Referee's Report on "A Transformational Approach to Specifying Recovery in Asynchronous Communicating Systems" by Zhiming Liu & Mathai Joseph, University of Warwich. Paper No. 40

I don't think this paper can be salvaged. A major reason for this pessimistic view is the abundance of implicitly adopted, ill-considered naming conventions.

In the introduction we encounter "The behaviour of a program P on a system with a specified set of fault actions F_p ...". I think that the authors intended to write "a specified set F_p of fault actions", but even then. Why F_p instead of just F? I think they should have written:

> "For a program P and a set F of fault actions, the behaviour of P on a system with F is then simulated by a transformation into a fault-affected version $\mathcal{F}(P, F)$",

a statement in which P and F are dummies, and $\mathcal{F}$ is the only global variable.

This obscurity pervades the whole document, e.g. on the next page: "To make the execution of a program P recoverable from a specified set of faults F_p, a backward recovery transformation $\mathcal{R}$ is required to transform P into

a program

$$\mathcal{R}(P) \equiv P \,[]\, C_P \,[]\, P_R$$

by adding checkpointing actions C_P and recovery actions P_R". The reader wonders whether the Ps in C_P and P_R have anything to do with the P to which $\mathcal{R}$ is applied.

The answer is not obvious at all. In section 5, "process p" is a constituent of "program P"; we then encounter the function "CP_p", where the subscript p refers, indeed, to "process p", but "CP" just seems an acronym for "Checkpoint", that has no relation to the name of "program P". Please don't think that this absence of relation is indicated by the switch from subscription to juxtaposition: in section 2, the "guard of action A" is denoted by "gA" (presumably for all A).

It turns out that the authors don't know that there is no such concept as "the dummy expression". After the introduction of "$P_1, \ldots, P_k$" - section 4 -, a sentence starts with "For any process p_i, let C_{p_i}, etc." This is not a slip of the pen: in section 6.1, a boolean expression starts with $\neg \exists\, CP_q(j') : \ldots$, and a set definition starts with $\{ CP_q(i) \mid \ldots\ldots$. Also - section 2.1 - both blunders in one:

$$A \circ B \triangleq \{(s,s') \mid \exists\, s_1 : ((s,s_1) \in A) \wedge ((s_1,s') \in B)\}$$

* *

*

An action is a relation on the set of states, a program P is a finite set Act_P of actions and a finite set $Init_P$ of predicates. But then

"In general, an action of a program is composed of a set of primitive actions. If A and B are actions, so are the union $A \cup B$ and the composition $A \circ B$ [....]. During the transition from one state to another, defined by an atomic action of P, the system may pass through a number of intermediate states [....]".

Since these are the first occurrences of "actions of a program", "primitive actions", and "atomic actions of [program] P", the reader is duly baffled. So are, in all probability, the authors. If composition is allowed as mechanism for the creation of new actions, it is "in general" hard to keep their number finite: $x := x+1$ yields $x := x+n$ for all positive integers n.

* * *

The authors don't know -it seems- much about type discipline. They are not afraid of overloading. The symbol $\wedge$ is used both for conjunction and concatenation. And, believe it or not, in section 6.1, they write a conjunction of two expressions, the left one of which ends on a string and the right

one of which begins with a string.

The symbol $[\!]$ has operands of all sorts of types, such as programs, sets of actions, single actions.

The authors state about CP_p that

$$CP_p(m)(x) = CP_p(rec_x)(m)$$

and suggest this as a definition of "the function $CP_p(m)$", but $CP_p(m)(x)$ does not depend on the value of x! (At the right-hand side, the subscript x determines the identity of the sequence variable rec_x, which is a "sequence variable which records the saved values of variable x".)

Mysterious is - section 6.1 -

"1. $\forall p \in \mathcal{P}: \exists! CP_p(m) \in \overleftarrow{\mathcal{P}}$ (where $\exists!$ means there is exactly one)"

The question is: one what? $CP_p(m)$ denotes a checkpoint, but equality of checkpoints is not defined. (Are identical twins equal?) Note, by the way, that $\overleftarrow{\mathcal{P}}$ is not a function of $\mathcal{P}$: the superarrow does <u>not</u> denote an operator.

The formulation of Theorem 3 is almost certainly wrong: it contains "n_p", but only in the condition "$CP_p(n_p) \preceq CP_p(m)$"; as $\preceq$ is reflexive it would be true for $n_p = m$, but the whole condition is a mystery.

As time goes on, it becomes increasingly hard to parse formulae. Don't ask me what to do with

$$[]_{\mathcal{P}} : \mathcal{P} \subseteq Proc :: f_{\mathcal{P}} \rightarrow (||_x : x \in Var(P) :: x := v_x).REC_P$$

for the text has left me as ignorant as you. The overall recommendation "Definite Reject" seems unavoidable.

* * *

The text is so terribly inadequate that its authors must live in an intellectual desert. Who have been their teachers? Who are their colleagues? What do they read? A refereeing job like this makes me very sad.

Austin, 1 April 1992
(how appropriate!)

prof. dr. Edsger W. Dijkstra
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712-1188
USA