Science of Computer Programming 43 (2002) 77–89

Science of Computer Programming

www.elsevier.com/locate/scico

A source of inconsistency in theories of nondeterministic functions

J.M. Morris[a, *], A. Bunkenburg[b, 1]

[a] *School of Computer Applications, Dublin City University, Dublin, Ireland*
[b] *Department of Computing Science, University of Glasgow, Scotland, UK*

Received 9 January 2001; received in revised form 27 June 2001; accepted 10 July 2001

Abstract

Nondeterminacy is a useful feature of specification languages because it allows the customer to express that any of a range of outcomes is acceptable for a particular operation. However, the classical theory of functions becomes considerably complicated and counter-intuitive in the presence of nondeterminacy, and inconsistencies can easily creep in. All this is well known. In this paper, we describe a potential new source of inconsistency when functions and nondeterminacy are combined. We show that some existing theories fall foul of it, and show how to avoid it. The root cause of the problem is the substitution of a variable in a nonmonotonic position when the type of the variable is nonflat. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Program correctness; Programming calculi; Nondeterministic functions

1. Introduction

Nondeterminacy is a useful notion in calculi for specifying and writing computer programs. It allows a specifier to record that the customer is willing to accept a range of behaviours, which gives the implementor some added freedom when designing the implementation. Although it is not a trivial matter to give a formal account of nondeterminacy (see, for example, [5,7,14,15]), at least it is reasonably intuitive in the case of basic types such as the integers. When applied to higher types such as function types, however, our intuition can easily let us down, and laws long regarded as incontrovertible can lead to absurdities. Meertens [6] was among the first to point this out, showing that unconstrained η-equivalence leads to an inconsistency in the presence of nondeterminacy.

We have recently been designing programming calculi which incorporate both nondeterminacy and functions in all their generality [7,8]. Despite what we believed to be

* Corresponding author.
[1] Supported by the Leverhulme Trust.

a well-developed intuition, we were on several occasions surprised to discover at the model-building stage that our fledgling theories were inconsistent. One inconsistency in particular surprised us. We surveyed the literature [1–5,9–15] to see whether other theories had met similar difficulties, and discovered two theories that had indeed fallen into the same trap as we had. The remainder either placed serious restrictions on the use of nondeterministic functions, or they left gaps in the axiomatisation to the extent that consistency remained an open question. We describe the inconsistency and how it can be avoided. For readers familiar with the terminology, we can summarise the root cause of the problem as the substitution of a variable in a nonmonotonic position when the type of the variable is nonflat.

2. The language

We treat a typed specification language. A specification language is a richly expressive superset of a programming language; the added feature of primary interest to us is nondeterminacy. We are only concerned with the terms of the language, not any state-changing commands. The types include the booleans ($\mathbf{B}$), the integers ($\mathbf{Z}$), and functions ($T \to U$). We use the letters T and U to stand for arbitrary types, E, F, etc. to stand for arbitrary terms, and P, Q, etc. to stand for arbitrary *boolean* terms. The boolean constants are *true* and *false*. Some types may be equipped with a boolean-valued equality operator which we denote by $=$, possibly subscripted with the type of its operands. Type $\mathbf{Z}$ is not actually necessary, but we include it for convenience. We will use integer addition and the integer relational operators in examples.

The function that maps each x of type T to E is written $\lambda x : T \cdot E$. Function application is denoted by juxtaposing function and argument in the usual way, as in $(\lambda x : \mathbf{Z} \cdot x + 1)3$. Function application binds tightest of all operations.

The predicates include equivalence on terms which we write as $E \equiv F$, where E and F stand for terms of the same type. The predicate $E : T$ asserts that term E is of type T. We employ the usual logical connectives, including existential and universal quantifiers. $\equiv$ binds more tightly that the logical connectives.

Nondeterminacy in the language is introduced via the infix binary operator $\sqcap$. For terms E and F of the same type T, say, $E \sqcap F$ is also a term of type T. It is pronounced "E choice F" and denotes the nondeterministic choice among E and F. In informal operational terms, an evaluation of $E \sqcap F$ yields a single outcome drawn from an evaluation of E or F, and repeated evaluations are not guaranteed to yield identical results. For example, $1 \sqcap 2$ has possible outcomes 1 and 2, and evaluation of it may well yield 1 on one occasion, and 2 on another. Actually, after we have introduced the special term *null* below, we will see that it is possible for nondeterministic terms to have *no* outcome. $\sqcap$ is symmetric, associative, and idempotent. Elementary operators, including equality and integer addition, distribute over $\sqcap$. For example, $3 + (1 \sqcap 2) \equiv 4 \sqcap 5$ holds. The standard operators bind more tightly than $\sqcap$.

We classify all terms as either *proper* or *improper*. Intuitively, we deem a term to be *proper* iff we think of it as denoting a single entity as opposed to a choice among entities. We also use the term *elementary* in place of *proper*, and *element* in place of *proper term*. For example, all of *true*, 1, $2+3$, and $1 \sqcap (3-2)$ are elementary, while $1 \sqcap 2$ is not. In the case of basic types such as integers and booleans, their propers are just those terms that are equivalent to one of the familiar constants. For example, the proper booleans are just those terms that are equivalent to one of *true* or *false*. In the case of function types there are competing definitions of properness. There is universal agreement that a term such as, say, $(\lambda x:\mathbf{Z} \cdot x) \sqcap (\lambda x:\mathbf{Z} \cdot x+1)$ is improper because it clearly denotes a choice among different functions. However, theories differ in whether they regard a term such as $\lambda x:\mathbf{Z} \cdot x \sqcap x+1$ (in which the choice is internal) as proper. We will show that our inconsistency can arise in either approach. The motivation behind the introduction of *proper* is that in the presence of nondeterminacy many traditional laws hold good only when certain participating terms are proper. See β-substitution below for an example. We introduce the predicate ΔE as a formal encoding of "E is proper".

Unbounded nondeterminacy is introduced in the form of terms called *prescriptions*. A prescription of type T, say, has the form $\Box x{:}T \mid P$ in which x is a dummy variable that may appear free in P. $\Box x{:}T \mid P$ denotes the choice over those elements u for which $P[x\backslash u] \equiv true$ holds ($E[x\backslash F]$ denotes the term E with F substituted for the place-holder x, with the usual caveat about avoiding variable capture). Note that the bound variable in prescriptions ranges over *elements*; we will illustrate this later when we have introduced a strong equality operator. As examples of prescriptions we have $\Box z{:}\mathbf{Z} \mid 0 \leqslant z \leqslant 3$ which is equivalent to $0 \sqcap 1 \sqcap 2 \sqcap 3$, and $\Box z{:}\mathbf{Z} \mid z=1$ which is equivalent to 1. The prescription $\Box x{:}T \mid false$ denotes the empty choice and is abbreviated to $null_T$; we may omit the type subscript when it can be inferred from context or it is not significant. Note that prescriptions may or may not be elementary; for example, $\Box x{:}\mathbf{Z} \mid x=1$ is elementary while $\Box x{:}\mathbf{Z} \mid 0 \leqslant x$ and *null* are not.

Application of a λ-term to an element y is defined by the usual β-substitution rule, i.e. $(\lambda x:T \cdot E)y \equiv E[x\backslash y]$. For improper arguments, we postulate that application distributes over choice (whether bounded or unbounded). A consequence of these rules is that the parameter in the body of a λ-term is always proper.

Terms include so-called *guarded expressions* which have the form $P \to E$ and are of the same type as E. $true \to E$ is equivalent to E, and $false \to E$ is equivalent to *null*. The *conditional expression* **if** P **then** E **else** F is short-hand for $(\lambda b{:}\mathbf{B} \cdot (b \to E) \sqcap (\neg b \to F))P$.

We introduce a 2-place infix predicate $\sqsubseteq$ on terms of the same type. $E \sqsubseteq F$ is pronounced "E is refined by F". The relationship captured by $\sqsubseteq$ is called *refinement*. Conceptually, E is refined by F iff E is equivalent to F except for possibly reduced nondeterminacy in F. For base types, $E \sqsubseteq F$ holds iff the possible outcomes of F are a subset of the possible outcomes of E. For example $2 \sqcap 3 \sqcap 4 \sqsubseteq 2 \sqcap 4$ holds. (The reader is justified in thinking that $\sqsupseteq$ would be a more appropriate symbol for refinement, but $\sqsubseteq$ is what is used historically.) In the case of λ-abstractions F and G of the same type,

$F \sqsubseteq G$ iff $Fx \sqsubseteq Gx$ for every element x in the domain type of F (and G). For example, $\lambda x{:}\mathbf{Z} \cdot 1 \sqcap 2 \sqsubseteq \lambda x{:}\mathbf{Z} \cdot \mathbf{if}\ x>0\ \mathbf{then}\ 1\ \mathbf{else}\ 2$ holds. (Some readers may be prompted to compare our definition of functional refinement with a similar-looking definition in theories of partial functions. There is no harm in that, but we alert the reader not to confuse the undefined outcome (usually represented by $\perp$) with *null*. In theories which combine partiality with nondeterminacy, $\perp$ is a minimum with respect to $\sqsubseteq$.) $\sqsubseteq$ is reflexive, transitive and antisymmetric with $null_T$ a maximum in each type T.

It is standard in theories of refinement that $E \sqsubseteq F$ is provably equivalent to $E \sqcap F \equiv E$ for all terms E and F of the same type, i.e. $E \sqsubseteq F \Leftrightarrow E \sqcap F \equiv E$. It easily follows that $E \sqcap F \sqsubseteq E$ holds. We write $E \sqsubset F$ to abbreviate $E \sqsubseteq F \wedge E \not\equiv F$. $\sqsubseteq$ has the same binding power as $\equiv$.

A type is said to be *flat* iff for all *elements* of the type, refinement is identical with equivalence, i.e. if $x \sqsubseteq y \Leftrightarrow x \equiv y$ holds for all elements x and y. The integers and the booleans are flat. We have italicised *elements* in the definition of flatness to draw the reader's attention to its importance. One should not, for example, jump to the conclusion that the booleans are not flat based on the (true) facts $true \sqcap false \sqsubseteq true$ and $true \sqcap false \not\equiv true$—that has no bearing on flatness because $true \sqcap false$ is not an element as required by the definition. From our informal explanations, it may seem counter-intuitive that we can have a type with *elements* x and y satisfying $x \sqsubset y$, but that is indeed possible in some theories as we shall see shortly, and the inconsistency we describe depends on it.

A term G is said to be *monotonic in* x iff for all terms E and F of appropriate type

$$E \sqsubseteq F \Rightarrow G[x\backslash E] \sqsubseteq G[x\backslash F].$$

A weaker notion is what we shall call *e-monotonicity* (the *e* standing for *elementary*). A term G is *e-monotonic in* x iff for all *elements* u and v of appropriate type

$$u \sqsubseteq v \Rightarrow G[x\backslash u] \sqsubseteq G[x\backslash v].$$

All terms in which the place-holder is of a flat type are trivially *e*-monotonic, but not necessarily so in the case of nonflat types as we shall see.

The meaning of the language constructs beyond what is given above is not of interest for our purposes. For example, we do not care what meaning is given to $P \rightarrow E$ when P is *null* or $true \sqcap false$. Neither are the model-theoretic semantics of the language of any relevance to our argument.

3. The inconsistency

3.1. Inconsistent theories

The root cause of the inconsistency is substitution of a variable in a non-*e*-monotonic position. We shall illustrate it in the setting of function application, but later we show that it is more general. In this subsection, we show that any theory of nondeterministic functions for which (1) to (4) below hold is necessarily inconsistent. Later we show that two published theories satisfy the criteria and so are inconsistent.

$$E \sqsubseteq F \quad \Leftrightarrow \quad (E \sqcap F \equiv E), \tag{1}$$

$$(\lambda x\!:\!T \cdot E)(F \sqcap F') \equiv (\lambda x\!:\!T \cdot E)F \sqcap (\lambda x\!:\!T \cdot E)F', \tag{2}$$

$$(\lambda x\!:\!T \cdot E)y \equiv E[x\backslash y] \quad \text{for any element } y \text{ of } T, \tag{3}$$

$$\text{Some term } H \text{ is not } e\text{-monotonic.} \tag{4}$$

As is well known, it follows easily from (1) and (2) that function application is monotonic in the argument, i.e.

$$F \sqsubseteq F' \Rightarrow (\lambda x\!:\!T \cdot E)F \sqsubseteq (\lambda x\!:\!T \cdot E)F'. \tag{5}$$

To construct the inconsistency, let H denote a term which is not e-monotonic in placeholder x. It follows that the type of x must be nonflat—call it *NonFlat*. We can therefore choose two distinct *elements* u and v of *NonFlat* such that $u \sqsubseteq v$, but not $H[x\backslash u] \sqsubseteq H[x\backslash v]$. Now define $f \mathrel{\hat{=}} \lambda x\!:\!NonFlat \cdot H$. From $u \sqsubseteq v$ we conclude via (5) that $fu \sqsubseteq fv$, which by (3) reduces to $H[x\backslash u] \sqsubseteq H[x\backslash v]$, contra our choice of u and v.

On first appearances, the trap seems a simple one which we would be unlikely to fall into. The subtlety, however, lies in the existence of two *elements* such that one refines the other. Intuition might not lead us to expect this, and hence we might not be on our guard to examine carefully all its consequences, in particular to ensure that all terms are e-monotonic. We shall show that the theories of Norvell and Hehner [9] and Ward [15] fall into the trap, each in its own way. The theory of Norvell and Hehner [9] explicitly includes so-called *strong equality* which is well known to have poor monotonicity properties. In this case the theory would seem to be safe from inconsistency because the type constructors of Norvell and Hehner [9] appear to rule out nonflat types. However, we shall see that they can slip in through a back door. By contrast the theory of Ward [15] explicitly embraces nonflat types, but inadvertently allows a non-e-monotonic equality to sneak in. In both cases, the damage is easily repaired, and we shall show how to do so.

3.2. The theory of Norvell and Hehner

We show that the theory of Norvell and Hehner [9] has the ingredients (1) to (4) for inconsistency. Both (2) and (3) are postulated in [9]. Norvell and Hehner [9] also postulates (1) in the form $E \sqsubseteq F \Leftrightarrow (X \sqcap F \equiv E)$ *for some* X, and this is sufficient to infer (5) (the only role of (1) is in inferring (5)).

To construct the term H of (4), we will make use of the boolean-valued strong equality operator $==$ provided by Norvell and Hehner [9] (the symbol $\equiv$ is used in [9]). It is defined by $E == F \equiv \mathit{true}$ if $E \equiv F$, and otherwise $E == F \equiv \mathit{false}$. Let u and v be elements of some type such that $u \sqsubset v$ holds. Observe that both $u == v \equiv \mathit{false}$ and $v == v \equiv \mathit{true}$ hold, but not $\mathit{false} \sqsubseteq \mathit{true}$. Hence $x == v$ is not e-monotonic in x. It only remains to exhibit a nonflat type with $==$. Since $==$ is defined on all types in [9], we need only exhibit any nonflat type, and we shall choose $\mathbf{Z} \to \mathbf{Z}$. (Aside: We fulfil our promise to give an example illustrating that the bound variable in $\Box x{:}T \mid P$ ranges over elements only: $(\Box x{:}\mathbf{Z} \mid x == 1 \sqcap 2)$ is equivalent to $null_{\mathbf{Z}}$, not $1 \sqcap 2$—although $1 \sqcap 2 == 1 \sqcap 2$ is equivalent to *true*, $1 \sqcap 2$ is not an element.)

We shall need some further notation. We write $(\forall x : T \cdot \mathscr{P})$ as an abbreviation for $(\forall x \cdot x : T \Rightarrow \mathscr{P})$, and $(\exists x : T \cdot \mathscr{P})$ as an abbreviation for $(\exists x \cdot x : T \wedge \mathscr{P})$. By convention, bound variables in quantifications range over *elements* only, and so there is no distinction between, say $(\exists x : T \cdot \mathscr{P})$ and $(\exists x : T \cdot \Delta x \wedge \mathscr{P})$. Our inconsistency argument does not rely on this convention, which we adopt for convenience only. It is convenient because it matches the behaviour of dummy variables in prescriptions and λ-terms.

In establishing that $\mathbf{Z} \to \mathbf{Z}$ is not flat, we shall appeal to the following:

$$(\lambda x : T \cdot E) \sqsubseteq (\lambda x : T \cdot F) \;\Leftrightarrow\; (\forall x : T \cdot E \sqsubseteq F), \tag{6}$$

$$\sqsubseteq \text{ is reflexive, antisymmetric, and transitive,} \tag{7}$$

$$E \sqsubseteq null, \tag{8}$$

$$(E \not\equiv null) \Rightarrow (\exists x : T \cdot E \sqsubseteq x), \quad \text{where } E : T \tag{9}$$

(6)–(8) are postulated in [9], and (9) can be inferred (we do so later). Although we shall be appealing to (6) to (9), we point out that they are not of themselves a source of inconsistency, even in theories with nonflat types, and we have no quarrel to make with them.

We easily infer (10) below from (6) and the antisymmetry component of (7):

$$((\lambda x : T \cdot E) \equiv (\lambda x : T \cdot F)) \;\Leftrightarrow\; (\forall x : T \cdot E \equiv F). \tag{10}$$

The theory of Norvell and Hehner [9] defines elementhood of functions as follows:

$$\Delta(\lambda x : T \cdot E) \;\Leftrightarrow\; (\forall x : T \cdot \Delta E). \tag{11}$$

For example, $\lambda x : \mathbf{Z} \cdot x + 1$ is an element of $\mathbf{Z} \to \mathbf{Z}$, but not $\lambda x : \mathbf{Z} \cdot x \sqcap (x+1)$. (Aside: To illustrate a point made earlier that there is no one obvious definition of elementhood for nonbasic types, compare (11) with the corresponding definition (17) from Ward [15]. In this paper, we have no argument with either definition, or with their relative merits. A sound theory can be built around either.)

We are now ready to show that $\mathbf{Z} \to \mathbf{Z}$ is not flat, i.e. that there are *elementary* functions g and h in $\mathbf{Z} \to \mathbf{Z}$ such that $g \sqsubset h$. We shall exhibit g as a λ-term, but rely on an existence proof for h. The existence of h will be established via an intermediary function f satisfying $g \sqsubset f \sqsubseteq h$. Define f and g as follows:

$$f \mathrel{\hat{=}} (\lambda x : \mathbf{Z} \cdot x = 0 \to 0),$$

$$g \mathrel{\hat{=}} (\lambda x : \mathbf{Z} \cdot 0).$$

First, it is stated in [9] that g is elementary and refined by f. If we prefer not to rely on this assertion, we can deduce it from the axioms as follows. The fact that g is elementary follows from Eq. (11). The fact that g is refined by f follows from (6), the reflexivity component of (7), the definition of guarding, and (8).

Second, by instantiating x with 1 in the bodies of f and g, it is clear that f and g differ. In deducing this we appeal to (10), the definition of guarding, and the fact that $0 \not\equiv \mathit{null}$.

Third, f is evidently non-*null*, and is explicitly stated to be so in [9]. If we prefer to deduce this from the axioms, we are hampered by the fact that [9] does not state the result of applying function $\mathit{null}_{T\to U}$ to an argument. However, no outcome other than null_U seems reasonable. In contrast, $f\,0$ is 0 by (3) and the definition of guarding. As 0 differs from $\mathit{null}_{\mathbf{Z}}$, it follows that f is non-*null*.

Finally, by (9) and the fact that f is non-*null*, there exists an elementary function h such that $f \sqsubseteq h$ (*elementary* because the bound variable in (9) ranges over elements). From $g \sqsubset f \sqsubseteq h$, and the transitivity of $\sqsubseteq$ it follows that $g \sqsubset h$. We conclude that $\mathbf{Z} \to \mathbf{Z}$ is not flat.

It remains to prove (9) in the theory of Norvell and Hehner [9]. First Norvell and Hehner [9] includes a boolean-valued refinement operator $\preccurlyeq$, defined by

$$E \preccurlyeq F \equiv \mathit{true} \;\;\Leftrightarrow\;\; E \sqsubseteq F. \tag{12}$$

(In [9], $F{:}E$ is written where we write $E \preccurlyeq F$.) We shall have need of the following:

$$E \equiv (\Box x{:}T \mid E \preccurlyeq x). \tag{13}$$

In proving this we shall appeal to (14) and (15):

$$\sqcap \text{ is idempotent,} \tag{14}$$

$$E \sqcap F \equiv (\Box x{:}T \mid E \preccurlyeq x \vee F \preccurlyeq x), \quad \text{where } E : T,\;\; F : T \tag{15}$$

(15) is an axiom of Norvell and Hehner [9] (except that the type annotation is omitted), and (14) is stated in the text. The proof of (13) follows:

$$\begin{array}{ll} & E \\ \equiv & \qquad (14) \\ & E \sqcap E \\ \equiv & \qquad (15) \\ & (\Box x{:}T \mid E \preccurlyeq x \vee E \preccurlyeq x) \\ \equiv & \qquad \text{disjunction is idempotent} \\ & (\Box x{:}T \mid E \preccurlyeq x). \qquad \Box \end{array}$$

The proof of (9) follows:

$$
\begin{aligned}
& E \not\equiv \textit{null} \\
\Leftrightarrow \quad & (13) \\
& (\Box x : T \mid E \preccurlyeq x) \not\equiv \textit{null} \\
\Leftrightarrow \quad & \text{definition of } \textit{null} \\
& (\Box x : T \mid E \preccurlyeq x) \not\equiv (\Box x : T \mid \textit{false}) \\
\Rightarrow \quad & \text{substitution of equals, all dummies range over elements} \\
& \neg(\forall x : T \cdot (E \preccurlyeq x) \equiv \textit{false}) \\
\Leftrightarrow \quad & (12) \\
& \neg(\forall x : T \cdot \neg(E \sqsubseteq x)) \\
\Leftrightarrow \quad & \text{relationship between universal and existential quantification} \\
& (\exists x : T \cdot E \sqsubseteq x). \qquad \Box
\end{aligned}
$$

There are gaps in the formal axiomatisation of the theory of Norvell and Hehner [9], and so in part the above argument necessarily relies on the informal text. In particular, the idempotence of choice and the fact that the dummy variable in $\Box x{:}T \mid P$ ranges over elements is taken from the informal text of Norvell and Hehner [9]. Whether or not our interpretation agrees with that intended by the authors is not pertinent as long as our interpretation is seen to be reasonable. The point remains that a seemingly reasonable axiomatisation of nondeterministic functions nevertheless harbours an inconsistency.

3.3. The theory of Ward

We show that the theory of Ward [15] has the ingredients (1) to (4) for inconsistency. The theory of Ward [15] is presented as a collection of so-called *laws*, among them (2) and (3). No logic is supplied. Instead we are invited to prove new laws by appealing to what are called *semantic meaning functions*. In this way we can prove (1). We shall not present a proof of (1), in part because it would require us to describe at some length the model theory of Ward [15], but mainly because (1) is not controversial (it is no more than a reformulation of the simple set-theoretic relationship $S \supseteq T \Leftrightarrow S \cup T = S$ for sets S and T).

In establishing (4), we shall appeal to (6), (7), and (16) below:

$$E \sqcap F \sqsubseteq E. \tag{16}$$

Although (6) is omitted from the assembled list of laws in [15], we take this to be inadvertent: it is the standard definition of refinement of functions and is employed regularly in the text of Ward [15]. Eqs. (7) and (16) are given laws. We deduce (10) from (6) and (7) as before.

Function types are quite explicitly nonflat in [15], which postulates that all λ-terms are proper:

$$\Delta(\lambda x{:}T \cdot E). \tag{17}$$

In particular, in $\mathbf{Z} \to \mathbf{Z}$ both $\lambda x{:}\mathbf{Z} \cdot 1$ and $\lambda x{:}\mathbf{Z} \cdot 1 \sqcap 2$ are proper. Moreover, it follows from (6) and (16) that $(\lambda x{:}\mathbf{Z} \cdot 1 \sqcap 2) \sqsubseteq \lambda x{:}\mathbf{Z} \cdot 1$. By (10) and the fact that $1 \sqcap 2 \not\equiv 1$, the two functions differ from one another. Hence $\mathbf{Z} \to \mathbf{Z}$ is not flat and it only remains to exhibit an operator on $\mathbf{Z} \to \mathbf{Z}$ that is not e-monotonic. We will show that the equality operator fits the bill. Equality obeys the following law given in [15]:

$$(\Box x{:}T \mid x =_T E) \equiv E \quad \text{if } \Delta E \text{ and } x \text{ not in } E. \tag{18}$$

Although (18) is beyond reproach in theories with only flat types, it is not appropriate when the type T is nonflat. We shall show shortly that if $x =_T y$ is e-monotonic in y for all x, then T is necessarily flat. As $\mathbf{Z} \to \mathbf{Z}$ is not flat, it follows that there is a function f of type $\mathbf{Z} \to \mathbf{Z}$ such that $f =_{\mathbf{Z}\to\mathbf{Z}} y$ is not e-monotonic in y, and we have established (4).

It remains to show that flatness of T follows from (18) and e-monotonic equality. Let c and d be arbitrary *elements* of T such that $c \sqsubseteq d$; we prove $c \equiv d$. By the antisymmetry component of (7), this reduces to showing $d \sqsubseteq c$. We shall appeal to the following law:

$$(\Box x{:}T \mid P) \sqsubseteq (\Box x : T \mid Q) \quad \text{if } (\forall x{:}T \cdot Q \sqsubseteq P). \tag{19}$$

(According to the conventions of Ward [15], the universal quantification in (19) is implicit and can be omitted.) Eq. (19) can be proved using the semantic meaning functions of Ward [15]. However, we shall rely on the fact that it is stated in the text of Ward [15] (p. 33). By (18), $d \sqsubseteq c$ is equivalent to $(\Box x{:}T \mid x =_T d) \sqsubseteq (\Box x{:}T \mid x =_T c)$. By (19) this follows from $(\forall x{:}T \cdot x =_T c \sqsubseteq x =_T d)$. Under the assumption that $x =_T y$ is e-monotonic in y for all elements x of type T, $(\forall x{:}T \cdot x =_T c \sqsubseteq x =_T d)$ follows from $c \sqsubseteq d$ which is assumed.

The above argument assumes that equality is defined on function types in [15]. Although equality on function types would not be included in a programming language, it is perfectly at home in a specification language, and [15] does not rule it out. We note that prescriptions of function types are supported, which would be pointless without equality on functions because (18) is the only given elimination law for $\Box$.

3.4. *An example without function application*

The inconsistency can occur whenever a variable in a non-e-monotonic position can be substituted. Function application is not a crucial ingredient, as we now show. Suppose a theory includes the following axiom taken from Norvell and Hehner [9]:

$$(\Box x : T \mid P) \sqsubseteq y \;\Leftrightarrow\; (P[x\backslash y] \equiv \mathit{true}) \quad \text{for any element } y \text{ of type } T. \tag{20}$$

Let u and v be two distinct elements of a nonflat type *NonFlat*, say, such that $u \sqsubseteq v$. Let Q be a boolean-valued term such that $Q[x\backslash u] \equiv \mathit{true}$ and $Q[x\backslash v] \equiv \mathit{false}$ (it follows that the place-holder x is of type *NonFlat*). Let E abbreviate $(\Box x{:}\mathit{NonFlat} \mid Q)$. Trivially, $E \sqsubseteq u$ by (20), and hence $E \sqsubseteq v$ by transitivity of $\sqsubseteq$. However, by direct application of (20), $E \sqsubseteq v$ does *not* hold! There is no appeal to function application, but substitution is retained via (20). The theory of Norvell and Hehner [9] has the necessary ingredients for inconsistency by this route. Let elementary functions g and h of type $\mathbf{Z} \to \mathbf{Z}$ be as introduced in Section 3.2 (you need only recall that $g \sqsubset h$ holds). For u and v take g and h, respectively, and instantiate E as $(\Box x{:}\mathbf{Z} \to \mathbf{Z} \mid x == g)$.

4. Avoiding the inconsistency

4.1. Avoiding nonflat types or non-e-monotonic operators

The inconsistency relies on the language having certain constructs defined in certain ways. By avoiding any of these, we avoid the inconsistency. For example, if the application of a function to an improper term is defined other than by distribution then the problem does not arise (although other concerns may have to be addressed). Alternatively, we might construct the language so that all types are flat. Our demonstration that function types are nonflat in the theory of Norvell and Hehner [9] relied on the fact that the body of a λ-term could reduce to *null* for some arguments. If we prohibit the use of *null* or guarded expressions as the body of a λ-term, then all nonflat types in [9] disappear and consistency is restored.

Restricting a language to flat types can impair its convenience. To achieve flatness, we typically define elementhood of functions along the lines of (11). This has the consequence that, in contrast with (17), we cannot tell syntactically whether a λ-term G is proper. Hence, we cannot tell syntactically whether we can use β-substitution when G is the argument of a higher-order function F. Moreover, if G is not proper, then in applying F to G, we must reduce G to a choice over its proper refinements, and for function types these are usually infinite in number.

In practice (17) is more convenient than (11). The price we pay for (17) is nonflat function types and a consequent potential for inconsistency. We can avoid inconsistency in the presence of (17) by ensuring that all operators on function types are e-monotonic in all arguments. In particular, we could restore consistency to the theory of Ward [15] either by ruling out equality on nonflat types, or by redefining it so that it is e-monotonic. To define an e-monotonic equality, we introduce the notion of atomic terms. A term E is *atomic* iff it is elementary and satisfies $E \sqsubseteq F \Leftrightarrow E \equiv F$ for every term F of the same type as E (clearly all elements of a flat type are atomic). For example, if we define elementhood on functions by (17), both $\lambda x{:}\mathbf{Z} \cdot \mathit{null}_{\mathbf{Z}}$ and $\lambda x{:}\mathbf{Z} \cdot 0 \sqcap 1$ are elementary, but only $\lambda x{:}\mathbf{Z} \cdot \mathit{null}_{\mathbf{Z}}$ is atomic—$\lambda x{:}\mathbf{Z} \cdot 0 \sqcap 1$ is refined by, say, the element $\lambda x{:}\mathbf{Z} \cdot 0$. Equality becomes e-monotonic if we define it as follows. Define $E = F$ to behave like strong equality when E and F are atomic, to be equal to $\mathit{null}_{\mathbf{B}}$ if either argument is *null*, and otherwise to be equivalent to $true \sqcap false$. To see that this $=$ is e-monotonic, let u and v be elements such that $u \sqsubseteq v$; we show $u = E \sqsubseteq v = E$. Assume $u \sqsubset v$ as when $u \equiv v$ the result follows trivially from the reflexivity of $\sqsubseteq$. If E is equivalent to *null* then both $u = E$ and $v = E$ are *null*, and $u = E \sqsubseteq v = E$ follows from the reflexivity of $\sqsubseteq$. Otherwise $u = E$ is equivalent to $true \sqcap false$ (u is not atomic as it is refined by v), and $true \sqcap false$ is refined by $v = E$ no matter what the outcome of $v = E$. (This e-monotonic equality is formally defined in [10] in which it is shown to have reasonable properties such as symmetry and transitivity.) If we replace the requirement in (18) that E be proper with the stronger requirement that E be atomic, the inconsistency can no longer be constructed in the theory of Ward [15].

4.2. Retaining nonflat types and non-e-monotonic operators

It is possible to retain a combination of nonflat types and non-e-monotonic operators, but with great caution. One way to ensure consistency is to forbid the use of a substitutable variable in any position that is not e-monotonic in that position. For example, using the notation of Section 3.2, terms such as $x == E$ and $x \preccurlyeq E$ would be forbidden where x can be substituted, for example where x is bound by λ. This approach is adopted in [8], which in essence re-constructs the theory of Norvell and Hahner [9] using nonflat types while retaining $==$ and $\preccurlyeq$.

It may appear at first sight that the restriction of the preceding paragraph is needlessly strong in that it forbids occurrences of variables in positions that have merely the *potential* to give rise to non-e-monotonicity. Suppose, for example, that the language includes a type *Two* containing precisely two elements u and v such that $u \sqsubset v$, the boolean operators $\preccurlyeq$ and $==$ of Section 3.2, and conditional expressions. (Actually, we do not need $==$, but it allows us to give a simple example.) Since $==$ is not in general e-monotonic in either argument, function G defined by

$$G \mathrel{\hat{=}} (\lambda x : Two \cdot \textbf{if } x == u \textbf{ then } 1 \sqcap 2 \textbf{ else } 1),$$

would not be regarded as a legitimate term of the language according to the restriction proposed above. However, it turns out that the body of G is e-monotonic in x. To see this, observe that the only refinement in *Two* other than equality is $u \sqsubset v$. Substituting u for x in the body of G yields $1 \sqcap 2$, substituting v yields 1, and clearly $1 \sqcap 2 \sqsubseteq 1$.

The example of G above might tempt us to adopt the more liberal approach of considering each term on its merits, accepting or rejecting it based on its e-monotonicity properties regardless of how it is constructed. G would be legitimate by this criterion. Consider now g_1 and g_2:

$$g_1 \mathrel{\hat{=}} (\lambda x : Two \cdot \textbf{if } x == u \textbf{ then } 1 \textbf{ else } 1),$$

$$g_2 \mathrel{\hat{=}} (\lambda x : Two \cdot \textbf{if } x == u \textbf{ then } 2 \textbf{ else } 1).$$

The body of g_1 is e-monotonic (it is equivalent to $\lambda x : Two \cdot 1$), but not that of g_2 and so g_2 is ruled out as a legitimate term of the language. We will now show that this more liberal approach may lead to inconsistency in some theories, in particular those that include (11) and (13).

It follows from (11) that G is not elementary, since **if** $x == u$ **then** $1 \sqcap 2$ **else** 1 reduces to $1 \sqcap 2$ when u is substituted for x, and $1 \sqcap 2$ is not elementary. Moreover, g_1 and g_2 are the only candidates for elementary refinements of G. We have already ruled out g_2, and so g_1 is the only elementary refinement of G. By (13), G is equivalent to $(\Box g{:}\mathit{Two} \to \mathbf{Z} \mid G \preccurlyeq g)$, which is equivalent to $(\Box g{:}\mathit{Two} \to \mathbf{Z} \mid g == g_1)$, which we should expect to be equivalent to g_1. Hence $G \equiv g_1$ holds which is absurd because the two functions yield different outcomes when applied to u! Actually, we do not require that $(\Box g{:}\mathit{Two} \to \mathbf{Z} \mid g == g_1)$ be equivalent to g_1, but only that $(\Box g{:}\mathit{Two} \to \mathbf{Z} \mid g == g_1)u$ differ from $1 \sqcap 2$. We have not defined the meaning of $F\,E$ when F is nondeterministic, but however we may do so, there is no conceivable way in which it will yield $1 \sqcap 2$ as the outcome of $(\Box g{:}\mathit{Two} \to \mathbf{Z} \mid g == g_1)u$.

The core of the problem described in the preceding paragraph is that we imposed a requirement on certain terms—above, we imposed a monotonicity requirement on functions—notwithstanding the fact that not all refinements of those terms also enjoy that property. This can be a source of trouble in theories which includes (13). Eq. (13), which states that all terms are the limit of their proper refinements, is a fundamental property and is not easily dispensed with. The inconsistency can be avoided by abandoning (11), replacing it perhaps with (17).

5. Other theories

In carrying out this work we examined the theories of Refs. [1–5,7–15]. Some theories, among them Z [13] and B [1], avoid the anomaly by forbidding choice on functions or by not treating such functions as first-class. (Functions are said to be *first-class* if the language does not impose restrictions on them that do not apply to terms in general. For example, functions are not first-class in a language which forbids the result type of a function to be a function type.) Other theories such as VDM (see [2, Section 13] and [4]) acknowledge that combining functions and choice would be useful, but forego it in the face of seemingly unresolvable difficulties. Yet other calculi admit first-class functions and choice, but lack the operators or the axioms to allow calculation with them. For instance, Larsen and Hansen [5] does not provide a refinement relation. In Refs. [3,11,12], all of functions, choice, and refinement are present, but the published literature provides an insufficient set of axioms for calculating with them. In the CIP language and method [10], no quantification on function types, and no choice over functions are allowed. It has been suggested to us that the restrictions in the CIP language were motivated by the requirement to avoid certain theoretical problems that arise when nondeterminism is combined with fixed-point semantics. For a resolution of that, we refer the reader to [7]. Ref. [7] also contains a summary of other well-known sources of inconsistency.

Acknowledgements

E.C.R. Hehner took the time to explain to us various points regarding his theories of nondeterminacy. Several anonymous referees took considerable pains in uncovering weaknesses and lack of clarity, and the final paper has greatly benefited from their criticisms.

References

[1] J.-R. Abrial, The B-book: Assigning Programs to Meanings, Cambridge University Press, Cambridge, 1996, ISBN 0521496159.
[2] J.C. Bicarregui, J.S. Fitzgerald, P.A. Lindsay, R. Moore, B. Ritchie, Proof in VDM: A Practioner's Guide, FACIT, Springer, Berlin, 1994.
[3] E. Hehner, A Practical Theory of Programming. Springer, New York, London, ISBN 0387941061, 1993.
[4] C.B. Jones, Systematic Software Development using VDM, Prentice-Hall International, 1986.
[5] P.G. Larsen, B.S. Hansen, Semantics of under-determined expressions, Formal Aspects Comput. 8 (1) (1996) 47–66.
[6] L. Meertens, Algorithmics—towards programming as a mathematical activity, in: J.W. de Bakker, M. Hazewinkel, J.K. Lenstra (Eds.), Mathematics and Computer Science, Vol. 1, CWI Monographs, North-Holland, Amsterdam, 1986, pp. 1–46.
[7] J.M. Morris, A. Bunkenburg, Specificational functions, ACM Trans. Programming Languages Systems 21 (1999) 677–701.
[8] J.M. Morris, A. Bunkenburg, A theory of bunches, Acta Inform. 37 (2001) 541–561.
[9] T.S. Norvell, E.C.R. Hehner, Logical specifications for functional programs, Proceedings of the 2nd International Conference on Mathematics of Program Construction, Oxford, 29 June–3 July 1992, Vol. 669, Lecture Notes in Computer Science, Springer, Berlin, 1993, pp. 269–290.
[10] H.A. Partsch, Specification and Transformation of Programs, Springer, New York, 1990.
[11] The Raise Language Group, The RAISE Specification Language, Prentice-Hall, London, 1992.
[12] The RAISE Method Group, The RAISE Development Method, Prentice-Hall, London, 1995.
[13] J.M. Spivey, The Z notation: A reference manual, International Series in Computer Science, Prentice-Hall, Englewood Cliffs, NJ, 2nd Edition, 1992.
[14] M. Walicki, S. Meldal, Algebraic approaches to nondeterminism—an overview, ACM Comput. Surveys 29 (1) (1997) 30–81.
[15] N. Ward, A refinement calculus for nondeterministic expressions, Ph.D. Thesis, University of Queensland, 1994.