Calculating risk in functional programming

J.N. Oliveira
(joint work with D. Murta)

IFIP WG2.1 meeting #70 - Ulm, July 2013

QAIS Project - Grant PTDC/EIA-CCO/122240/2010

INESC TEC & University of Minho
Motivation

Two-sided motivation:

**Practical**  Software safety and certification standards concerned with calculating risk involved in safety-critical software.

**Theoretical**  Quantitative methods in the algebra of programming (AoP) lead to a LAoP ("L" for linear).

Question:

*Can we transform (functional) programs so as to mitigate unexpected faults better than the original ones?*
Safety and certification

Formal method bias:

Interested in the opportunities open for Formal Methods by RTCA DO 178C for certifying airborne software.

Challenged by

the use of formal methods to be "at least as good as" a conventional approach that does not use formal methods.

(Joyce, 2011)

[ ... "at least as good" ? ... ]
Qualitative vs quantitative

Quoting Jackson (2009):

A dependable system is one (..) in which you can place your reliance or trust. A rational person or organization only does this with evidence that the system’s benefits outweigh its risks.

In formula

\[
\text{dependable system} = \text{benefit} + \text{risk}
\]

we identify:

- **benefit** = qualitative
- **risk** = quantitative.
MOD Defence Standard 00-56:

9.1 The Contractor shall produce a Safety Case for the system [which] shall [provide] a compelling, comprehensible and valid case that a system is safe for a given application in a given environment.

DS 00-56 (contd.):

10.5.4 All assumptions, data, judgement and calculations underpinning the Risk Estimation shall be recorded in the Safety Case, such that the risk estimates can be reviewed and reconstructed.

Risk estimation? Calculations? How, when and where is this performed in a FM life-cycle?
P(robabilistic)R(isk)A(nalysis)

NASA/SP-2011-3421 (Stamatelatos and Dezfuli, 2011):

1.2.2 A PRA characterizes risk in terms of three basic questions: (1) What can go wrong? (2) How likely is it? and (3) What are the consequences?

The PRA process

answers these questions by systematically (...) identifying, modeling, and quantifying scenarios that can lead to undesired consequences

Moreover,

1.2.3 (...) The total probability from the set of scenarios modeled may also be non-negligible even though the probability of each scenario is small.
Doesn’t work in FMs — why?

Program semantics are usually qualitative — how does one quantify risk in standard denotational semantics?

PRA performed a posteriori — we’ve seen this before, eg. in ’a posteriori’ program correctness.

Need for a change:

Programming should incorporate risk as the rule rather than the exception (absence of risk = ideal case).

Need for combinatorsexpressing risk of failure, eg. probabilistic choice (McIver and Morgan, 2005)

bad $p \diamond$ good

between expected behaviour and misbehaviour.
Quantitative semantics

Program semantics denoted by (typed) stochastic matrices.

Semantics of language constructs modelled by linear algebra operators — for instance,

\[
\begin{bmatrix} P_1; P_2 \end{bmatrix} = \begin{bmatrix} P_2 \end{bmatrix} \cdot \begin{bmatrix} P_1 \end{bmatrix}
\]

where the dot means matrix multiplication — including recursion.

Laws of the LAoP enable the calculation of risk (eg. fault propagation).

Simulation easy to perform in a monadic language such as Haskell (distribution monad).
Quantitative functional programming

Monadic code is in general ready to accommodate PRA simulation in functional programming. Example: a lossy channel

\[
fcat \ p = \ \text{fold} \ (\text{lose} \ p \odot \text{send}) \ \text{nil}
\]

(for \(\text{send} = \text{cons}\) and \(\text{lose} = \text{snd}\)) in which we express the fact that, with probability \(p\), \(fcat\) fails to pass data from input to output.

For \(p = 0.1\), for instance, distribution \(fcat \ p \ "abc"\) will range from perfect copy (72.9%) to complete loss (0.1%) — cf. “quantified suffix”:

<table>
<thead>
<tr>
<th>String</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;abc&quot;</td>
<td>72.9%</td>
</tr>
<tr>
<td>&quot;ab&quot;</td>
<td>8.1%</td>
</tr>
<tr>
<td>&quot;ac&quot;</td>
<td>8.1%</td>
</tr>
<tr>
<td>&quot;bc&quot;</td>
<td>8.1%</td>
</tr>
<tr>
<td>&quot;a&quot;</td>
<td>0.9%</td>
</tr>
<tr>
<td>&quot;b&quot;</td>
<td>0.9%</td>
</tr>
<tr>
<td>&quot;c&quot;</td>
<td>0.9%</td>
</tr>
<tr>
<td>&quot;&quot;</td>
<td>0.1%</td>
</tr>
</tbody>
</table>
Details

Nothing special, just a monadic variant of \textit{foldr}

\[
\text{fold} :: \text{Monad } m \Rightarrow ((a, b) \to m b) \to m b \to [a] \to m b
\]
\[
\text{fold } f \ d \ [\ ] = d
\]
\[
\text{fold } f \ d \ (h : t) = \text{do } \{ x \leftarrow \text{fold } f \ d \ t; f \ (h, x) \}\n\]

which switches to distributions or lists (cf. the suffix view above) as you wish.

Later we will need \texttt{for}-loops, so we anticipate this combinator:

\[
\text{for} :: (\text{Monad } m, \text{Integral } t) \Rightarrow (b \to m b) \to b \to t \to m b
\]
\[
\text{for } b \ i \ 0 = \text{return } i
\]
\[
\text{for } b \ i \ (n + 1) = \text{do } \{ x \leftarrow \text{for } b \ i \ n; b \ x \}\n\]
Quantitative functional programming

Another example:

\[\text{fcoun}\text{t } q = \text{fold } ((id \ q \ \Diamond \ \text{succ}) \cdot \text{snd}) \ 0\]

is a risky \textit{length} function: with probability \( q \), it doesn’t count. For instance, for \( q = 0.15 \), distribution \textit{fcoun}\text{t } q \ "abc" \ will be:

<table>
<thead>
<tr>
<th>Count</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>61.4%</td>
</tr>
<tr>
<td>2</td>
<td>32.5%</td>
</tr>
<tr>
<td>1</td>
<td>5.7%</td>
</tr>
<tr>
<td>0</td>
<td>0.3%</td>
</tr>
</tbody>
</table>

However, we are \textbf{simulating} — not predicting!

**Question:** what can we predict about \((\text{fcoun}\text{t } q) \cdot (\text{fcat } p)\)? Can we fuse this?
Quantitative functional programming

Another example:

\[
\text{fcount } q = \text{fold } ((\text{id } q \diamond \text{succ}) \cdot \text{snd}) \ 0
\]

is a risky \textit{length} function: with probability \(q\), it doesn’t count. For instance, for \(q = 0.15\), distribution \text{fcount } q \ "abc" \ will be:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>61.4%</td>
</tr>
<tr>
<td>2</td>
<td>32.5%</td>
</tr>
<tr>
<td>1</td>
<td>5.7%</td>
</tr>
<tr>
<td>0</td>
<td>0.3%</td>
</tr>
</tbody>
</table>

However, we are \textit{simulating} — not predicting!

**Question**: what can we predict about \((\text{fcount } q) \cdot (\text{fcat } p)\)? Can we fuse this?
Fault fusion

Fold-fusion law in the LAoP

\[ k \cdot (\text{fold } g e) = \text{fold } f d \quad \Leftarrow \quad k \cdot [e|g] = [d|f] \cdot (F k) \quad (1) \]

holds in the probabilistic setting, but now

- regard function variables (eg. \( k, g, e \) etc) as (column) stochastic typed matrices;
- such matrices represent the Kleisli category of the distribution monad;
- \( F k = id \oplus (id \otimes k) \) is the base functor, where \( \cdot \otimes \cdot \) denotes Kronecker product and \( \cdot \oplus \cdot \) denotes direct sum of matrices;
- \([f|g]\) glues \( f \) and \( g \) horizontally (coproduct combinator).
Fault fusion

We want to solve equation \((fcount q) \cdot (fcat p) = fold \ x \ y\) for unknowns \(x\) and \(y\):

\[
(fcount q) \cdot (fcat p) = fold \ x \ y
\]

\[
\iff \quad \{ \text{fold fusion (1)} ; \text{definition of } fcat \ \}
\]

\[
(fcount q) \cdot [\text{nil} | (\text{lose } p \diamond \text{send})] = [x | y] \cdot (F (fcount q))
\]

\[
\iff \quad \{ \text{coproduct fusion (2)} ; \text{definition of } F ; (3) ; (4) \ \}
\]

\[
\begin{cases}
(fcount q) \cdot \text{nil} = x \\
(fcount q) \cdot (\text{lose } p \diamond \text{send}) = y \cdot (id \otimes (fcount q))
\end{cases}
\]

where (LAoP):

\[
P \cdot [M | N] = [P \cdot M | P \cdot N] \quad (2)
\]

\[
[M | N] \cdot (P \oplus Q) = [M \cdot P | N \cdot Q] \quad (3)
\]

\[
[M | N] = [P | Q] \iff M = P \land N = Q \quad (4)
\]
Fault fusion (cntd.)

From \((f\text{count } q) \cdot \text{nil} = 0\) we obtain \(x = 0\).

We are left with the second equality, which we solve for \(y\) knowing that choice-fusion laws

\[
\begin{align*}
h \cdot (f \cdot g) &= (h \cdot f) \cdot (h \cdot f) \\
(f \cdot g) \cdot h &= (f \cdot h) \cdot (g \cdot h)
\end{align*}
\]

hold:

\[
\begin{align*}
((f\text{count } q) \cdot (\text{snd} \cdot \text{cons}) &= y \cdot (id \otimes (f\text{count } q)) \\
\Leftrightarrow \{ \ \text{choice fusion (5)} \} \\
((f\text{count } q) \cdot \text{snd}) \cdot ((f\text{count } q) \cdot \text{cons}) &= y \cdot (id \otimes (f\text{count } q)) \\
\Leftrightarrow \{ \ \text{unfolding } (f\text{count } q) \cdot \text{cons} \} \\
((f\text{count } q) \cdot \text{snd}) \cdot ((id \cdot \text{succ}) \cdot \text{snd} \cdot (id \otimes (f\text{count } q))) &= y \cdot (id \otimes (f\text{count } q))
\end{align*}
\]
Fault fusion (cntd.)

The free theorem of \( \text{snd} \)

\[
\text{snd} \cdot (f \otimes g) = g \cdot \text{snd}
\]  \hspace{1cm} (7)

helps in the next step:

\[
((\text{fcount } q) \cdot \text{snd}) \circ ((\text{id } q \circ \text{succ}) \cdot (\text{fcount } q) \cdot \text{snd}) = y \cdot (\text{id } \otimes (\text{fcount } q))
\]

\[\iff \{ \text{choice fusion (6)} \}\]

\[
(\text{id } p \circ (\text{id } q \circ \text{succ})) \cdot (\text{fcount } q) \cdot \text{snd} = y \cdot (\text{id } \otimes (\text{fcount } q))
\]

\[\iff \{ \text{free theorem (7) again} \}\]

\[
(\text{id } p \circ (\text{id } q \circ \text{succ})) \cdot \text{snd} \cdot (\text{id } \otimes (\text{fcount } q)) = y \cdot (\text{id } \otimes (\text{fcount } q))
\]

\[\iff \{ \text{Leibniz — cancel } \text{id } \otimes (\text{fcount } q) \text{ from both sides} \}\]

\[
y = (\text{id } p \circ (\text{id } q \circ \text{succ})) \cdot \text{snd}
\]
Putting everything together, we have **consolidated** the risk of pipeline \((\text{fcount } q) \cdot (\text{fcat } p)\) into

\[
\text{fold } y \ 0 \ \text{where} \\
y = ((p + q - pq) \ id + (1 - p) (1 - q) \ succ) \cdot \ snd
\]

using definition

\[
f \ p \diamond g = p \otimes f + (1 - p) \otimes g
\]

Higher \(p, q\) reduce the probability of \(succ\) taking place.

**Fault fusion:** the risk of the **whole** expressed in terms of the risk of the **parts**.
Fault fusion (conc.)

From the calculation we can infer eg.

\[(fcount \ 0) \cdot (fcat \ p) = (fcount \ p) \cdot (fcat \ 0)\]

since terms

\[(0 + p - 0 \ p \ id + (1 - 0) (1 - p) \ succ)\]
\[(p + 0 - p0 \ id + (1 - p) (1 - 0) \ succ)\]

are the same. In words:

(for the same probabilities), a perfect counter reading from a faulty channel is indistinguishable from a faulty counter reading from a perfect channel.

Clearly, black-box testing and simulation wouldn’t be able to spot where the fault is.
LAoP vs AoP

Summing up,

- in the same way relations are needed in standard AoP for calculating functions,
- so are (typed) matrices in the LAoP for calculating probabilistic functions.
- AoP extends smoothly to the LAoP, but not the whole of it.
- A significant difference can be found in pairing (tupling in general) and mutual recursion.

Thus we focus on probabilistic pairing in the sequel.
Running examples

Consider two little programs in C, one which supposedly computes the square of a non-negative integer $n$,

```c
int sq(int n) {
    int s=0; int o=1; int i;
    for (i=1;i<n+1;i++) {s+=o; o+=2;}
    return s;
}
```

and the other

```c
int fib(int n) {
    int x=0; int y=1; int i;
    for (i=1;i<=n;i++) {int a=y; y=y+x; x=a;}
    return x;
}
```

which supposedly computes the $n$-th entry in the Fibonacci series, for $n$ positive.
Running examples

Both programs are for-loops whose bodies rely on the same operation: addition of natural numbers.

Suppose one knows that, in the machine where such programs will run, there is the risk of addition misbehaving in some known way: with probability $p$, $x + y$ may evaluate to $y$, in which case $(x+) = id$.

Or one might know that, in some unfriendly environment, the processor’s ALU may reset addition output to 0, with probability $q$.

Question: how do the above programs “react” to such faults?
Simulation

As before, we may encode the two programs in Haskell using the \texttt{for} combinator,

\begin{verbatim}
sq n =
do \{(s, o) \leftarrow \text{for loop} (0, 1) n; return s\} \textbf{where}
loop (s, o) = do \{z \leftarrow fadd 0.1 s o; return (z, o + 2)\}

fib n =
do \{(x, y) \leftarrow \text{for loop} (0, 1) n; return x\} \textbf{where}
loop (x, y) = do \{z \leftarrow fadd 0.1 x y; return (y, z)\}
\end{verbatim}

both calling

\begin{verbatim}
fadd p a = \_ \_ p \diamond (a+)\end{verbatim}

— a risky addition which resets with probability $p$. 
Simulation

Then we may simulate, for instance ($p = 0.1$)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11</td>
<td>10%</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>9%</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>8%</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>7%</td>
</tr>
<tr>
<td></td>
<td>35</td>
<td>7%</td>
</tr>
<tr>
<td></td>
<td>36</td>
<td>59%</td>
</tr>
</tbody>
</table>

$sq\ 6 = 

and, for instance:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>15%</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>15%</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>66%</td>
</tr>
</tbody>
</table>
Simulation

However, this does not tell anything special about what's happening.

We know that both programs can be derived from their specification (resp. $sq \ n = n^2$ and the binary recursive definition of Fibonacci) using the mutual-recursion law, vulg. tupling (Hu et al., 1997).

One way to compare the two implementations would be to check how far they are from their specifications (under the same faults).

By experimentation, we observed that spec + imp of $sq$ seem probabilistically indistinguishable, while Fibonacci does not: the linear version is (as much as we could test) less risky — next slide:
Simulation (faulty Fibonacci)

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\text{recursive spec}$</th>
<th>$\text{for loop implementation}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\leq 4$</td>
<td>the same</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5 65.6%</td>
<td>5 72.9%</td>
</tr>
<tr>
<td></td>
<td>4 21.9%</td>
<td>3 16.2%</td>
</tr>
<tr>
<td></td>
<td>3 10.5%</td>
<td>4 8.1%</td>
</tr>
<tr>
<td></td>
<td>2 1.9%</td>
<td>2 2.7%</td>
</tr>
<tr>
<td></td>
<td>1 0.1%</td>
<td>1 0.1%</td>
</tr>
<tr>
<td>6</td>
<td>8 47.8%</td>
<td>8 65.6%</td>
</tr>
<tr>
<td></td>
<td>7 26.6%</td>
<td>6 14.6%</td>
</tr>
<tr>
<td></td>
<td>6 11.8%</td>
<td>5 14.6%</td>
</tr>
<tr>
<td></td>
<td>5 9.8%</td>
<td>3 2.4%</td>
</tr>
<tr>
<td></td>
<td>4 2.7%</td>
<td>4 2.4%</td>
</tr>
<tr>
<td></td>
<td>3 1.1%</td>
<td>2 0.4%</td>
</tr>
<tr>
<td></td>
<td>2 0.2%</td>
<td>1 0.0%</td>
</tr>
<tr>
<td></td>
<td>1 0.0%</td>
<td></td>
</tr>
</tbody>
</table>
These experiments pointed towards checking the validity of **tupling** in the LAoP: while “vertical” (sequential) loop-fusion laws hold,

\[
\text{loop } g \\
\text{loop } f \\
\Rightarrow \\
\text{loop}(f \cdot g)
\]
Mutual recursion?

... “horizontal loop-fusion”

\[ \text{loop } f \quad \text{loop } g \]

\[ \Rightarrow (?) \]

\[ \text{loop}(f \triangle g) \]

does not seem to hold in general. Why?
Probabilistic pairing

Pairing the outputs of two probabilistic functions \( f \) and \( g \) is captured by their **Khatri-Rao** matrix product (keep thinking in terms of matrices)

![Diagram showing probabilistic pairing]

but (important!) this is a **weak** categorial product:

\[
k = f \triangle g \quad \Rightarrow \quad \begin{cases} 
  \text{fst} \cdot k = f \\
  \text{snd} \cdot k = g 
\end{cases}
\]  

(cf. the \( \Rightarrow \) in (9) — Khatri-Rao is injective but not surjective (unlike pairing in Sets).
Weak product (9) still grants the **cancellation** rule,

\[
\text{fst} \cdot (f \triangle g) = f \land \text{snd} \cdot (f \triangle g) = g
\]  

\[ (10) \]
Probabilistic pairing

... but **fusion** becomes side-conditioned

$$(f \triangle g) \cdot h = (f \cdot h) \triangle (g \cdot h) \iff h \text{ is "sharp" (100%)} \quad (11)$$

and **reconstruction** doesn’t hold in general

$$k = (\text{fst} \cdot k) \triangle (\text{snd} \cdot k)$$

cf. eg.

$$k : 2 \rightarrow 2 \times 3$$

$$k = \begin{bmatrix}
0 & 0.4 \\
0.2 & 0 \\
0.2 & 0.1 \\
0.6 & 0.4 \\
0 & 0 \\
0 & 0.1
\end{bmatrix}$$

$$(\text{fst} \cdot k) \triangle (\text{snd} \cdot k) = \begin{bmatrix}
0.24 & 0.4 \\
0.08 & 0 \\
0.08 & 0.1 \\
0.36 & 0.4 \\
0.12 & 0 \\
0.12 & 0.1
\end{bmatrix}$$

($k$ is not recoverable from its projections — Khatri-Rao not surjective).
Asymmetric Khatri-Rao fusion

Another side condition granting fusion is

\[(f \triangle g) \cdot h = (f \cdot h) \triangle (g \cdot h) \iff f \cdot h \text{ or } g \cdot h \text{ is } 100\% \quad (12)\]

which enables the following **probabilistic** mutual recursion law (**tupling**):

\[
\begin{cases}
  f \cdot \text{in} = h \cdot F (f \triangle g) \\
  g \cdot \text{in} = k \cdot F (f \triangle g)
\end{cases} \iff f \triangle g = (|h \triangle k|) \quad (13)
\]

provided one of

\[h \cdot F (f \triangle g) \text{ or } k \cdot F (f \triangle g)\]

is 100\% — generic statement for polynomial \(F\).
Asymmetric tupling

The calculation of (13) uses the two conditioned pairing fusion laws in different places:

\[ f \triangle g = (h \triangle k) \]

\[ \Leftrightarrow \quad \{ \text{cata (fold) universal property} \} \]

\[ (f \triangle g) \cdot \text{in} = (h \triangle k) \cdot F (f \triangle g) \]

\[ \Leftrightarrow \quad \{ \text{“sharp” fusion (11) ; asymmetric fusion (12)} \} \]

\[ (f \cdot \text{in}) \triangle (g \cdot \text{in}) = (h \cdot F (f \triangle g)) \triangle (k \cdot F (f \triangle g)) \]

\[ \Leftrightarrow \quad \{ \text{Khatri-Rao equality} \} \]

\[ \begin{cases} 
  f \cdot \text{in} = h \cdot F (f \triangle g) \\
  g \cdot \text{in} = k \cdot F (f \triangle g) 
\end{cases} \]
Back to square and Fibonacci

Standard tupling derivations,

\[
\begin{align*}
\text{sq} \ 0 &= 0 \\
\text{sq} \ (n + 1) &= \text{sq} \ n + \text{odd} \ n \\
\text{odd} \ 0 &= 1 \\
\text{odd} \ (n + 1) &= 2 + \text{odd} \ n
\end{align*}
\]

\[
\begin{align*}
\text{fib} \ 0 &= 0 \\
\text{fib} \ (n + 1) &= \text{f} \ n \\
\text{f} \ 0 &= 1 \\
\text{f} \ (n + 1) &= \text{fib} \ n + \text{f} \ n
\end{align*}
\]

show why \( \text{sq} \triangle \text{odd} \) and \( \text{fib} \triangle \text{f} \) react differently to faulty addition, cf.

— \( \text{odd} \) does not depend on \( \text{sq} \) and therefore remains 100% — as opposed to \( \text{fib} \) and \( \text{f} \), which contaminate each other.
Probabilistic banana-split

This also helps to see why **banana-split** still holds for $f$ and $g$ probabilistic:

$$(|f| \triangledown |g|) = (|(f \otimes g) \cdot (F \text{fst} \triangledown F \text{snd})|)$$  \hspace{1cm}  (14)

— the two computations go side-by-side and don’t interfere with each other.

This time the proof relies in something I’ve been using only recently: **free theorems** in linear algebra, in this case

$$(F f \otimes F g) \cdot \text{unzip}_F = \text{unzip}_F \cdot F (f \otimes g)$$

derived using hom-functors in matrix categories — inspired by Hinze (2012).
Wrapping up

First round of AoP extension towards LAoP (folds)

Probabilistic unfolds require sub-distributions while computing fixpoints (current work)

Currently using them in checking fault propagation in Barbosa (2001)’s components as coalgebras (probabilistic automata networks)

Probabilistic hyloorphisms are next.
Wrapping up

Weak tupling has opened new perspectives, namely in relation to \textbf{Rel} and to categorial quantum physics, under the umbrella of \textbf{monoidal} categories.

In fact, these also include \textit{FdHilb}, the category of finite dimensional Hilbert spaces. — thus the remarks by Coecke and Paquette, in their \textit{Categories for the Practising Physicist} (Coecke, 2011):

\begin{quote}
Rel [the category of relations] possesses more 'quantum features' than the category Set of sets and functions [...] The categories FdHilb and Rel moreover admit a categorical matrix calculus.
\end{quote}

I agree: \textit{Set} is too perfect to “belong to reality”...


J. Joyce. Proposed Formal Methods Supplement for RTCA DO
