Formal Aspects of Computing (1998) 10: 76–96

Formal Aspects of Computing

Partiality and Nondeterminacy in Program Proofs

Joseph M. Morris and Alexander Bunkenburg[1]

Dept. of Computing Science, University of Glasgow, Glasgow

Keywords: Many-valued logic; Equational reasoning; Partial expressions; Nondeterminacy; Program refinement

Abstract. Specifications and programs make much use of nondeterministic and/or partial expressions, i.e. expressions which may yield several or no outcomes for some values of their free variables. Traditional 2-valued logics do not comfortably accommodate reasoning about undefined expressions, and do not cater at all for nondeterministic expressions. We seek to rectify this with a 4-valued typed logic **E4** which classifies formulae as either "true", "false", "neither true nor false", or "possibly true, possibly false". The logic is derived in part from the 2-valued logic **E** and the 3-valued LPF, and preserves most of the theorems of **E**. Indeed, the main result is that nondeterminacy can be added to a logic covering partiality at little cost.

1. Introduction

The basic idea of a function as a rule which determines a unique outcome for any given input (in a certain domain) is intuitively simple, and adequately describes most of the functions we meet in everyday mathematics. Just occasionally we have to stretch the point a little, as in the case of the integer division function, where we have to either invent some outcome for division by zero, or introduce the notion of partial functions. In mathematics these exceptions are for the most part just an occasional irritation, but in reasoning about programs and specifications of programs, they are the rule. For example, in computing we naturally define

[1] Supported by Leverhulme Trust.

Correspondence and offprint requests to: Joseph M. Morris, Dept. of Computing Science, University of Glasgow, Glasgow G12 8QQ, Scotland, UK. e-mail: jmm@dcs.glasgow.ac.uk, bunkenba@dcs.glasgow.ac.uk

computation rules recursively, where the recursion need not describe any outcome for inputs which do not interest the customer. Indeed, not only may rules fail to specify *any* outcome, in program specifications we meet functions which admit any of *several* outcomes for a given input. This arises when the customer tells us that a range of behaviours is acceptable. For example, the customer might be a university asking for a function that displays a list of its best students in alphabetical order by surname, not caring how two students with the same surname are ordered. We present a logic for reasoning uniformly about partial and nondeterministic expressions.

Familiar two-valued logics do not comfortably admit reasoning about partial or multi-valued functions. As an example of partiality consider

$$(\forall x{:}\mathbf{Z} \bullet x \geqslant 0 \Rightarrow fac(x) \geqslant 0)$$

where $fac \mathrel{\widehat{=}} \textbf{fun}\ x{:}\mathbf{Z} \bullet \textbf{if}\ x = 0\ \textbf{then}\ 1\ \textbf{else}\ x * fac(x-1)\ \textbf{fi}$. For $x < 0$, the definition of *fac* assigns no outcome to $fac(x)$ (and indeed a computer evaluation of such a $fac(x)$ would fail to terminate), and so the consequent in $x \geqslant 0 \Rightarrow fac(x) \geqslant 0$ is undefined. Nevertheless, all reasonable people would interpret the statement as valid, and we would like the formal logic we use to accord with this. Such a logic will have to cope with formulae such as $fac(x) \geqslant 0$, which for some values of their free variables are neither true nor false, but have the status of "neither true nor false".

To see how multi-valued expressions arise, consider for example the assertion

$$(\Box x{:}\mathbf{Z} \mid x^2 \leqslant n < (|x|+1)^2) \sqsubseteq f(|n|, 0)$$

where $f \mathrel{\widehat{=}} \textbf{fun}\ n, x{:}\mathbf{Z} \bullet \textbf{if}\ n < (x+1)^2\ \textbf{then}\ x\ \textbf{else}\ f(n, x+1)\ \textbf{fi}$. Here, the term on the left hand side is a specification of the integer part of the square root of n, where n stands for an integer. For $\Box$ read "some", for "$:\ \mathbf{Z}$" read "of type integer", for "$|$" read "such that", and for "$|x|$" read "the absolute value of x". It has no outcome if n is negative, and otherwise has two possible outcomes (the positive and negative square roots). For "$\sqsubseteq$" read here "is implemented by"; roughly speaking, a function implements a specification if for every input for which the specification defines at least one outcome, the function delivers one of those outcomes. ("Input" in the preceding means a binding of free variables.) The expression on the right-hand side of $\sqsubseteq$ is a function which yields the positive square root of $|n|$, fraction dropped. It meets the requirements laid down in the specification and so the assertion is true. We would like our logic to accommodate such assertions. Clearly we will be faced with formulae, such as $(\Box x{:}\mathbf{Z} \bullet x^2 \leqslant 10 < (x+1)^2) \geqslant 0$, with the status "possibly true, possibly false".

We conclude that formal reasoning about specifications and programs has good use for a logic which classifies formulae as "true", "false", "neither true nor false", or "possibly true, possibly false". It might seem at first sight that a four-valued logic would prove hopelessly unwieldy in comparison with two-valued logics, but we shall see that this is not necessarily so.

The logics that interest us is a family of equational logics that includes **E**, **E3**, **E**$\Box$, and **E4**. **E** is an equational version of classical predicate logic, and is described in [DiS90] and [GrS93]. **E3** is a three-valued logic with values "true", "false", and "neither true nor false" which preserves "most" of the theorems of **E**. It is, in essence, a fusion of **E** and the three-valued typed LPF [JoM94], and has been described in [MoB98]. In this paper we present **E**$\Box$ for handling nondeterminacy, and **E4** for handling partiality and nondeterminacy together, and we show how all the logics in the family are closely related.

2. E3: Partiality

We begin by briefly summarising **E3** (see [MoB98] for more details). The language of **E3** is a collection of terms (also called expressions), each one associated with a "type". Types are denoted by type symbols. The letters E, F, G, and t (possibly subscripted) stand for terms in general, and the letters T and U stand for type symbols. We classify expressions as being either "defined" or "undefined". Informally, we use the symbol $\bot$ to represent an undefined term.

For each type we are given an infinite supply of "variable symbols". We use the letters x and y to stand for variable symbols. Variable symbols are terms (of the type with which they are associated). For each type we are also given a supply of atomic terms, called "constants".

We are given one type symbol **B** (pronounced "bool"). Terms of type **B** are called "formulae"; we let P, Q, and R stand for formulae. *True* and *False* are constants of type **B** and hence are formulae. Further formulae are composed in the standard way using the "boolean connectives" $\wedge$, $\vee$, $\Rightarrow$, $\neg$, Δ, $\equiv$, $\not\equiv$, $=$, and the quantifiers $\forall$ and $\exists$. Intuitively, the term ΔE means "E is defined". The syntax of quantifiers is $(\forall x{:}T \bullet P)$, $(\forall x{:}T \mid R \bullet P)$, $(\exists x{:}T \bullet P)$ and $(\exists x{:}T \mid R \bullet P)$; the syntax of formulae is otherwise standard. Whenever we write a quantification $(\forall x{:}T \bullet P)$ etc. we require that x be of type T, but we will not explicitly say so on each occasion. Brackets may be omitted using the following precedence list (highest first):

$$\Delta \qquad \neg \qquad \wedge, \vee \qquad = \qquad \sqsubset \qquad \Rightarrow \qquad \sqsubseteq \qquad \equiv, \not\equiv$$

(The list anticipates the introduction of symbols $\sqsubset$ and $\sqsubseteq$ later on.) Note that $\wedge$ and $\vee$ have the same operator precedence, as have $\equiv$ and $\not\equiv$. It turns out that $\wedge$ and $\vee$ are associative, and we use this fact from the outset to omit brackets. Prefix unary operators such as $\neg$ and Δ bracket to the right.

The connectives $=$, $\equiv$, $\not\equiv$, and Δ are extended to all types. The boolean term $E = F$ (where E and F have the same type) is undefined if either argument is undefined and otherwise behaves like equality. The boolean term $E \equiv F$ (where E and F have the same type) is true if both arguments are undefined, false if precisely one argument is undefined, and otherwise behaves like equality. The boolean term ΔE is true if E is defined, and otherwise it is false. One of the motivations behind introducing Δ is to aid the definition of partial operations in arbitrary types. For example, a presentation of the integers may include among the axioms for division:

$$\Delta(E \div F) \equiv \Delta E \wedge \Delta F \wedge F \neq 0$$

We are given a (possibly empty) supply of "operator symbols" which are used to build up terms from simpler terms. We use the letter $\mathbf{f}$ to stand for an arbitrary operator symbol, and write $\mathbf{f}(E_0)$, $\mathbf{f}(E_0, E_1)$ etc. to stand for a term composed using $\mathbf{f}$ and terms E_0, E_1 etc. Each operator symbol has a "class" consisting of a finite sequence of type symbols of length at least 2. The "arity" of an operator symbol $\mathbf{f}$, denoted by *arity*($\mathbf{f}$), is defined to be the length of its class less 1. The i'th component of $\mathbf{f}$'s class, for i in the range 1 to *arity*($\mathbf{f}$) inclusive, is called the i'th "argument type" of $\mathbf{f}$. The final component is called its "result type". Operator symbol $\mathbf{f}$ is used to combine *arity*($\mathbf{f}$) terms where the i'th term, for each i in the range 1 to *arity*($\mathbf{f}$), is of $\mathbf{f}$'s i'th argument type. The resulting term has the result type of $\mathbf{f}$. When setting up a theory based on

the logic, class information is typically given in the form of a simple rule such as

$$\frac{E{:}\mathbf{Z} \qquad F{:}\mathbf{Z}}{E + F{:}\mathbf{Z}}$$

— this defines the class of operator + to be a sequence of three types each of which is **Z**. Note that the symbol **f** does not range over the built-in boolean connectives (Δ, ¬, ∧, etc.).

We denote by $E[x := t]$ the term got by substituting each free occurrence of x in E with t, where x and t are of the same type (with renaming as necessary to avoid free variables in t becoming bound as a result of the substitution). Substitution binds tightest of all.

The truth tables for the boolean connectives are given below:

∧	*True*	*False*	⊥
True	*True*	*False*	⊥
False	*False*	*False*	*False*
⊥	⊥	*False*	⊥

∨	*True*	*False*	⊥
True	*True*	*True*	*True*
False	*True*	*False*	⊥
⊥	*True*	⊥	⊥

¬	
True	*False*
False	*True*
⊥	⊥

Δ	
True	*True*
False	*True*
⊥	*False*

⇒	*True*	*False*	⊥
True	*True*	*False*	⊥
False	*True*	*True*	*True*
⊥	*True*	*True*	*True*

≡	*True*	*False*	⊥
True	*True*	*False*	*False*
False	*False*	*True*	*False*
⊥	*False*	*False*	*True*

=	*True*	*False*	⊥
True	*True*	*False*	⊥
False	*False*	*True*	⊥
⊥	⊥	⊥	⊥

$(\forall x{:}T \bullet P)$ is true if P is true for every x of type T; it is false if P is false for some x of type T, and otherwise it is undefined. $(\forall x{:}T \mid R \bullet P)$ is the same as $(\forall x{:}T \bullet R \Rightarrow P)$. As far as existential quantification is concerned, $(\exists x{:}T \bullet P)$ is equivalent to $\neg(\forall x{:}T \bullet \neg P)$, and $(\exists x{:}T \mid R \bullet P)$ to $\neg(\forall x{:}T \mid R \bullet \neg P)$. Observe that in quantifications, the range of the dummy excludes ⊥; formally, $(\forall x{:}T \bullet \Delta x)$ holds. In fact we can show that a term is defined iff it is equivalent to some value in the type; formally, $\Delta E \equiv (\exists x{:}T \bullet E \equiv x)$ where x not free in E. One could imagine making a different design in which the dummy ranges over ⊥ as well as over the individuals of the type. We have explored this alternative, and feel that in the end it comes down to choosing the one which leads to the simplest formulae for the kind of applications the user has in mind.

The choice of implication is perhaps a bit surprising. It was chosen for the simple empirical reason that it manages to carry over to three-valued logic just about all the important properties of implication that hold in 2-valued logic. It has a long history, dating back to [Mon67], and is discussed at length in [Avr88, Avr91].

The axioms common to all logics in the family are given in Fig. 1. The inference rules are Modus Ponens and Generalization:

$$\frac{P \qquad P \Rightarrow Q}{Q} \qquad\qquad \frac{P}{(\forall x{:}T \bullet P)}$$

The deduction theorem holds. In practice, we do not use the inference rules

Equivalence

$\equiv$-reflexivity:	$E \equiv E$
$\equiv$-symmetry:	$(E \equiv F) \equiv (F \equiv E)$
$\equiv$-truth:	$((E \equiv F) \equiv \mathit{True}) \equiv (E \equiv F)$

Negation

exchange:	$(\neg P \equiv Q) \equiv (\neg Q \equiv P)$
False-definition:	$\mathit{False} \equiv \neg \mathit{True}$
$\not\equiv$-definition:	$(E \not\equiv F) \equiv \neg(E \equiv F)$

Disjunction

$\vee$-symmetry:	$P \vee Q \equiv Q \vee P$
$\vee$-associativity:	$P \vee (Q \vee R) \equiv (P \vee Q) \vee R$
$\vee$-idempotency:	$P \vee P \equiv P$
$\vee$-zero:	$P \vee \mathit{True} \equiv \mathit{True}$
$\vee$-truth:	$((P \vee Q) \equiv \mathit{True}) \equiv (P \equiv \mathit{True}) \vee (Q \equiv \mathit{True})$

Conjunction

$\wedge$-definition:	$P \wedge Q \equiv \neg(\neg P \vee \neg Q)$
$\wedge/\vee$:	$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$
consistency:	$(P \wedge Q \equiv P) \equiv (P \vee Q \equiv Q)$
$\wedge$-truth:	$(P \wedge Q \equiv \mathit{True}) \equiv (P \equiv \mathit{True}) \wedge (Q \equiv \mathit{True})$

Implication

$\Rightarrow$-definition:	$P \Rightarrow Q \equiv (P \not\equiv \mathit{True}) \vee Q$
$\Rightarrow/\equiv$:	$P \Rightarrow (Q \equiv R) \equiv ((P \Rightarrow Q) \equiv (P \Rightarrow R))$
$\equiv$-weakening:	$(P \equiv Q) \Rightarrow (P \Rightarrow Q))$
Leibniz:	$(E \equiv F) \Rightarrow (G[x := E] \equiv G[x := F])$

Boolean definedness

Δ-definition:	$\Delta P \equiv ((P \equiv \mathit{True}) \equiv P)$

Universal quantification

$\forall/\wedge$:	$(\forall x{:}T \bullet P \wedge Q) \equiv (\forall x{:}T \bullet P) \wedge (\forall x{:}T \bullet Q)$
$\vee/\forall$:	$(\forall x{:}T \bullet P \vee Q) \equiv P \vee (\forall x{:}T \bullet Q)$, where x not free in P
$\forall/\equiv$:	$(\forall x{:}T \bullet P \equiv Q) \Rightarrow ((\forall x{:}T \bullet P) \equiv (\forall x{:}T \bullet Q))$
$\forall$-truth:	$((\forall x{:}T \bullet P) \equiv \mathit{True}) \equiv (\forall x{:}T \bullet P \equiv \mathit{True})$
interchange:	$(\forall x{:}T \bullet (\forall y{:}U \bullet P)) \equiv (\forall y{:}U \bullet (\forall x{:}T \bullet P))$
renaming:	$(\forall x{:}T \bullet P) \equiv (\forall y{:}T \bullet P[x := y])$, where y is fresh
trading $\forall$:	$(\forall x{:}T \mid R \bullet P) \equiv (\forall x{:}T \bullet R \Rightarrow P)$

Existential quantification

$\exists$-definition:	$(\exists x{:}T \bullet P) \equiv \neg(\forall x{:}T \bullet \neg P)$
$\exists\mid$-definition:	$(\exists x{:}T \mid R \bullet P) \equiv \neg(\forall x{:}T \mid R \bullet \neg P)$
$\exists$-truth:	$((\exists x{:}T \bullet P) \equiv \mathit{True}) \equiv (\exists x{:}T \bullet P \equiv \mathit{True})$

Term definedness

instantiation:	$(\forall x{:}T \bullet P) \wedge \Delta t \Rightarrow P[x := t]$
constants proper:	Δc
variables proper:	Δx

Fig. 1. Basic axioms of **E**, **E3**, **E**$\Box$, and **E4**.

definedness:	ΔE
=-definition:	$E = F \equiv (E \equiv F)$

Fig. 2. Additional axioms for **E**.

directly, but employ textual substitution mechanisms that reduce proving to an algebraic style of replacing "equals with equals"; see [DiS90] and [GrS93].

We have constructed the rather long-winded axiomatisation of Figure 1 because it captures what is common to each member of the family. Using it we can construct once-off a large body of theorems shared by all, and indeed it is surprising how much of traditional two-valued logic is preserved. For example, it can be shown that all theorems of **E** that employ only $\wedge$, $\vee$, and $\Rightarrow$ also follow from the axioms of Fig. 1. (The proof relies on the fact that LPF preserves the $\{\wedge, \vee, \Rightarrow\}$-fragment of classical logic [Avr88], that LPF proofs can be translated into **E3** proofs [MoB98], and that the translation process uses only those **E3** axioms contained in Fig. 1.) Some theorems will involve Δ, of course, such as $P \vee \neg P \vee \neg \Delta P$, where the equivalent in **E** is Δ-free, here $P \vee \neg P$, but in general Δ has a low-key presence. To obtain **E** we merely add the axioms of Fig. 2.

In **E**, there is no distinction between equality and equivalence, other than operator precedence. Of course once we admit ΔE as an axiom, we can greatly simply the axioms of Fig. 1, but that is beside the point. Our purpose is just to offer reassurance that our many-valued logics behave as we should expect when we are working in contexts in which we know that there is no possibility of partiality or nondeterminacy.

To obtain **E3** we supplement the axioms of Fig. 1 with those of Fig. 3. Axiom one-$\bot$ asserts that there is at most one undefined value. Applying it to the booleans, we can deduce that the logic has at most three values. If we want to ensure at least one undefined value, and hence that the logic has precisely three values, we can formally introduce $\bot_{\mathbf{B}}$ of type boolean by the axiom $\neg\Delta\bot_{\mathbf{B}}$. Surprisingly few of the useful theorems that arise in reasoning about programs rely on the axiom one-$\bot$. In fact, just about the only ones worth mentioning are the symmetry of equality and

$$P = Q \equiv (P \wedge Q) \vee (\neg P \wedge \neg Q). \tag{1}$$

To see that (1) relies on one-$\bot$, consider a model with two undefined boolean values $\bot_0$ and $\bot_1$ with the booleans constituting a chain lattice such that $ff < \bot_0 < \bot_1 < tt$ (where tt and ff are the denotations of *True* and *False*, respectively, in the model). Let $\wedge$ and $\vee$ be the lattice operations, let $\bot_0$ and $\bot_1$ be their own complements, and let $\bot_0 = \bot_1$ yield $\bot_1$. Observe now that with P and Q denoting the respective undefined values, the left hand side of (1) is $\bot_1$ while the right hand side is $\bot_0$. This is a useful observation because it tells us that we can introduce another strange value with relatively little impact on the existing theorems and indeed we will shortly incorporate "possibly true, possibly false" in this way.

3. E□: Nondeterministic Choice

As customers, we often leave open some details of our requirements, as when in a café we ask for "coffee with either milk or cream". The waiter in the café,

one-$\perp$:	$\Delta E \vee \Delta F \vee (E \equiv F)$
strictness:	$\Delta \mathbf{f}(..., E, ...) \Rightarrow \Delta E$
$=$-definedness:	$\Delta(E = F) \equiv \Delta E \wedge \Delta F$
$=$-definition:	$\Delta(E = F) \equiv (E = F \equiv (E \equiv F))$

Fig. 3. Additional axioms for **E3**.

whom we can view as the specifier, may feel entitled to write on his note pad one of "coffee with cream" (say), or "coffee with milk or cream" — as customers we would not object to either of these. On the other hand, when the waiter hands the order to the chef, whom we can view as the implementor, the chef will have a definite preference for "coffee with milk or cream" because if there is no cream in the kitchen, he may still be able to fulfil the order. The moral is that specifications should record the customer's alternatives for the sake of the implementor.

We assume the availability of a choice operator $\Box$: for E and F expressions of the same type, $E \Box F$ yields as outcome either the outcome of E or the outcome of F, and we have no further information about which outcome is actually delivered. We cannot even gather information by experiment, because if an evaluation of $2 \Box 3$ delivers 2 one day, there is no guarantee that it will not deliver 3 on the following day. This kind of nondeterminacy is important in deriving programs from specifications: if we ask two different programmers to make a program that involves displaying the names of persons in a database ordered alphabetically by surname, we cannot expect them to produce identical lists when there are several persons with the same surname.

The bad news is that nondeterministic expressions can create havoc with familiar laws. Consider for example, $2 \Box 4 + 2 \Box 4 = 2 * (2 \Box 4)$ — the left hand side may yield 4, 6, or 8, whereas the right-hand side yields either 4 or 8. We shall attach the logical status "possibly true, possibly false", i.e. *True* $\Box$ *False*, to an equality such as the preceding. Of course the presence of $\Box$ in a formula does not exclude the possibility that the formula holds — for example, we shall give $2 \Box 4 < 6$ the status *True*. The problem we face in admitting $\Box$ into formulae is not to complicate the logic unduly.

To begin, we introduce choice into traditional 2-valued logic without any notion of partiality, i.e. for the moment we are building a 3-valued logic in which the third value is *True* $\Box$ *False*. We call it $\mathbf{E}\Box$. The properties of $\Box$ we expect are

$$E \Box F \equiv F \Box E$$

$$E \Box (F \Box G) \equiv (E \Box F) \Box G$$

$$E \Box E \equiv E.$$

We should also expect that $\neg$, $\wedge$, $\vee$, and $=$ distribute over $\Box$ (we do not postulate that $\equiv$ distributes over $\Box$). From these postulates it is a minor clerical exercise to construct the truth tables for $\neg$, $\wedge$, $\vee$, $=$, and $\equiv$. If we continue to interpret ΔP as "P is either *True* or *False*" we easily construct the truth table for Δ. We then find — with hindsight, perhaps not surprisingly — that we get precisely the same truth tables as above, i.e. with $\perp$ replaced by *True* $\Box$ *False* throughout. However, replacing $\perp$ with *True* $\Box$ *False* in the truth table for $\Rightarrow$ does not result in the truth table we would expect from assuming that $\Rightarrow$ (right-)distributes over $\Box$ (for example, *True* $\Box$ *False* $\Rightarrow$ *False* is equivalent to *True* $\Box$ *False*

	Choice
$\sqsubseteq$-definition:	$E \sqsubseteq F \equiv (E \sqcap F \equiv E)$
$\equiv - \sqsubseteq$:	$(E \equiv F) \equiv (\forall x{:}T \bullet E \sqsubseteq x \equiv F \sqsubseteq x)$, where E, F have type T
$\sqcap$-definition:	$E \sqcap F \sqsubseteq x \equiv E \sqsubseteq x \vee F \sqsubseteq x$
	Refinement ordering
indiv-$\sqsubseteq$:	$(\forall x, y{:}T \bullet x \sqsubseteq y \equiv (x \equiv y))$
	Distributivity
$\sqcap$-distributivity:	$\mathbf{f}(..., E \sqcap F, ...) \equiv \mathbf{f}(..., E, ...) \sqcap \mathbf{f}(..., F, ...)$
	Extremes
excluded 4th:	$\Delta P \vee \Delta Q \vee (P \equiv Q)$
excluded miracle:	$(\exists x{:}T \bullet E \sqsubseteq x)$, where E has type T
	Equality
$= - True$:	$(E = F \equiv True) \equiv (\forall x{:}T \bullet E \sqsubseteq x \equiv (E \equiv x)) \wedge (E \equiv F)$, where E, F have type T
$= - False$:	$(E = F \equiv False) \equiv \neg(\exists x{:}T \bullet E \sqsubseteq x \wedge F \sqsubseteq x)$, where E, F have type T

Fig. 4. Additional axioms for **E**$\sqcap$.

if we assume that $\Rightarrow$ (right-) distributes over $\sqcap$, whereas if we replace $\bot$ with *True* $\sqcap$ *False* in the truth table for $\Rightarrow$ we are lead to the outcome *True*). On the balance of convenience, we elect to forego right-distribution of $\Rightarrow$ over $\sqcap$, with the pleasing effect that the large body of theorems that follows from the axioms of Fig. 1 continue to hold.

We now have to add axioms to govern the behaviour of $\sqcap$. To do so, it is convenient to introduce the "refinement relation" $\sqsubseteq$ on terms, defined by $E \sqsubseteq F \equiv (E \sqcap F \equiv E)$. In the context of **E**$\sqcap$ we can think of $E \sqsubseteq F$ (read "E is refined by F" or "F refines E") as encoding "any outcome of F is a possible outcome of E". For the present, $\sqsubseteq$ provides no more than a convenient shorthand for axiomatisation purposes, but it turns out to have a pivotal role in deriving programs from specifications because it captures the idea of "customer satisfaction": If a customer gives us a term E which is a specification (meaning that it makes use of constructs such as $(\sqcap x{:}\mathbf{Z} \bullet x^2 \leqslant 10 < (x+1)^2)$ which are not mechanically executable), and asks us as programmers to create an executable derivative, he will have no grounds for complaint if we offer him a term F employing only implementable components and satisfying $E \sqsubseteq F$. The axioms for **E**$\sqcap$ are those given in Figs 1 and 4.

Axiom indiv-$\sqsubseteq$ asserts that as far as the individuals of a type are concerned, $\sqsubseteq$ is just equivalence. Actually we only require that the booleans be so ordered but in practice all types will have this ordering and so we choose to build it in for convenience. In future developments, we will introduce constructed types (such as function types) for which the orderings will be more complex. The axiom of the excluded miracle precludes the introduction of the empty term which yields nothing, not even $\bot$. Such a term is not without its mathematical uses — for

example, it would be a unit of $\Box$ — and in another context we might choose to admit it. Note that some authors refer to $P \vee \neg P \vee \neg \Delta P$ as "the excluded fourth". Given our collection of axioms, $P \vee \neg P \vee \neg \Delta P$ does *not* suffice of itself to exclude four truth values, and indeed we shall see that $P \vee \neg P \vee \neg \Delta P$ holds in the four-valued logic below. Of course, $P \vee \neg P \vee \neg \Delta P$ is a theorem of both **E3** and **E**$\Box$. One might have expected to see axioms governing the interaction of $\Box$ with the quantifiers; for example, is $(\forall x{:}T \bullet P) \Box Q \equiv (\forall x{:}T \bullet P \Box Q)$ a theorem if x is not free in Q? (It is.) It turns out that we can deduce the behaviour of the quantifiers with respect to $\Box$ from the given axioms, primarily because we can resort if necessary to the brute force method of proof by truth cases:

$$\text{Truth Cases} \quad \frac{(P \equiv \mathit{True}) \equiv (Q \equiv \mathit{True}) \qquad (P \equiv \mathit{False}) \equiv (Q \equiv \mathit{False})}{P \equiv Q}$$

This is a derived inference rule of both **E3** and **E**$\Box$.

Whereas in **E3**, Δ captures the idea of a term being defined or not, in **E**$\Box$ it captures the idea of a term being determined or not. A term is "undetermined" if it contains a choice which cannot be eliminated, and otherwise it is "determined". For example, *True* is determined, while $\mathit{True} \Box \mathit{False}$ is undetermined.

The model theory for **E**$\Box$ is a simplified version of that for **E4** which is presented below, and so for brevity we omit it. All the properties of $\Box$ postulated above are derivable, as well as the following sample theorems:

$\sqsubseteq$ is a partial order

$\Delta(E \sqsubseteq F)$

$(E \sqsubseteq F) \equiv (\forall x{:}T \bullet F \sqsubseteq x \Rightarrow E \sqsubseteq x)$

$E \sqsubseteq F \Box G \equiv E \sqsubseteq F \wedge E \sqsubseteq G$

$\Delta(E \Box F) \equiv \Delta E \wedge (E \equiv F)$

$(P \Box Q \equiv \mathit{True}) \equiv (P \equiv \mathit{True}) \wedge (Q \equiv \mathit{True})$

$(P \Box Q \equiv \mathit{False}) \equiv (P \equiv \mathit{False}) \wedge (Q \equiv \mathit{False})$

$\mathit{True} \not\equiv \mathit{True} \Box \mathit{False}$

$\mathit{False} \not\equiv \mathit{True} \Box \mathit{False}$

$P \sqsubseteq Q \equiv \neg \Delta P \vee (P \equiv Q)$

$P \sqsubseteq \mathit{True} \equiv (P \not\equiv \mathit{False})$

$(\forall x{:}T \bullet P \sqsubseteq Q) \Rightarrow ((\forall x{:}T \bullet P) \sqsubseteq (\forall x{:}T \bullet Q))$

$(\forall x{:}T \bullet P) \sqsubseteq \mathit{True} \equiv (\forall x{:}T \bullet P \sqsubseteq \mathit{True})$

$E = F \Rightarrow (E \equiv F)$

$=$ distributes over $\Box$

The truth of $\Delta(E \Box F) \equiv \Delta E \wedge (E \equiv F)$ relies on axiom indiv-$\sqsubseteq$. Without this axiom, we can prove the weaker $\Delta(E \Box F) \equiv (\Delta E \wedge E \sqsubseteq F) \vee (\Delta F \wedge F \sqsubseteq E)$.

It follows from the axiom of the excluded fourth and the provable fact that $\mathit{True} \Box \mathit{False}$ differs from both *True* and *False* that the logic has precisely three values.

4. E4: Partiality and Nondeterminacy

We now merge **E3** and **E**$\Box$ to yield a logic catering for both undefined and undetermined terms; we call the resulting logic **E4**. We shall refer to terms which are either undefined or undetermined as "improper"; the remaining terms are said to be "proper". We shall need an operator τ (on all types) to distinguish between undetermined terms and undefined terms. Intuitively, τE (read "E is defined") holds iff E is a proper value or a choice among proper values, while ΔE (read now "E is proper") holds iff E denotes a value that is neither undefined nor a choice among values. For the booleans:

	τ	Δ
True	*True*	*True*
False	*True*	*True*
True $\Box$ *False*	*True*	*False*
$\perp$	*False*	*False*

The behaviour of the boolean connectives is as for **E3** and **E**$\Box$, with the addition of the following to fix the way in which *True* $\Box$ *False* and $\perp$ behave in combination:

$\wedge$	*True* $\Box$ *False*	$\perp$
True $\Box$ *False*	*True* $\Box$ *False*	$\perp$
$\perp$	$\perp$	$\perp$

$\vee$	*True* $\Box$ *False*	$\perp$
True $\Box$ *False*	*True* $\Box$ *False*	$\perp$
$\perp$	$\perp$	$\perp$

$\Rightarrow$	*True* $\Box$ *False*	$\perp$
True $\Box$ *False*	*True*	*True*
$\perp$	*True*	*True*

$=$	*True* $\Box$ *False*	$\perp$
True $\Box$ *False*	*True* $\Box$ *False*	$\perp$
$\perp$	$\perp$	$\perp$

$\equiv$	*True* $\Box$ *False*	$\perp$
True $\Box$ *False*	*True*	*False*
$\perp$	*False*	*True*

The axioms for **E4** are those given in Figs 1 and 5. The axiom of 3-defined asserts that *True*, *False*, and *True* $\Box$ *False* account for all the defined booleans. As in **E3**, it is provable that these values differ from one another. Together with axiom one-$\perp$ we infer that there is at most one other truth value. If we want to guarantee precisely four truth values we can formally introduce $\perp_{\mathbf{B}}$ of type boolean by the axiom $\neg\tau\perp_{\mathbf{B}}$.

It follows from the axioms that in quantifiers dummies range over proper values, not over choices or $\perp$. It also follows that $E \sqsubseteq F$ holds if E is $\perp$ (or, as before, if each outcome of F is a possible outcome of E). The theorems of **E4** include those listed below (it includes some that have previously been presented as theorems of **E3**):

$\Box$ is symmetric, associative, and idempotent

$\sqsubseteq$ is a partial order

$\Delta(E \sqsubseteq F)$

$\Delta\tau E$

$\neg\tau E \Rightarrow E \sqsubseteq F$

	Choice
$\sqsubseteq$-definition:	$E \sqsubseteq F \equiv (E \sqcap F \equiv E)$
$\equiv - \sqsubseteq$:	$(E \equiv F) \equiv (\tau E \equiv \tau F) \wedge (\forall x{:}T \bullet E \sqsubseteq x \equiv F \sqsubseteq x)$, where E, F have type T
$\sqcap$-definition:	$E \sqcap F \sqsubseteq x \equiv E \sqsubseteq x \vee F \sqsubseteq x$
	Refinement ordering
indiv-$\sqsubseteq$:	$(\forall x, y{:}T \bullet x \sqsubseteq y \equiv (x \equiv y))$
	Constants and variables
constants defined:	τc
variables defined:	τx
	Distributivity of connectives
$\neg/\sqcap$:	$\neg(P \sqcap Q) \equiv \neg P \sqcap \neg Q$
$\vee/\sqcap$:	$P \vee (Q \sqcap R) \equiv (P \vee Q) \sqcap (P \vee R)$
$\tau/\sqcap$:	$\tau(E \sqcap F) \equiv \tau E \wedge \tau F$
$\tau/\forall$:	$\tau(\forall x{:}T \bullet P) \equiv (\forall x{:}T \bullet \tau P) \vee (\exists x{:}T \bullet \neg P \wedge \Delta P)$
	Distributivity of operator symbols
strictness:	$\tau \mathbf{f}(..., E, ...) \Rightarrow \tau E$
$\sqcap$-distributivity:	$\mathbf{f}(..., E \sqcap F, ...) \equiv \mathbf{f}(..., E, ...) \sqcap \mathbf{f}(..., F, ...)$
	Extremes
3-defined:	$\neg \tau P \vee \Delta P \vee (P \equiv \mathit{True} \sqcap \mathit{False})$
one-$\perp$:	$\tau E \vee \tau F \vee (E \equiv F)$
excluded miracle:	$(\exists x{:}T \bullet E \sqsubseteq x)$, where E has type T
	Equality
$= - \mathit{True}$:	$(E = F \equiv \mathit{True}) \equiv (\forall x{:}T \bullet E \sqsubseteq x \equiv (E \equiv x)) \wedge E \equiv F$, where E, F have type T
$= - \mathit{False}$:	$(E = F \equiv \mathit{False}) \equiv \neg(\exists x{:}T \bullet E \sqsubseteq x \wedge F \sqsubseteq x)$, where E, F have type T
$\tau/=$:	$\tau(E = F) \equiv \tau E \wedge \tau F$

Fig. 5. Additional axioms for **E4**.

$$\tau E \vee (\forall x{:}T \bullet E \sqsubseteq x)$$

$$(E \sqsubseteq F) \equiv (\tau E \Rightarrow \tau F) \wedge (\forall x{:}T \bullet F \sqsubseteq x \Rightarrow E \sqsubseteq x)$$

$$E \sqsubseteq F \sqcap G \equiv E \sqsubseteq F \wedge E \sqsubseteq G$$

$$\Delta(E \sqcap F) \equiv (\Delta E \wedge E \equiv F)$$

$$(\forall x{:}T \bullet P \sqsubseteq Q) \Rightarrow ((\forall x{:}T \bullet P) \sqsubseteq (\forall x{:}T \bullet Q))$$

$$(E \sqsubseteq F) \Rightarrow (\mathbf{f}(..., E, ...) \sqsubseteq \mathbf{f}(..., F, ...))$$

$$(P \sqcap Q \equiv \mathit{True}) \equiv (P \equiv \mathit{True}) \wedge (P \equiv \mathit{True})$$

$$P \sqsubseteq Q \equiv \neg \Delta P \vee (P \equiv Q)$$

$$(\forall x{:}T \bullet P) \sqsubseteq \mathit{True} \equiv (\forall x{:}T \bullet P \sqsubseteq \mathit{True})$$

$$\tau P \vee \tau Q \vee \neg \tau(P \vee Q)$$

$$\tau P \equiv (P \equiv \mathit{True}) \vee (P \equiv \mathit{False}) \vee (P \equiv \mathit{True} \sqcap \mathit{False})$$

$$(P \equiv Q) \equiv (\tau P \equiv \tau Q) \wedge (\Delta P \equiv \Delta Q) \wedge (\Delta P \wedge \Delta Q \Rightarrow P \equiv Q)$$

$E = F \Rightarrow (E \equiv F)$

$=$ distributes over $\Box$

Note that theorems of **E3** or **E**$\Box$ are not necessarily theorems of **E4**. For example, $(\forall x{:}T \bullet P) \Box Q \equiv (\forall x{:}T \bullet P \Box Q)$ where x is not free in Q, is a theorem of **E**$\Box$ but not of **E4**.

When constructing theories using **E4** we introduce data type axioms in the usual way, except that we use τ to capture partiality. For example, an axiomatisation of the integers might include $\tau(E \div F) \equiv \tau E \wedge \neg(F \sqsubseteq 0)$.

True, *False*, *True* $\Box$ *False*, and $\bot$ are precisely the truth values we need for a total correctness calculus. However, we can imagine other programming calculi in which $\bot \Box$ *False* and $\bot \Box$ *True* are distinct from $\bot$, and indeed from $\bot \Box$ *True* $\Box$ *False*. We surmise that such multi-valued logics can be accommodated within our framework, and in particular that the meaning of the boolean connectives can be extended in a reasonable way such that the theorems following from the axioms of Fig. 1 continues to hold.

5. Model Theory for E4

The proof system of **E4** is based on an interpretation of types and terms with respect to a given set D containing at least two elements. Such a set is called the "domain of interpretation", or simply the "domain". We use $\mathscr{P}D$ to denote the set of non-empty subsets of D. We use $\mathscr{P}_\bot D$ to denote the set $\mathscr{P}D \cup \{\bot\}$ where $\bot$ is a distinguished element that is not a member of D.

Each type symbol T is interpreted as an element $[T]$ of $\mathscr{P}D$. $[\mathbf{B}]$ is required to be a two-element set whose members we will denote by *tt* and *ff*. We write $\mathscr{P}T$ to denote the set of non-empty subsets of $[T]$, and $\mathscr{P}_\bot T$ to denote $\mathscr{P}T \cup \{\bot\}$. For example, $\mathscr{P}_\bot \mathbf{B}$ is $\{\bot, \{tt,ff\}, \{tt\}, \{ff\}\}$. Each expression E of type T is interpreted as an element $[E]$ of $\mathscr{P}_\bot T$. Interpretations of constants (of type T, say) are constrained to be singleton subsets of $[T]$. We denote this set by $\mathscr{V}T$; for example $\mathscr{V}\mathbf{B}$ is $\{\{tt\}, \{ff\}\}$. *True* is interpreted as $\{tt\}$, and *False* as $\{ff\}$.

Binary choice is interpreted as the 2-place function on $\mathscr{P}_\bot D$ that yields the "strict union" of its arguments. The strict union of a non-empty subset of $\mathscr{P}_\bot D$ is $\bot$ if the subset contains $\bot$, and otherwise it is just normal set union.

The interpretations of the $\wedge$, $\vee$, and $\Rightarrow$ are 2-place functions on $\mathscr{P}_\bot D$ that are in agreement with the truth tables given earlier when both their arguments are elements of $\mathscr{P}_\bot \mathbf{B}$. Similarly, $\neg$ is a 1-place function in agreement with its truth tables. We denote such interpreting functions by $[\wedge]$, $[\vee]$, etc. Observe that $[\wedge]$ behaves as a minimisation on $\mathscr{P}_\bot \mathbf{B}$ with respect to the total order $<_\wedge$ defined:

$$\{ff\} <_\wedge \bot <_\wedge \{tt,ff\} <_\wedge \{tt\}.$$

Similarly, $[\vee]$ behaves as a minimisation with respect to the total order $<_\vee$ defined:

$$\{tt\} <_\vee \bot <_\vee \{tt,ff\} <_\vee \{ff\}$$

The interpretation of $(\forall x{:}T \bullet P)$ is the minimum of $[P]$ with respect to $<_\wedge$ when the interpretation of x ranges over $\mathscr{V}T$. To interpret existential quantification, the order $<_\vee$ is used. Similarly, the interpretation of $(\forall x{:}T \mid R \bullet P)$ is the minimum of $[P]$ with respect to $<_\wedge$ when the interpretation of x ranges over those elements

of $\mathscr{V}T$ for which $[R]$ is $\{tt\}$. The interpretation of $(\exists x{:}T \mid R \bullet P)$ is similar except the order $<_\vee$ is used instead of $<_\wedge$.

Equality $(=)$ is interpreted by the 2-place function on $\mathscr{P}_\perp D$ that yields $\perp$ if one of its arguments is itself $\perp$, and otherwise, when its arguments are the sets S and T, it yields the (non-empty) set $\{x \in S, y \in T \bullet \textbf{if } x = y \textbf{ then } tt \textbf{ else } ff\}$. Here we re-use the symbol "=" to stand for primitive equality in the domain. The interpretation of $\equiv$ is the 2-place function on $\mathscr{P}_\perp D$ that yields $\{tt\}$ if its arguments are identical, and $\{ff\}$ otherwise. The interpretation of $\sqsubseteq$ is the 2-place function on $\mathscr{P}_\perp D$ which yields $\{tt\}$ if either the first argument is $\perp$, or else neither argument is $\perp$ and the second is a subset of the first; in all other cases it yields $\{ff\}$. Both τ and Δ are interpreted as one-place functions on $\mathscr{P}_\perp D$ — τ maps $\perp$ to $\{ff\}$ and all other arguments to $\{tt\}$; Δ maps singletons to $\{tt\}$ and everything else to $\{ff\}$.

Every operator symbol **f** is interpreted by a "matching" function $[\mathbf{f}]$ on $\mathscr{P}_\perp D$. By "matching" we mean that $[\mathbf{f}]$ takes $arity(\mathbf{f})$ arguments, and that if each i'th argument of $[\mathbf{f}]$ is an element of $\mathscr{P}_\perp T$ where T is **f**'s i'th argument type, for i in the range 1 to $arity(\mathbf{f})$ inclusive, the result is an element of $\mathscr{P}_\perp U$ where U is **f**'s result type. We require firstly that $[\mathbf{f}]$ be $\perp$-strict in every argument, i.e. $[\mathbf{f}](..., \perp, ...) = \perp$, and secondly that it be strictly additive in every argument, i.e. $[\mathbf{f}](..., S, ..)$ is equivalent to the strict union of $\{x \in S \bullet [\mathbf{f}](..., \{x\}, ...)\}$.

Terms are interpreted by induction on their structure.

A "valid interpretation" is a domain D and a mapping from types to elements of $\mathscr{P}D$, and from expressions without free variables to $\mathscr{P}_\perp D$ that respects the requirements set out above. A valid interpretation extends naturally to expressions with free variables by giving each variable symbol an interpretation. Interpretations of variables (of type T, say), are constrained to be elements of $\mathscr{V}T$. A mapping from the set of variable symbols to their respective interpretations that meets this requirement is called an "environment", and we refer to the interpretation of expressions "in" that environment. It follows that for every valid interpretation and in every environment, every expression (possibly with free variables) has an interpretation.

We want **E4** to classify each formula as a theorem if and only if every valid interpretation of it is $\{tt\}$ in every environment. A logic which satisfies the "if" part of the preceding statement is said to be "sound", and one that satisfies the "only if" part it is said to be "complete". It is relatively routine to show soundness, by checking that each axiom is interpreted as $\{tt\}$ for every valid interpretation and in every environment, and that this is preserved by the inference rules. We do not offer a completeness proof. Although desirable, completeness proofs are not as important in programming logics as in traditional logical studies. Other matters take precedence, such as discovering a set of theorems that are useful in programming practice, and learning how best to apply them. Indeed, many investigators are content to live with incompleteness if the practical utility of the logic is established. For example, the creators of the Raise specification language (RSL) devote a large volume to the formal underpinnings of their language [Mil90], and yet "The RSL proof rules are intended to be sound, but not necessarily complete with respect to the semantics of RSL." [Rai95, p.267]. This attitude is echoed in a NASA study of the use of formal methods in the space program [NAS97, page 80]: "In general, completeness has theoretical importance for logicians, but less importance for those working in formal methods. It is quite difficult to establish completeness for systems of any complexity, and many interesting and even important formal systems are provably incomplete". That

said, we do not undervalue the attraction of a completeness proof. We have already shown the completeness of **E3** in [MoB98].

6. Conclusion

Our concern has been with expressions whose outcomes are not confined to simple values, but which may have no defined outcome, or which may yield any of several outcomes. These are the sort of expressions we meet when we derive programs from formal specifications, or when we prove the correctness of programs with respect to formal specifications. Reasoning about such expressions requires a logic in which formulae may have the status "true", "false", "neither true nor false", or "possibly true, possibly false"; we have presented such a logic. The logic preserves most of the useful theorems of classical 2-valued predicate calculus, while allowing us to reason equationally in the style of [DiS90] and [GrS93].

With respect to deriving programs from formal specifications, many authors have appreciated the value of nondeterminacy on the one hand, and refinement at the level of expressions on the other. Unfortunately, a satisfactory combination of both features has proved elusive. For various approaches see [Par90], [NoH93], [LaH96], [Rai92], [Rai95], [WaM95], and [War94].

The language of the CIP development method, CIP-L [Par90], has constructs similar to our choice, equivalence (called "strong equality"), refinement ("descendancy"), τ and Δ. However, its nondeterminacy is erratic ($\bot$ is not a zero of choice) rather than demonic, which is arguably less useful for the kind of underspecification encountered in specifications. All of strong equality, descendancy, τ, and Δ are part of the meta-language used to reason about programs, whereas in our calculus there is only one language level. We shall argue the advantage of our approach below. The logical combinators within the language (e.g. conjunction and universal quantification) are strict, and so are less than adequate for writing and reasoning about specifications (see [Bli88]).

Norvell and Hehner [NoH93] present a refinement calculus for expressions in which they seek to accommodate partiality and nondeterminacy. Their propositions are boolean expressions, but only deterministic ones are "acceptable" in the logic. The authors are aware of this shortcoming and list as future work "allowing nondeterministic predicates without complicating the laws". This paper presents just such a logic.

Larsen and Hansen [LaH96] present a denotational semantics for a functional language with what they call "under-determinism". The type language includes comprehension-types of the form $\{x{:}T \mid P \bullet E\}$, and the expression language is extended by **choice** T, which under-deterministically selects an element of the type T. The proof system is based on generalised type inference, with propositions of the form $E{:}T$, where E is an expression and T a type, asserting that the possible outcomes of E are included in T. They do not supply a formal logic by which one reasons about such assertions, but presume the availability of logical connectives that have a few simple properties. Unfortunately the properties they postulate include strictness, an approach that we don't consider appropriate for program refinement. On the contrary, we surmise that the logic presented in this paper is the kind of logic that might better underpin their work.

In the program development methodology RAISE [Rai92, Rai95] the central concept is that of a "class", which is a collection of declarations of names and

axioms about them. Each class generates a theory, and implementation is treated as theory extension. Refinement is at the level of classes and the logic is two-valued. The RAISE language also contains nondeterministic expressions, but most of the proof rules given in [Rai95] do not apply to them. In contrast, our work provides the logical basis for nondeterminacy at the level of expressions, and potentially could be used as the logic underlying RAISE.

Walicki and Meldal's work [WaM95] treats nondeterministic operators in the context of algebraic specifications. It gives completeness results with respect to a set-based semantics and a computational semantics, in which specifications of nondeterministic functions are transformed to under-specifications of deterministic functions. In contrast, our work treats nondeterminacy and partiality at the level of expressions in the logic (in [WaM95] these are always determined). Their relations $\prec$ and $\asymp$ roughly correspond to our refinement and equivalence, respectively, and their $E \doteq F$ and $E\#F$ correspond to our $E = F \equiv \mathit{True}$, and $E = F \equiv \mathit{False}$, respectively.

Ward's thesis [War94] presents a specification and programming language based on functions, including demonic and angelic nondeterminacy (i.e., where $\bot$ is a unit of choice). However, the language is not accompanied by a proof theory. Ward makes no claim that the given refinement laws are sufficient in practice. When a new refinement law is needed, the user must rely on the given semantics to confirm its validity. As far as the booleans are concerned, Ward restricts boolean expressions to be so-called "value-expressions" which in effect bans nondeterministic propositions, but still allows undefined ones arising from recursion. However, no logical axioms governing such propositions are presented.

When authors admit nondeterminacy at the level of expressions, they often have to resort to unnatural devices to keep the nondeterminacy at bay, usually with the consequence that programmers are forced to be unnecessarily conscious of whether expressions are nondeterministic or not, and indeed with the consequence that they are obliged to prove many side conditions concerning determinacy. Typically no formal logic is provided which supports reasoning in the presence of nondeterminacy, with the consequence that the programmer is forced to reason at the level of semantics. Limiting the domain of nondeterminacy seems almost impossible: once we let it out of the bottle, it is inevitably going to spill into the logical assertions with which we reason about programs. The best way forward is to accept it, and modify our formal logic to accommodate it.

The logic we have presented paves the way to constructing a genuinely deductive calculus of programming which fully embraces nondeterminacy. In this calculus, there seems little need to distinguish between the language in which we express properties of specifications, and the language of specifications themselves. A statement about programs, even one as exotic as $E \sqsubseteq F$, is itself a term, and not a meta-statement. This uniformity should facilitate the smooth transition of specifications to programs, because propositions can migrate effortlessly into specifications/programs, as they often want to when we formally extract programs from specifications. For example, suppose we wish to implement a specification E, and we succeed in proving that the refinement $E \sqsubseteq F$ is valid provided some assumption P (a proposition) holds. (The proof will be carried out in **E4**, not in some "meta-logic".) Suppose we also succeed in proving that $E \sqsubseteq G$ is valid if $\neg P$ holds. By the deduction theorem we will have established

$$P \Rightarrow (E \sqsubseteq F)$$

$$\neg P \Rightarrow (E \sqsubseteq G)$$

In our refinement calculus we will then be able to conclude immediately that

$$E \sqsubseteq \textbf{if } P \textbf{ then } F \textbf{ else } G$$

— P has become a boolean expression. We can then proceed to attack P to reduce it to an executable equivalent.

We leave it to forthcoming papers to show how **E4** combines with programming and specification elements such as unbounded choice, conditional expressions, function definition, function application, recursion, recursive types, state and state-changing commands, etc.

Acknowledgements

We appreciate the very helpful comments of two anonymous referees which led to several issues being clarified, and to a large improvement in the presentation.

References

[Avr88] Avron, A.: Foundations and proof theory of 3-valued logics. LFCS report series ECS-LFCS-88-48, Laboratory for the Foundations of Computer Science, Edinburgh University, 1988.

[Avr91] Avron, A.: Natural 3-valued logics — characterisation and proof theory. *Journal of Symbolic Logic*, 56:276–294, 1991.

[Bli88] Blikle, A.: Three-valued predicates for software specification and development. In R. et al. Bloomfield, editor, *VDM — the way ahead*, volume 328 of *Lecture Notes in Computer Science*, pages 243–266. Springer Verlag, 1988.

[DiS90] Dijkstra, E. W. and Scholten, C. S.: *Predicate Calculus and Program Semantics*. Springer Verlag, New York, 1990.

[GrS93] Gries, D. and Schneider, F. B.: *A Logical Approach to Discrete Math*. Springer Verlag, New York, 1993.

[JoM94] Jones, C. B. and Middelburg, C. A.: A typed logic of partial functions reconstructed classically. *Acta Informatica*, 31(5):399–430, 1994.

[LaH96] Larsen, P. G. and Hansen, B. S.: Semantics of under-determined expressions. *Formal Aspects of Computing*, 8(1):47–66, 1996.

[MoB98] Morris, J. M. and Bunkenburg, A.: E3: A logic for reasoning equationally in the presence of partiality. To be published., 1998.

[Mil90] Milne, R. E.: The semantic foundations of the Raise specification language. RAISE/STC/REM/11, STC/STL, Harlow, UK, 1990. Technical Report.

[Mon67] Monteiro, A.: Construction des algèbres de Lukasiewicz trivalentes dans les algèbres de Boole monadiques I. *Mathematica Japonica*, 12:1–23, 1967.

[NAS97] NASA Office of Safety and Mission Assurance. *Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems, Vol. II: A Practicioner's Companion, release 1.0*, May 1997. NASA-GB-001-97.

[NoH93] Norvell, T. S. and Hehner, E. C. R.: Logical specifications for functional programs. In *Proceedings of the Second International Conference on Mathematics of Program Construction, Oxford, 29 June - 3 July 1992*, volume 669 of *Lecture Notes in Computer Science*, pages 269–290. Springer Verlag, 1993.

[Par90] Partsch, H. A.: *Specification and Transformation of Programs*. Springer Verlag, New York, 1990.

[Rai92] The Raise Language Group. *The RAISE Specification Language*. Prentice-Hall, London, 1992.

[Rai95] The RAISE Method Group. *The RAISE Development Method*. Prentice-Hall, London, 1995.

[War94] Ward, N.: *A Refinement Calculus for Nondeterministic Expressions*. PhD thesis, University of Queensland, 1994.
[WaM95] Walicki, M. and Meldal, S.: A complete calculus for the multialgebraic and functional semantics of nondeterminism. *ACM Transactions on Programming Languages and Systems*, 17:366–393, 1995.

A. Appendix: Some theorems which follow from the axioms of Fig. 1

Equivalence and negation

$\equiv$-reflexivity:	$E \equiv E$
$\equiv$-symmetry:	$(E \equiv F) \equiv (F \equiv E)$
$\equiv$-trans:	$(P \equiv Q) \wedge (Q \equiv R) \Rightarrow (P \equiv R)$
exchange:	$(\neg P \equiv Q) \equiv (\neg Q \equiv P)$
$\neg$-exch:	$(\neg P \equiv Q) \equiv (P \equiv \neg Q)$
$\neg$-invol:	$\neg\neg P \equiv P$
$\equiv$-mirror:	$(P \equiv Q) \equiv (\neg P \equiv \neg Q)$
$\not\equiv$-definition:	$(E \not\equiv F) \equiv \neg(E \equiv F)$
strong $\equiv$:	$(E \equiv F) \vee (E \not\equiv F)$

Disjunction and conjunction

$\vee$-symmetry:	$P \vee Q \equiv Q \vee P$
$\vee$-associativity:	$P \vee (Q \vee R) \equiv (P \vee Q) \vee R$
$\vee$-idempotency:	$P \vee P \equiv P$
$\wedge$-definition:	$P \wedge Q \equiv \neg(\neg P \vee \neg Q)$
consistency:	$(P \wedge Q \equiv P) \equiv (P \vee Q \equiv Q)$
de Morgan:	$\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$
de Morgan:	$\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$
$\wedge$-symm:	$(P \wedge Q) \equiv (Q \wedge P)$
$\wedge$-assoc:	$P \wedge (Q \wedge R) \equiv (P \wedge Q) \wedge R$
$\wedge$-idem:	$(P \wedge P) \equiv P$
$\wedge/\vee$:	$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$
$\vee/\wedge$:	$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$
absorption:	$P \wedge (P \vee Q) \equiv P$
absorption:	$P \vee (P \wedge Q) \equiv P$

Substitution

$\Rightarrow$-subst:	$(E \equiv F) \Rightarrow P[x := E] \equiv (E \equiv F) \Rightarrow P[x := F]$
$\wedge$-subst:	$(E \equiv F) \wedge P[x := E] \equiv (E \equiv F) \wedge P[x := F]$
cond $\Rightarrow$-subst:	$(P \Rightarrow (E \equiv F)) \Rightarrow (P \Rightarrow Q[x := E] \equiv P \Rightarrow Q[x := F])$
cond $\wedge$-subst:	$(P \Rightarrow (E \equiv F)) \Rightarrow$ $((P \equiv \mathit{True}) \wedge Q[x := E] \equiv (P \equiv \mathit{True}) \wedge Q[x := F])$

True and False

true:	$True$
$\vee$-zero:	$P \vee True \equiv True$
$\wedge$-zero:	$P \wedge False \equiv False$
$\vee$-unit:	$P \vee False \equiv P$
$\wedge$-unit:	$P \wedge True \equiv P$
$\vee$-truth:	$((P \vee Q) \equiv True) \equiv (P \equiv True) \vee (Q \equiv True)$
$\wedge$-truth:	$(P \wedge Q \equiv True) \equiv (P \equiv True) \wedge (Q \equiv True)$
$\vee$-falsity:	$(P \vee Q \equiv False) \equiv (P \equiv False) \wedge (Q \equiv False)$
$\wedge$-falsity:	$(P \wedge Q \equiv False) \equiv (P \equiv False) \vee (Q \equiv False)$
$\equiv$-truth:	$((E \equiv F) \equiv True) \equiv (E \equiv F)$
$\equiv$-Falsity:	$((E \equiv F) \equiv False) \equiv (E \not\equiv F)$
False-definition:	$False \equiv \neg True$
two values:	$False \not\equiv True$
excl false:	$(P \vee \neg P) \not\equiv False$

Implication

$\Rightarrow$-definition:	$P \Rightarrow Q \equiv (P \not\equiv True) \vee Q$
Leibniz:	$(E \equiv F) \Rightarrow (G[x := E] \equiv G[x := F])$
$\Rightarrow$-reflex:	$P \Rightarrow P$
$\Rightarrow$-conn:	$(P \Rightarrow Q) \vee (Q \Rightarrow P)$
$\Rightarrow$-left-unit:	$(True \Rightarrow P) \equiv P$
$\Rightarrow$-right-zero:	$(P \Rightarrow True) \equiv True$
$\Rightarrow/\vee$:	$P \Rightarrow Q \vee R \equiv (P \Rightarrow Q) \vee (P \Rightarrow R)$
$\Rightarrow/\wedge$:	$P \Rightarrow Q \wedge R \equiv (P \Rightarrow Q) \wedge (P \Rightarrow R)$
$\Rightarrow/\Rightarrow$:	$P \Rightarrow (Q \Rightarrow R) \equiv (P \Rightarrow Q) \Rightarrow (P \Rightarrow R)$
$\Rightarrow/\equiv$:	$P \Rightarrow (Q \equiv R) \equiv ((P \Rightarrow Q) \equiv (P \Rightarrow R))$
$\vee$-lub:	$P \vee Q \Rightarrow R \equiv (P \Rightarrow R) \wedge (Q \Rightarrow R)$
$\Rightarrow/\equiv/\wedge$:	$P \Rightarrow (Q \equiv R) \equiv (((P \equiv True) \wedge Q) \equiv ((P \equiv True) \wedge R))$
$\equiv$-weakening:	$(P \equiv Q) \Rightarrow (P \Rightarrow Q))$
weakening:	$P \Rightarrow P \vee Q$
weakening:	$P \wedge Q \Rightarrow P$
shunting:	$P \wedge Q \Rightarrow R \equiv P \Rightarrow (Q \Rightarrow R)$
$\Rightarrow$-trans:	$(P \Rightarrow Q) \wedge (Q \Rightarrow R) \Rightarrow (P \Rightarrow R)$
$\equiv/\Rightarrow$-trans:	$(P \equiv Q) \wedge (Q \Rightarrow R) \Rightarrow (P \Rightarrow R)$
$\equiv/\Rightarrow$-trans:	$(P \Rightarrow Q) \wedge (Q \equiv R) \Rightarrow (P \Rightarrow R)$
modus ponens:	$P \wedge (P \Rightarrow Q) \Rightarrow Q$
True-$\Rightarrow$:	$(P \equiv True) \Rightarrow Q \equiv P \Rightarrow Q$
$\Rightarrow$-truth:	$((P \Rightarrow Q) \equiv True) \equiv P \Rightarrow (Q \equiv True)$

Monotonicity

$\Rightarrow$-right-mono:	$(P \Rightarrow Q) \Rightarrow ((R \Rightarrow P) \Rightarrow (R \Rightarrow Q))$
$\Rightarrow$-left-anti-mono:	$(P \Rightarrow Q) \Rightarrow ((Q \Rightarrow R) \Rightarrow (P \Rightarrow R))$
$\Rightarrow$-comb-$\wedge$:	$(P \Rightarrow Q) \wedge (R \Rightarrow S) \Rightarrow (P \wedge R \Rightarrow Q \wedge S)$
$\Rightarrow$-comb-$\vee$:	$(P \Rightarrow Q) \wedge (R \Rightarrow S) \Rightarrow (P \vee R \Rightarrow Q \vee S)$
$\wedge$-mono:	$(P \Rightarrow Q) \Rightarrow ((P \wedge R) \Rightarrow (Q \wedge R))$
$\vee$-mono:	$(P \Rightarrow Q) \Rightarrow ((P \vee R) \Rightarrow (Q \vee R))$

Theorems requiring defined terms

$\vee/\equiv$:	$(P \vee (Q \equiv R)) \equiv ((P \vee Q) \equiv (P \vee R))$,	if ΔP
absorption:	$P \wedge (\neg P \vee Q) \equiv P \wedge Q$,	if ΔP
absorption:	$P \vee (\neg P \wedge Q) \equiv P \vee Q$,	if ΔP
absorption:	$P \wedge (P \Rightarrow Q) \equiv (P \wedge Q)$,	if ΔP
Shannon:	$Q[x := P] \equiv (P \wedge Q[x := \mathit{True}]) \vee (\neg P \wedge Q[x := \mathit{False}])$,	if ΔP
$\Rightarrow$-$\vee$:	$P \Rightarrow Q \equiv (P \wedge Q \equiv P)$,	if ΔP, ΔQ
$\Rightarrow$-$\wedge$:	$P \Rightarrow Q \equiv (P \vee Q \equiv Q)$,	if ΔP, ΔQ
bi-impli:	$(P \Rightarrow Q) \wedge (Q \Rightarrow P) \equiv (P \equiv Q)$,	if ΔP, ΔQ
$\neg$-imp/exp:	$\neg(P \equiv Q) \equiv (\neg P \equiv Q)$,	if ΔP, ΔQ
contrapos:	$(P \Rightarrow Q) \equiv (\neg Q \Rightarrow \neg P)$,	if ΔP, ΔQ
absorption:	$P \wedge (P \equiv Q) \equiv P \wedge Q$,	if ΔP, ΔQ
$\equiv$-assoc:	$((P \equiv Q) \equiv R) \equiv (P \equiv (Q \equiv R))$,	if ΔP, ΔQ, ΔR

Δ

Δ-definition:	$\Delta P \equiv ((P \equiv \mathit{True}) \equiv P)$
$\equiv$-truth:	$\Delta(E \equiv F)$
true defined:	$\Delta \mathit{True}$
$\Delta\Delta$:	$\Delta\Delta E$
estab:	$(P \equiv \mathit{True}) \equiv \Delta P \wedge P$
incl middle:	$\neg\Delta P \vee \neg P \vee P$
Δ-intro:	$P \vee \neg P \Rightarrow \Delta P$
Δ-bool:	$\Delta P \equiv (P \equiv \mathit{True}) \vee (P \equiv \mathit{False})$
Δ-bool:	$\Delta P \equiv \neg(P \Rightarrow \neg P) \vee \neg(\neg P \Rightarrow P))$
$\Delta\neg$:	$\Delta(\neg P) \equiv \Delta P$
$\Rightarrow$-defn:	$(P \Rightarrow Q) \equiv \neg\Delta P \vee \neg P \vee Q$

Monotonicity of quantifications

$\forall/\equiv$:	$(\forall x{:}T \mid R \bullet P \equiv Q) \Rightarrow ((\forall x{:}T \mid R \bullet P) \equiv (\forall x{:}T \mid R \bullet Q))$
$\exists/\equiv$:	$(\forall x{:}T \mid R \bullet P \equiv Q) \Rightarrow ((\exists x{:}T \mid R \bullet P) \equiv (\exists x{:}T \mid R \bullet Q))$
$\forall/\Rightarrow$:	$(\forall x{:}T \mid R \bullet P \Rightarrow Q) \Rightarrow ((\forall x{:}T \mid R \bullet P) \Rightarrow (\forall x{:}T \mid R \bullet Q))$
$\exists/\Rightarrow$:	$(\forall x{:}T \mid R \bullet P \Rightarrow Q) \Rightarrow ((\exists x{:}T \mid R \bullet P) \Rightarrow (\exists x{:}T \mid R \bullet Q))$
$\forall$ range anti-mono:	$(\forall x{:}T \bullet R \Rightarrow S) \Rightarrow ((\forall x{:}T \mid S \bullet P) \Rightarrow (\forall x{:}T \mid R \bullet P))$
$\exists$ range mono:	$(\forall x{:}T \bullet S \Rightarrow R) \Rightarrow ((\exists x{:}T \mid S \bullet P) \Rightarrow (\exists x{:}T \mid R \bullet P))$

Range manipulation

trading $\forall$:	$(\forall x{:}T \mid R \bullet P) \equiv (\forall x{:}T \bullet R \Rightarrow P)$
trading $\exists$:	$(\exists x{:}T \mid R \bullet P) \equiv (\exists x{:}T \bullet (R \equiv \mathit{True}) \wedge P))$
range split $\forall$:	$(\forall x{:}T \mid R \vee S \bullet P) \equiv (\forall x{:}T \mid R \bullet P) \wedge (\forall x{:}T \mid S \bullet P)$
range split $\exists$:	$(\exists x{:}T \mid R \vee S \bullet P) \equiv (\exists x{:}T \mid R \bullet P) \vee (\exists x{:}T \mid S \bullet P)$
interchange:	$(\forall x{:}T \bullet (\forall y{:}U \bullet P)) \equiv (\forall y{:}U \bullet (\forall x{:}T \bullet P))$
renaming:	$(\forall x{:}T \bullet P) \equiv (\forall y{:}T \bullet P[x := y])$, where y is fresh

Truth and falsity

∀-truth:	$((\forall x{:}T \mid R \bullet P) \equiv \mathit{True}) \equiv (\forall x{:}T \mid R \bullet P \equiv \mathit{True})$
∃-truth:	$((\exists x{:}T \mid R \bullet P) \equiv \mathit{True}) \equiv (\exists x{:}T \mid R \bullet P \equiv \mathit{True})$
∀-falsity:	$((\forall x{:}T \mid R \bullet P) \equiv \mathit{False}) \equiv (\exists x{:}T \mid R \bullet P \equiv \mathit{False})$
∃-falsity:	$((\exists x{:}T \mid R \bullet P) \equiv \mathit{False}) \equiv (\forall x{:}T \mid R \bullet P \equiv \mathit{False})$
∀-non-truth:	$((\forall x{:}T \mid R \bullet P) \not\equiv \mathit{True}) \equiv (\exists x{:}T \mid R \bullet P \not\equiv \mathit{True})$
∃-non-truth:	$((\exists x{:}T \mid R \bullet P) \not\equiv \mathit{True}) \equiv (\forall x{:}T \mid R \bullet P \not\equiv \mathit{True})$

Constants

∀-constant:	$(\forall x{:}T \mid R \bullet P) \equiv P \vee (\forall x{:}T \bullet R \not\equiv \mathit{True})$, if x not free in P
∃-constant:	$(\exists x{:}T \mid R \bullet P) \equiv P \wedge (\exists x{:}T \bullet R \equiv \mathit{True})$, if x not free in P
habitation:	$(\exists x{:}T \bullet \mathit{True})$
empty range ∀:	$(\forall x{:}T \mid \mathit{False} \bullet P)$

Distribution

∀/≡:	$(\forall x{:}T \bullet P \equiv Q) \Rightarrow ((\forall x{:}T \bullet P) \equiv (\forall x{:}T \bullet Q))$
∀/∧:	$(\forall x{:}T \mid R \bullet P \wedge Q) \equiv (\forall x{:}T \mid R \bullet P) \wedge (\forall x{:}T \mid R \bullet Q)$
∃/∨:	$(\exists x{:}T \mid R \bullet P \vee Q) \equiv (\exists x{:}T \mid R \bullet P) \vee (\exists x{:}T \mid R \bullet Q)$
∨/∀:	$(\forall x{:}T \mid R \bullet P \vee Q) \equiv P \vee (\forall x{:}T \mid R \bullet Q)$, if x not free in P
∧/∃:	$(\exists x{:}T \mid R \bullet P \wedge Q) \equiv P \wedge (\exists x{:}T \mid R \bullet Q)$, if x not free in P
∀/∨:	$(\forall x{:}T \mid R \bullet P) \vee (\forall x{:}T \mid R \bullet Q) \Rightarrow (\forall x{:}T \mid R \bullet P \vee Q)$
∃/∧:	$(\exists x{:}T \mid R \bullet P \wedge Q) \Rightarrow (\exists x{:}T \mid R \bullet P) \wedge (\exists x{:}T \mid R \bullet Q)$
⇒/∀:	$(\forall x{:}T \mid R \bullet P \Rightarrow Q) \equiv P \Rightarrow (\forall x{:}T \mid R \bullet Q)$, if x not free in P
∧/∀:	$(\forall x{:}T \mid R \bullet P \wedge Q) \equiv P \wedge (\forall x{:}T \mid R \bullet Q)$, if x not free in P, $(\exists x{:}T \bullet R \equiv \mathit{True})$
∨/∃:	$(\exists x{:}T \mid R \bullet P \vee Q) \equiv P \vee (\exists x{:}T \mid R \bullet Q)$, if x not free in P, $(\exists x{:}T \bullet R \equiv \mathit{True})$
⇒/∃:	$(\exists x{:}T \mid R \bullet P \Rightarrow Q) \equiv P \Rightarrow (\exists x{:}T \mid R \bullet Q)$, if x not free in P, $(\exists x{:}T \bullet R \equiv \mathit{True})$

Instantiation

instantiation:	$(\forall x{:}T \bullet P) \wedge \Delta t \Rightarrow P[x := t]$	
instantiation:	$(\forall x{:}T \bullet P) \wedge P[x := t] \equiv (\forall x{:}T \bullet P)$,	if Δt
instantiation:	$(\exists x{:}T \bullet P) \vee P[x := t] \equiv (\exists x{:}T \bullet P)$,	if Δt
instantiation:	$(\forall x{:}T \bullet P) \vee P[x := t] \equiv P[x := t]$,	if Δt
instantiation:	$(\exists x{:}T \bullet P) \wedge P[x := t] \equiv P[x := t]$,	if Δt

∀-∃

de Morgan:	$\neg(\exists x{:}T \mid R \bullet P) \equiv (\forall x{:}T \mid R \bullet \neg P)$
de Morgan:	$\neg(\forall x{:}T \mid R \bullet P) \equiv (\exists x{:}T \mid R \bullet \neg P)$
∃/∀:	$(\exists x{:}T \mid R \bullet (\forall y{:}U \mid S \bullet P)) \Rightarrow (\forall y{:}U \mid S \bullet (\exists x{:}T \mid R \bullet P))$, if x not free in S, y not free in R.
∀-∧-∃:	$(\forall x{:}T \mid R \bullet P) \wedge (\exists x{:}T \mid R \bullet Q) \Rightarrow (\exists x{:}T \mid R \bullet P \wedge Q)$
∀-∃:	$(\forall x{:}T \mid R \bullet P) \Rightarrow (\exists x{:}T \mid R \bullet P) \equiv (\exists x{:}T \bullet R \equiv \mathit{True})$

One point and shifting

1-point $\forall$: $(\forall x{:}T \mid x \equiv t \bullet P) \equiv P[x := t] \vee \neg\Delta t$,
where x not free in t.

1-point $\exists$: $(\exists x{:}T \mid x \equiv t \bullet P) \equiv P[x := t] \wedge \Delta t$, where x not in t.

shifting $\forall$: $(\forall x{:}T \mid R \bullet P) \equiv (\forall x{:}T \mid R[x := t] \bullet P[x := t])$,
if $(\forall y{:}U \bullet (\exists x{:}T \bullet y \equiv t))$ and $(\forall x{:}T \bullet \Delta t)$

shifting $\exists$: $(\exists x{:}T \mid R \bullet P) \equiv (\exists x{:}T \mid R[x := t] \bullet P[x := t])$,
if $(\forall y{:}U \bullet (\exists x{:}T \bullet y \equiv t))$ and $(\forall x{:}T \bullet \Delta t)$

End

Received July 1996
Accepted in revised form April 1998 by C. B. Jones