, Mac01].

The first author of the current paper has long advocated the use of a non-classical "Logic of Partial Functions" (LPF) [BCJ84]. LPF is a first order predicate logic *designed* to handle non-denoting values that can arise from terms that apply partial functions and operators. LPF underlies the *Vienna Development Method* (*VDM*) [Jon90, BFL$^+$94, Fit07]. A soundness proof of untyped LPF is given in [Che86] and of the typed version in [JM94].

Recently, all three authors have been looking at the issue of providing (efficient) mechanisations of LPF. One fruit of this is [JL11] that presents two semantic models for LPF. A paper on the adaptation of (semi-)decision procedures such as resolution and refutation to cope with LPF is about to be submitted to a journal [JLS11]. The underpinning of that research is a semantics that maps logical expressions to relations over interpretations and results. This nicely captures Blamey's [Bla80, Bla86] view that non-denoting terms correspond to "gaps": so $7/0$ or the head of an empty sequence map to an empty relation but i/i maps to interpretations which have a gap for $i = 0$.

In spite of the fact that the "gap" view is key to the semantic models presented later, it is convenient to first *illustrate* the three main approaches to handling partial terms being considered in this paper by using a surrogate for the "undefined" value (which, of course, can often not be computed). In these illustrations, $\perp_{\mathbb{Z}}$ is written to stand for a missing integer value and $\perp_{\mathbb{B}}$ for a missing Boolean value; then $\mathbb{B}_{\perp}$ ($\mathbb{Z}_{\perp}$) is taken to mean $\mathbb{B} \cup \{\perp_{\mathbb{B}}\}$ ($\mathbb{Z} \cup \{\perp_{\mathbb{Z}}\}$) respectively.

Essentially, the first two approaches below attempt to get by with classical logic by "catching undefinedness" before it collides with the logical operators to avoid any contact with non-denoting logical values. In other words providing work-arounds so that a classical (total) framework can still be used. The third approach considers using a non-classical (three-valued) logic.

The first approach (see Section 3) is to insist that all terms do in fact denote something (perhaps $0/0 = 42$) and is pictured in Figure 1(a). Another approach (see Section 4) is to accept that terms such as i/i can fail to denote but to make any predicates (e.g. the relational operators) denote, even in those situations where their arguments fail to denote; this approach is pictured in Figure 1(b).

The third approach uses a non-classical logic, notably LPF (see Section 5), whose attempt to "catch undefinedness" is pictured in Figure 1(c). Here the gaps from partial terms are allowed to propagate up so that the problem *can* be "resolved" by the logical operators. Although the conditional operators of [McC67] do not retain properties like the commutativity of disjunctions and conjunctions, they broadly fit the picture depicted for LPF and this approach is discussed in Section 6.

A semantic model of the sort first presented in [JL11] for LPF is in fact quite convenient for comparing different approaches to handling partial terms and this is the focus of this paper. Using the definitions and ideas introduced in Section 2, a semantic model is developed for each of the three approaches to handling partial terms described above (see Sections 3–5). These are then used to compare and contrast the three approaches in Section 6, which also highlights further approaches of interest.

$$\forall i{:}\,\mathbb{Z} \cdot (\overbrace{i/i}^{\in\mathbb{Z}} = 1) \vee (\overbrace{(i-1)/(i-1)}^{\in\mathbb{Z}} = 1) \quad \text{(a)}$$

$$\forall i{:}\,\mathbb{Z} \cdot \overbrace{(\underbrace{i/i}_{\in\mathbb{Z}_\perp} =_\exists 1)}^{\in\mathbb{B}} \vee \overbrace{(\underbrace{(i-1)/(i-1)}_{\in\mathbb{Z}_\perp} =_\exists 1)}^{\in\mathbb{B}} \quad \text{(b)}$$

$$\forall i{:}\,\mathbb{Z} \cdot \overbrace{\underbrace{(\underbrace{i/i}_{\in\mathbb{Z}_\perp} = 1)}_{\in\mathbb{B}_\perp} \vee \underbrace{(\underbrace{(i-1)/(i-1)}_{\in\mathbb{Z}_\perp} = 1)}_{\in\mathbb{B}_\perp}}^{\in\mathbb{B}} \quad \text{(c)}$$

Fig. 1. An illustration of where "undefinedness" can be caught: (a) a classical approach insisting that all terms denote; (b) a non-strict relational operator approach; and (c) the LPF approach.

2 The Basis of the Semantics

An abstract syntax (using VDM notation [Jon90]) is presented in Figure 2. It is this abstract syntax that is used in the semantic models presented in this paper (although, when writing expressions in examples, concrete syntax is used for readability).

As in most logic textbooks, only a few logical operators are considered since further logical operators can be defined from this subset — and a logic is unlikely to be usable unless its operators enjoy connections such as de Morgan's laws. One predicate –equality, defined only for integer operands– and two functions –subtraction and division– suffice to illustrate the issues. Finally, quantification is only considered to be over the set of integer values and the only constant values are Booleans and integers.

Context conditions for such a language are outlined in [JL11] and spelt out formally in [Lov10]. The context conditions ensure that the semantics only need be given for expressions that are well-formed thus removing the need to define semantics for ill-formed expressions such as $mk\text{-}Exists(x, 5)$, i.e. $\exists x \cdot 5$.

Two sorts of identifiers can occur in expressions, those for propositions ($Prop$) and those for integer variables (Var). The sets $Prop$ and Var are assumed to be disjoint. It is one of the functions of the context conditions to ensure that identifiers are used appropriately. Furthermore, it is required that all integer variables are explicitly bound by quantifiers.

States ($\sigma \in \Sigma$) provide a (possibly partial) interpretation for propositional and integer variable symbols. Formally, Σ is defined as the union of two sets of maps:

$$\begin{aligned}\Sigma = {} & Prop \xrightarrow{m} \mathbb{B} \mid \\ & Var \xrightarrow{m} \mathbb{Z}\end{aligned}$$

where the map involving $Prop$ is partial in the sense that a propositional identifier can be absent from the domain of a specific map ($\sigma \in \Sigma$) to allow for the possibility of undefined propositional identifiers. However, the Var map must be total since all Var are explicitly bound by quantifiers and in classical logic and in LPF quantification is only over defined values.

$$Expr = Value \mid Id \mid Arith \mid Equality \mid Not \mid Or \mid Exists$$

$$Value = \mathbb{B} \mid \mathbb{Z}$$

$$Id = Prop \mid Var$$

$$\begin{array}{lll} Arith :: & a & : Expr \\ & op & : - \mid / \\ & b & : Expr \end{array}$$

$$\begin{array}{lll} Equality :: & a & : Expr \\ & b & : Expr \end{array}$$

$$Not :: a : Expr$$

$$\begin{array}{lll} Or :: & a & : Expr \\ & b & : Expr \end{array}$$

$$\begin{array}{lll} Exists :: & bind & : Id \\ & body & : Expr \end{array}$$

Fig. 2. The abstract syntax of the language.

The semantics is given for each of the three approaches to handling partial terms by defining a semantic function for each with following form:

$$\mathcal{E} : Expr \to \mathcal{P}(\Sigma \times Value)$$
$$\mathcal{E}(e) \triangleq \ldots$$

3 Classical Logic: Making all Terms Denote

As indicated in Figure 1(a), it is possible to get by with classical logical operators by forcing an extension of functions and operators so that they are total. To make division yield a result with a zero divisor is a challenge but it is possible to say that $7/0$ yields some arbitrary integer and that perhaps no harm is done by this fiction providing that it is not possible to know which integer results. Figure 3 presents a formal semantics for this approach where division by zero is extended to return an arbitrary integer. The rest of this definition is straightforward in the sense that any feature of the language of Figure 2 has the obvious classical meaning.

To ensure that all propositional variables do denote, the set of variable state mappings needs to be appropriately defined. Let Σ^C be the set of mappings that contain denotations for all used elements of $Prop$ and Var:

$$\Sigma^C = \{\sigma \mid \sigma \in \Sigma \wedge \mathbf{dom}\,\sigma = Id\}$$

Relations are chosen as the space of denotations to facilitate comparison with the semantics in the next two sections.

Notice that this approach is total as the definition of $\mathcal{E}^C$ avoids the possibility of "gaps". In other words, for every expression e and each $\sigma \in \Sigma^C$ there exists a tuple $(\sigma, v) \in \mathcal{E}^C(e)$. This is straightforward to prove by structural induction over $Expr$. The relation, however, is not deterministic (or "functional") since it is not single-valued, i.e. $7/0 = 7/0$ can yield both true and false.

What has been done in $\mathcal{E}^C$ is to underspecify the partial division function so that it returns an arbitrary value when applied outside of its actual defined domain. An alternative approach is to overspecify the result, in other words, to define that a partial function must return a default value when applied outside of its actual defined domain, e.g. $i/0$ returns 42.[2] The $\mathcal{E}^D$ semantics presented in Figure 4 documents the small change needed to instead overspecify the partial division function. The rest of the expression cases follow as in the $\mathcal{E}^C$ semantics, if all other occurrences of $\mathcal{E}^C$ are replaced with $\mathcal{E}^D$.

It is straightforward to show the semantic function $\mathcal{E}^D$ is total and also deterministic (for any expression e it follows that $(\sigma, v_1) \in \mathcal{E}^D(e) \wedge (\sigma, v_2) \in \mathcal{E}^D(e) \;\Rightarrow\; v_1 = v_2$).

4 Classical Logic: Variant Relational Operators

Figure 1(b) indicates that there is another way to preserve classical logic and that is by having non-strict relational operators denote even when their arguments fail to denote. Non-strict notions of equality include *existential equality* ($=_\exists$) and *strong equality* ($==$). The truth table for existential equality is presented in Figure 5 and strong equality differs only in the case when both of their operands do not denote, so $\bot_\mathbb{Z} =_\exists \bot_\mathbb{Z}$ is false, but $\bot_\mathbb{Z} == \bot_\mathbb{Z}$ is true. Existential equality is the focus throughout this section.

The semantic function $\mathcal{E}^\exists$ is defined in Figure 6 using a similar approach to $\mathcal{E}^C$ but replacing the case for division by the normal partial division definition and by replacing the case for equality by existential equality. Additionally any further use of $\mathcal{E}^C$ needs to be replaced with $\mathcal{E}^\exists$. Note that the set of variable state mappings remains as Σ^C since propositional variables are not permitted to be a source of non-denoting terms.

The $\mathcal{E}^\exists$ semantics is total in the sense that for every Boolean expression e and each $\sigma \in \Sigma^C$ there must be a tuple $(\sigma, v) \in \mathcal{E}^\exists(e)$ but notice that if, as is pointed out in Section 6, the need for both strict and non-strict equality is recognised, then there still exists the danger of partial terms being written by mistake. The $\mathcal{E}^\exists$ semantics is also deterministic.

5 Non-classical Logic: LPF

If the truth is told about partial functions and operators, there are gaps in the denotations where they fail to denote: $7/0$ is not an integer; furthermore, if discussion is limited to the one strict notion of equality, $7/0 = 42$ fails to denote a Boolean value. Briefly revisiting Property 2, it should be clear that its truth relies on the truth of disjunctions such as $(1/1 = 1) \vee (0/0 = 1)$, which reduces to $(1 = 1) \vee (\bot_\mathbb{Z} = 1)$ and further to $true \vee \bot_\mathbb{B}$, since the equality is strict (i.e. undefined if either operand is undefined) and

[2] $0/0 = 42$ proof by Douglas Adams.

$$
\begin{array}{l}
\mathcal{E}^C : \mathit{Expr} \to \mathcal{P}(\Sigma^C \times \mathit{Value}) \\
\mathcal{E}^C(e) \triangleq \\
\quad \textbf{cases } e \textbf{ of} \\
\quad e \in \mathit{Value} \to \{(\sigma, e) \mid \sigma \in \Sigma^C\} \\
\quad e \in \mathit{Prop} \to \{(\sigma, \sigma(e)) \mid \sigma \in \Sigma^C\} \\
\quad e \in \mathit{Var} \to \{(\sigma, \sigma(e)) \mid \sigma \in \Sigma^C\} \\
\quad \mathit{mk\text{-}Arith}(a, -, b) \to \{(\sigma, a' - b') \mid (\sigma, a') \in \mathcal{E}(a) \wedge (\sigma, b') \in \mathcal{E}(b)\} \\
\quad \mathit{mk\text{-}Arith}(a, /, b) \to \{(\sigma, a'/b') \mid (\sigma, a') \in \mathcal{E}^C(a) \wedge \\
\qquad\qquad (\sigma, b') \in \mathcal{E}^C(b) \wedge b' \neq 0\} \cup \\
\qquad\qquad \{(\sigma, n) \mid (\sigma, a') \in \mathcal{E}^C(a) \wedge \\
\qquad\qquad\quad (\sigma, b') \in \mathcal{E}^C(b) \wedge b' = 0 \wedge n \in \mathbb{Z}\} \\
\quad \mathit{mk\text{-}Equality}(a, b) \to \{(\sigma, a' = b') \mid (\sigma, a') \in \mathcal{E}^C(a) \wedge (\sigma, b') \in \mathcal{E}^C(b)\} \\
\quad \mathit{mk\text{-}Not}(p) \to \{(\sigma, \neg\, p') \mid (\sigma, p') \in \mathcal{E}^C(p)\} \\
\quad \mathit{mk\text{-}Or}(p, q) \to \{(\sigma, p' \vee q') \mid (\sigma, p') \in \mathcal{E}^C(p) \wedge (\sigma, q') \in \mathcal{E}^C(q)\} \\
\quad \mathit{mk\text{-}Exists}(x, p) \to \{(\sigma, \exists i{:}\,\mathbb{Z} \cdot (\sigma \dagger \{x \mapsto i\}, \textbf{true}) \in \mathcal{E}^C(p)) \mid \sigma \in \Sigma^C\} \\
\quad \textbf{end}
\end{array}
$$

Fig. 3. The semantic function $\mathcal{E}^C$ — an approach to making all terms denote.

$$
\begin{array}{l}
\mathcal{E}^D : \mathit{Expr} \to \mathcal{P}(\Sigma^C \times \mathit{Value}) \\
\mathcal{E}^D(e) \triangleq \\
\quad \textbf{cases } e \textbf{ of} \\
\quad \vdots \\
\quad \mathit{mk\text{-}Arith}(a, /, b) \to \{(\sigma, a'/b') \mid (\sigma, a') \in \mathcal{E}^D(a) \wedge \\
\qquad\qquad (\sigma, b') \in \mathcal{E}^D(b) \wedge b' \neq 0\} \cup \\
\qquad\qquad \{(\sigma, 42) \mid (\sigma, a') \in \mathcal{E}^D(a) \wedge \\
\qquad\qquad\quad (\sigma, b') \in \mathcal{E}^D(b) \wedge b' = 0\} \\
\quad \vdots \\
\quad \textbf{end}
\end{array}
$$

Fig. 4. The semantic function $\mathcal{E}^D$ — another approach to making all terms denote.

$=_\exists$	0	1	2	...	$\perp_\mathbb{Z}$
0	**true**	**false**	**false**	...	**false**
1	**false**	**true**	**false**	...	**false**
2	**false**	**false**	**true**	...	**false**
...	...	...	...	...	...
$\perp_\mathbb{Z}$	**false**	**false**	**false**	...	**false**

Fig. 5. The truth table for existential equality with integer operands.

$$\mathcal{E}^{\exists} : \mathit{Expr} \to \mathcal{P}(\Sigma^C \times \mathit{Value})$$

$$\mathcal{E}^{\exists}(e) \triangleq$$

cases e **of**

$$\vdots$$

$$\begin{aligned}
\mathit{mk\text{-}Arith}(a, /, b) \to\ & \{(\sigma, a'/b') \mid (\sigma, a') \in \mathcal{E}^{\exists}(a) \wedge \\
& \qquad (\sigma, b') \in \mathcal{E}^{\exists}(b) \wedge b' \neq 0\} \\
\mathit{mk\text{-}Equality}(a, b) \to\ & \{(\sigma, a' = b') \mid (\sigma, a') \in \mathcal{E}^{\exists}(a) \wedge (\sigma, b') \in \mathcal{E}^{\exists}(b)\} \cup \\
& \{(\sigma, \textbf{false}) \mid \sigma \in (\Sigma^C \setminus \textbf{dom}\, \mathcal{E}^{\exists}(a))\} \cup \\
& \{(\sigma, \textbf{false}) \mid \sigma \in (\Sigma^C \setminus \textbf{dom}\, \mathcal{E}^{\exists}(b))\}
\end{aligned}$$

$$\vdots$$

end

Fig. 6. The semantic function $\mathcal{E}^{\exists}$ for the approach of including a non-strict relational operator.

ultimately to $\bot_{\mathbb{B}}$. This unfortunately makes no sense in classical logic since its truth tables only define the logical operators for proper Boolean values.

As can be seen in Figure 1(c), this approach leaves the propositional operators to take the strain. The truth tables (disjunction, conjunction and negation) in Figure 7 (presented in [Kle52, §64]) illustrate how the propositional operators in LPF have been extended to handle logical values that may fail to denote. These truth tables provide the strongest possible *monotonic* extension of the familiar propositional operators with respect to the following ordering on the truth values: $\bot_{\mathbb{B}} \preceq \mathit{true}$ and $\bot_{\mathbb{B}} \preceq \mathit{false}$. The truth tables can be viewed as describing a parallel lazy evaluation of the operands, whereby a result is delivered as soon as enough information is available and such a result cannot be contradicted if a $\bot_{\mathbb{B}}$ later evaluates to a proper Boolean value.

$\vee$	**true**	$\bot_{\mathbb{B}}$	**false**
true	**true**	**true**	**true**
$\bot_{\mathbb{B}}$	**true**	$\bot_{\mathbb{B}}$	$\bot_{\mathbb{B}}$
false	**true**	$\bot_{\mathbb{B}}$	**false**

$\wedge$	**true**	$\bot_{\mathbb{B}}$	**false**
true	**true**	$\bot_{\mathbb{B}}$	**false**
$\bot_{\mathbb{B}}$	$\bot_{\mathbb{B}}$	$\bot_{\mathbb{B}}$	**false**
false	**false**	**false**	**false**

$\neg$	
true	**false**
$\bot_{\mathbb{B}}$	$\bot_{\mathbb{B}}$
false	**true**

Δ	
true	**true**
$\bot_{\mathbb{B}}$	**false**
false	**true**

Fig. 7. The LPF truth tables for disjunction, conjunction, negation and definedness (Δ).

The quantifiers of LPF are a natural extension of the propositional operators — viewing existential quantification as an infinite disjunction (in the worst case) and universal quantification as an infinite conjunction. Thus, an existentially quantified expression in LPF is true if a witness value exists even if the quantified expression is undefined or false for some of the bound values. Such an expression is false if no witness value can be shown. Similar comments apply for universally quantified expressions.

For expressive completeness, LPF includes a definedness operator Δ whose truth table is also presented in Figure 7. Unlike all of the other operators presented, the Δ

operator is not monotone. However, Δ tends not to be used in normal assertions and it is considered to be an operator on the meta-level.

A semantics for the LPF version of the Predicate Calculus is detailed below. The abstract syntax is extended to include Δ, thus $Expr^L = Expr \mid Delta$ and where the abstract syntax for $Delta$ is the same as for Not.

Since in LPF, the logical operators are extended to allow for the possibility that non-denoting logical values can be "caught", the standard definition of Σ (given in Section 2) can be used for LPF, thus allowing for undefined propositional identifiers to occur in a specific σ.

The semantic function $\mathcal{E}^L$ is defined as $\mathcal{E}^C$, but with the additional and modified cases presented in Figure 8; also any use of $\mathcal{E}^C$ needs to be replaced with $\mathcal{E}^L$ and any use of Σ^C needs to be replaced with Σ. Note that the semantics for quantifiers ensures that "gaps" are handled by non-denoting propositional expressions being absent from the domain of $\mathcal{E}^L$.

$$
\begin{array}{l}
\mathcal{E}^L : Expr^L \to \mathcal{P}(\Sigma \times Value) \\
\mathcal{E}^L(e) \triangleq \\
\quad \textbf{cases } e \textbf{ of} \\
\quad \vdots \\
\quad e \in Prop \to \{(\sigma, \sigma(e)) \mid \sigma \in \Sigma \wedge e \in \textbf{dom}\, \sigma\} \\
\quad e \in Var \to \{(\sigma, \sigma(e)) \mid \sigma \in \Sigma\} \\
\quad mk\text{-}Arith(a, /, b) \to \{(\sigma, a'/b') \mid (\sigma, a') \in \mathcal{E}^L(a) \wedge \\
\qquad\qquad (\sigma, b') \in \mathcal{E}^L(b) \wedge b' \neq 0\} \\
\quad \vdots \\
\quad mk\text{-}Delta(p) \to \{(\sigma, \textbf{true}) \mid \sigma \in \textbf{dom}\, \mathcal{E}^L(p)\} \cup \\
\qquad\qquad \{(\sigma, \textbf{false}) \mid \sigma \in (\Sigma \setminus \textbf{dom}\, \mathcal{E}^L(p))\} \\
\quad mk\text{-}Not(p) \to \{(\sigma, \textbf{true}) \mid (\sigma, \textbf{false}) \in \mathcal{E}^L(p)\} \cup \\
\qquad\qquad \{(\sigma, \textbf{false}) \mid (\sigma, \textbf{true}) \in \mathcal{E}^L(p)\} \\
\quad mk\text{-}Or(p, q) \to \{(\sigma, \textbf{true}) \mid (\sigma, \textbf{true}) \in \mathcal{E}^L(p)\} \cup \\
\qquad\qquad \{(\sigma, \textbf{true}) \mid (\sigma, \textbf{true}) \in \mathcal{E}^L(q)\} \cup \\
\qquad\qquad \{(\sigma, \textbf{false}) \mid (\sigma, \textbf{false}) \in \mathcal{E}^L(p) \wedge (\sigma, \textbf{false}) \in \mathcal{E}^L(q)\} \\
\quad mk\text{-}Exists(x, p) \to \{(\sigma, \textbf{true}) \mid \\
\qquad\qquad\qquad \sigma \in \Sigma \ \wedge \\
\qquad\qquad\qquad \textbf{true} \in \textbf{rng}\, (\{\sigma \dagger \{x \mapsto i\} \mid i{:}\,\mathbb{Z}\} \lhd \mathcal{E}^L(p))\} \cup \\
\qquad\qquad \{(\sigma, \textbf{false}) \mid \\
\qquad\qquad\qquad \sigma \in \Sigma \ \wedge \\
\qquad\qquad\qquad \textbf{rng}\, (\{\sigma \dagger \{x \mapsto i\} \mid i{:}\,\mathbb{Z}\} \lhd \mathcal{E}^L(p)) = \{\textbf{false}\}\} \\
\quad \textbf{end}
\end{array}
$$

Fig. 8. The semantic function $\mathcal{E}^L$ for LPF.

The "gaps" that arise from partial terms and propositional expressions in LPF are modelled by choosing *relations* as the space of *denotations* here. This is in contrast to

the use of partial functions as is classical in *denotational semantics* [Sto77]. The use of relations might suggest non-determinacy but all denotations are in fact single valued, i.e. any relation $\mathcal{E}^L(e)$ is deterministic (or "functional"), that is, for any expression e it follows that $(\sigma, v_1) \in \mathcal{E}^L(e) \wedge (\sigma, v_2) \in \mathcal{E}^L(e) \;\Rightarrow\; v_1 = v_2$.

6 Discussion

6.1 A Comparison of the Approaches Considered

The "proof of the pudding" for any logic is the ease of proof. Consider constructing a proof of Property 2 in classical logic; first, it is necessary to introduce some knowledge about division and subtraction, since a proof is a game with symbols, it cannot use the semantics of the arithmetic operators:

$$\forall i{:}\,\mathbb{Z} \cdot i = 0 \;\Rightarrow\; \neg\,((i-1) = 0);\;\; \forall i{:}\,\mathbb{Z} \cdot \neg\,(i = 0) \;\Rightarrow\; i/i = 1 \;\vdash\; \forall i{:}\,\mathbb{Z} \cdot (i/i = 1) \vee ((i-1)/(i-1) = 1)$$

A proof of the above property in classical logic is presented in Figure 9 and it is pleasingly straightforward even though it hides the fact that the term $0/0$ with its undetermined denotation implicitly crops up in a number of places.

	from $\forall i{:}\,\mathbb{Z} \cdot i = 0 \;\Rightarrow\; \neg\,(i-1=0);\;\; \forall i{:}\,\mathbb{Z} \cdot \neg\,(i=0) \;\Rightarrow\; i/i = 1$	
1	**from** $i{:}\,\mathbb{Z}$	
1.1	$i = 0 \vee \neg\,(i = 0)$	$h1, \mathbb{Z}$
1.2	**from** $i = 0$	
1.2.1	$\neg\,(i - 1 = 0)$	$\Rightarrow$-E-$L(\forall$-$E(h1, h), h1.2)$
1.2.2	$(i-1)/(i-1) = 1$	$\Rightarrow$-E-$L(\forall$-$E(h1, h), 1.2.1)$
	infer $(i/i = 1) \vee ((i-1)/(i-1) = 1)$	$\vee$-I-$L(1.2.2)$
1.3	**from** $\neg\,(i = 0)$	
1.3.1	$i/i = 1$	$\Rightarrow$-E-$L(\forall$-$E(h1, h), h1.3)$
	infer $(i/i = 1) \vee ((i-1)/(i-1) = 1)$	$\vee$-I-$R(1.3.1)$
	infer $(i/i = 1) \vee ((i-1)/(i-1) = 1)$	$\vee$-$E(1.1, 1.2, 1.3)$
	infer $\forall i{:}\,\mathbb{Z} \cdot (i/i = 1) \vee ((i-1)/(i-1) = 1)$	$\forall$-$I(1)$

Fig. 9. A proof of Property 2.

This prompts the question of how a proof of the same property would look in LPF. The answer is that it would be identical! The proof in Figure 9 is a completely correct proof in LPF but the point is that nowhere is it necessary in LPF to make assumptions about the denotation of terms with zero divisors.

However, definedness does need to be established in some LPF proofs. There *are* certain constraints on inference rules in LPF. One issue is that the, so called, *law of the excluded middle*: $p \vee \neg\, p$, does not hold because the disjunction of two undefined Boolean values is still undefined: thus $(0/0 = 1) \vee \neg\,(0/0 = 1)$ is not a tautology in LPF.

The non-monotone Δ operator in LPF does, however, give rise to an alternative property which is known as the *law of the excluded fourth*: $p \vee \neg p \vee \neg \Delta p$, that is, p is true, false or undefined. Furthermore, adding definedness hypotheses for all terms in some logical expression p is sufficient to make the validity of p in LPF and in classical logic coincide. One place where Δ arises is when one wants to use what is, in classical logic, the unrestricted deduction theorem, which does not hold in LPF because knowing $\bot_{\mathbb{B}} \vdash \bot_{\mathbb{B}}$ is not the same as $\bot_{\mathbb{B}} \;\Rightarrow\; \bot_{\mathbb{B}}$. The use of Δ can provide a sound $\Rightarrow$-I rule for LPF:[3]

$$\boxed{\Rightarrow\text{-}I}\ \frac{\Delta p;\, p \vdash q}{p \;\Rightarrow\; q}$$

Interestingly, it can be argued that the law of the excluded middle doesn't necessarily hold in $\mathcal{E}^C$ semantics where the partial division function has been underspecified! If division by 0 yields a non-deterministic result then $7/0 = 0 \vee \neg\,(7/0 = 0)$ can be false. Since it is difficult in a logic to pin down a characterisation of "giving the same value within a context", the temptation to fix on a result such as $7/0 = 42$ becomes rather strong. Giving in to this temptation however leads to questions such as whether $7/0 = 5/0$ (see [Jon95] for further discussion). The law of the excluded middle does, however, hold in the $\mathcal{E}^D$ approach.

An obvious reservation about using multiple notions of equality is that anyone reasoning in this way has to observe different properties of the two or more notions (note that a strict (computational) notion of equality still needs to be written in function definitions and that a non-strict notion of equality is needed to cope with partial terms[4]). Furthermore, although the focus here is on equality, the complications extend to include all of the other relational operators/predicates. There are also surprises in that, for example, existential inequality is *not* the negation of existential equality — they can both be false. (There is an interesting formal connection between what can be proved in $\mathcal{E}^{\exists}$ and $\mathcal{E}^L$ which is explored in [FJ08].)

6.2 Further Approaches

A longer discussion of other approaches to handling non-denoting terms can be found in [CJ91] but it is worth here making a few further points.

McCarthy's conditional operators The propositional operators are defined by (non-strict) conditional expressions [McC67], for instance, the conditional disjunction operator (p **cor** q) is defined as: **if** p **then true else** q and the truth table for **cor** is presented in Figure 10. Such a semantics is used in Raise [Gro92, Gro95].

The conditional disjunction operator case of a semantic function (similar to what has been defined above for the other approaches) for McCarthy's approach $\mathcal{E}^M$ (that would use Σ in the function signature) would be defined as:

[3] To claim definedness in a proof, the related δ operator is often used which is monotone and is the same as Δ except that $\delta\bot_{\mathbb{B}} = \bot_{\mathbb{B}}$, and therefore δp is equivalent to the assertion $p \vee \neg p$.

[4] Note that in the $\mathcal{E}^{\exists}$ semantics, existential equality has replaced the strict equality. If the strict equality was to remain –in addition to the existential equality– then the $\mathcal{E}^{\exists}$ semantics would not be total for every Boolean expression.

$$\begin{aligned} mk\text{-}Or(p,q) \rightarrow\ & \{(\sigma, \mathbf{true}) \mid (\sigma, \mathbf{true}) \in \mathcal{E}^M(p)\} \cup \\ & \{(\sigma, \mathbf{true}) \mid (\sigma, \mathbf{false}) \in \mathcal{E}^M(p) \wedge (\sigma, \mathbf{true}) \in \mathcal{E}^M(q)\} \cup \\ & \{(\sigma, \mathbf{false}) \mid (\sigma, \mathbf{false}) \in \mathcal{E}^M(p) \wedge (\sigma, \mathbf{false}) \in \mathcal{E}^M(q)\} \end{aligned}$$

The first variable in the conditional expressions is usually referred to as the "inevitable variable" because, if it is undefined, then the entire expression is undefined since conditional expressions are strict in their first argument. This means that disjunction and conjunction are no longer commutative and, additionally, quantifiers are problematic with respect to undefined values. Thus, $\exists i{:}\,\{0,1\} \cdot i/i = 1$ may not have the same truth value as $1/1 = 1 \vee 0/0 = 1$. So while, Property 1 with the conditional implication operator can be proved in McCarthy's approach, neither the contrapositive of Property 1 nor Property 2 follow for conditionally defined operators.

The conditional form of the logical operators were used in the early IBM Vienna operational semantics definitions known as VDL (see [W$^+$69, §1.1.6.2]). It was an unpreparedness to tolerate the loss of properties like commutativity of disjunction and conjunction that drove the first author of the current paper to experiment with using both the conditional and the classical operators in [Jon72] (an idea also tried in [Dij76] and [GS96]). As can be seen from [GS96], the distribution laws become problematic.

cor	**true**	$\perp_{\mathbb{B}}$	**false**
true	**true**	**true**	**true**
$\perp_{\mathbb{B}}$	$\perp_{\mathbb{B}}$	$\perp_{\mathbb{B}}$	$\perp_{\mathbb{B}}$
false	**true**	$\perp_{\mathbb{B}}$	**false**

Fig. 10. The truth table for McCarthy's conditional disjunction operator.

Avoiding function application: Several authors have tried to avoid writing the expression $f(x) = y$ and instead write it as $(x, y) \in f^r$, where f^r is the relation that is the "graph" of the function f. The key idea is that $(x, y) \in f^r$ is false when $x \notin \mathbf{dom}\, f^r$, for all y. This idea in not analysed in detail here (see [CJ91, Jon06] for further detail), because the notation becomes rather heavy[5] but it is easy to see how it could be added to the semantics used above.

Restricting the bounds on quantifiers: Another solution is to restrict quantification to sets that do not contain any values outside of the actual domains of the functions used. For example, Property 1 could be written as: $\forall i{:}\,\{i \mid i{:}\,\mathbb{Z} \wedge i \neq 0\} \cdot i/i = 1$. Unfortunately, in general, the type structure becomes both clumsy and undecidable. Refer to [CJ91, GSE95] for further information. One could even encode the actual defined domain of a partial function in its type and make its application with argument(s) outside of that domain a type error.

[5] Property 2 has to be rewritten as: $\forall i{:}\,\mathbb{Z} \cdot ((i,i),1) \in /^r \vee (((i-1),(i-1)),1) \in /^r$ and rewriting $g(f(x))$ requires an extra existential quantifier.

Restricting the expressions written: It is possible to view the relation $\mathcal{E}^L$ as total by restricting the expressions e to those for which there exists, for all $\sigma \in \Sigma$, a tuple $(\sigma, v) \in \mathcal{E}^L(e)$. As seen in [Meh08, Sch11] such well definedness ("WD") restrictions can be complicated and expand exponentially in size.

7 Conclusions

LPF is a logic designed to handle non-denoting logical values that can arise from terms that apply partial functions and operators. This paper presents a semantic model for LPF and two other popular approaches: making all terms denote values; and using non-strict relational operators. The idea was to use semantic functions which map logical expressions to relations over interpretations and results and this leads naturally to the view of non-denoting terms corresponding to "gaps". Each of the approaches attempts to catch "undefinedness" in a different place (see Figure 1) and the semantic models presented were used to compare and contrast the approaches. LPF emerges as a strong approach to handling partial terms and it is to be hoped that the progress reported in [JLS11] on efficient semi-decision procedures for LPF will lead to its wider use.

Acknowledgements

The authors gratefully acknowledge the funding for their research from an EPSRC grant for AI4FM and the Platform Grant TrAmS-2 as well an EPSRC PhD Studentship.

References

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

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

[BFL$^+$94] Juan Bicarregui, John Fitzgerald, Peter Lindsay, Richard Moore, and Brian Ritchie. *Proof in VDM: A Practitioner's Guide*. FACIT. Springer-Verlag, 1994. ISBN 3-540-19813-X.

[Bla80] S. R. Blamey. *Partial Valued Logic*. PhD thesis, Oxford University, 1980.

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

[Bli88] A. Blikle. Three-valued predicates for software specification and validation. In R. Bloomfield, L. Marshall, and R. Jones, editors, *VDM—The Way Ahead*, pages 243–266. Springer-Verlag, 1988. Lecture Notes in Computer Science, Vol. 328.

[Che86] J. H. Cheng. *A Logic for Partial Functions*. PhD thesis, University of Manchester, 1986.

[CJ91] J. H. Cheng and C. B. Jones. On the usability of logics which handle partial functions. In C. Morgan and J. C. P. Woodcock, editors, *3rd Refinement Workshop*, pages 51–69. Springer-Verlag, 1991.

[Dij76] E. W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, Englewood Cliffs, N.J., USA, 1976.

[Far90] William M. Farmer. A partial functions version of Church's simple theory of types. *Journal of Symbolic Logic*, 55(3):1269–1291, 1990.

[Fit07] J. S. Fitzgerald. The Typed Logic of Partial Functions and the Vienna Development Method. In D. Bjørner and M. C. Henson, editors, *Logics of Specification Languages*, EATCS Texts in Theoretical Computer Science, pages 427–461. Springer, 2007.

[FJ08] J. S. Fitzgerald and C. B. Jones. The connection between two ways of reasoning about partial functions. *IPL*, 107(3–4):128–132, 2008.

[Gro92] The RAISE Language Group. *The RAISE Specification Language*. BCS Practitioner Series. Prentice Hall, 1992. ISBN 0-13-752833-7.

[Gro95] The RAISE Method Group. *The RAISE Development Method*. BCS Practitioner Series. Prentice Hall, 1995. ISBN 0-13-752700-4.

[GS96] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math*. Springer-Verlag, second edition, 1996.

[GSE95] David Gries, Fred B. Schneider, and Albert Einstein. Avoiding the undefined by underspecification. In *Computer Science Today: Recent Trends and Developments, number 1000 in Lecture Notes in Computer Science*, pages 366–373. Springer-Verlag, 1995.

[Hoo87] A. Hoogewijs. Partial-predicate logic in computer science. *Acta Informatica*, 24:381–393, 1987.

[Jer88] C.A. Jervis. *A Theory of Program Correctness with Three Valued Logic*. PhD thesis, Leeds University, 1988.

[JL11] C. B. Jones and M. J. Lovert. Semantic models for a logic of partial functions. *IJSI*, 5:55–76, 2011.

[JLS11] C. B. Jones, M. J. Lovert, and L. J. Steggles. Towards a mechanisation of a logic that copes with partial terms. *to be submitted*, 2011.

[JM94] C.B. Jones and C.A. Middelburg. A typed logic of partial functions reconstructed classically. *Acta Informatica*, 31(5):399–430, 1994.

[Jon72] C.B. Jones. Formal development of correct algorithms: an example based on Earley's recogniser. In *SIGPLAN Notices, Volume 7 Number 1*, pages 150–169. ACM, January 1972.

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

[Jon95] C.B. Jones. Partial functions and logics: A warning. *Information Processing Letters*, 54(2):65–67, 1995.

[Jon06] Cliff B. Jones. Reasoning about partial functions in the formal development of programs. In *Proceedings of AVoCS'05*, volume 145, pages 3–25. Elsevier, Electronic Notes in Theoretical Computer Science, 2006.

[Kle52] S. C. Kleene. *Introduction to Metamathematics*. Van Nostrad, 1952.

[Kol76] G. Koletsos. Sequent calculus and partial logic. Master's thesis, Manchester University, 1976.

[Kol81] G. Koletsos. Notational and logical completeness in three-valued logic. *Bull. of the Greek Mathematical Society*, 22:121–141, 1981.

[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—The Way Ahead*, pages 218–242. Springer-Verlag, 1988. Lecture Notes in Computer Science, Vol. 328.

[Lov10] M. J. Lovert. A semantic model for a logic of partial functions. In K. Pierce, N. Plat, and S. Wolff, editors, *Proceedings of the 8th Overture Workshop*, number CS-TR-1224 in School of Computing Science Technical Report, pages 33–45. Newcastle University, 2010.

[Łuk20] J. Łukasiewicz. O logice trójwartościowej. *Ruch Filozoficzny*, pages 169–171, 1920. Translated as (On three-valued logic) in Polish Logic 1920–39, S. McCall (ed.), Oxford U.P., 1967.

[Mac01] H. MacColl. A report on MacColl's three-valued logic. In E.O. Lovett, editor, *Mathematics at the Intern. Congress of Philosophy*, volume 7, pages 157–183. Bulletin of the American Mathematical Society, 1901.

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

[Meh08] Farhad Dinshaw Mehta. *Proofs for the Working Engineer*. PhD thesis, ETH Zuerich, 2008.

[MS97] O. Müller and K. Slind. Treating partiality in a logic of total functions. *The Computer Journal*, 40(10):640–652, 1997.

[Owe85] O. Owe. An approach to program reasoning based on a first order logic for partial functions. Technical Report 89, Institute of Informatics, University of Oslo, February 1985.

[Sch11] Matthias Schmalz. Term rewriting in logics of partial functions. In Shengchao Qin and Zongyan Qiu, editors, *Formal Methods and Software Engineering*, volume 6991 of *Lecture Notes in Computer Science*, pages 633–650. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-24559-6_42.

[Spi88] J.M. Spivey. *Understanding Z—A Specification Language and its Formal Semantics*. Cambridge Tracts in Computer Science 3. Cambridge University Press, 1988.

[Sto77] J. E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1977.

[Ten87] R.D. Tennent. A note on undefined expression values in programming logic. *Information Processing Letters*, 24(5), March 1987.

[vF66] B.C. van Fraasen. Singular terms, truth-value gaps and free logic. *J. Philosophy*, 63:481–495, 1966.

[W$^+$69] K. Walk et al. Abstract syntax and interpretation of PL/I. Technical Report TR25.098, IBM Laboratory Vienna, 1969.

[Wan61] H. Wang. The calculus of partial predicates and its extension to set theory. *Math. Logic*, 7:283–288, 1961.