Another Position Paper on "Fairness"

Fred B. Schneider[1]
Department of Computer Science
Cornell University
Ithaca, New York 14850

Leslie Lamport
Digital Equipment Corporation
Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301

In [2], Dijkstra argues that "fairness, being an unworkable notion, can be ignored with impunity". We show in this note that his argument leads to the same conclusion about termination and all other liveness properties. We also argue that ignoring liveness properties is a bad idea.

The argument of [2] can be summarized as follows. An obligation is considered *void* if it is impossible to detect that the obligation has not been fulfilled. Any property of program execution that cannot be checked by performing finite experiments is, therefore, a void obligation. A specification, being a contract between implementor and user, should not contain clauses that cannot be enforced. Therefore, void obligations should not appear in specifications, since there would be no way to determine whether or not an implementation satisfies void obligations in the specification. Obligations that should not appear in specifications can be "ignored with impunity".

The argument is applied to fairness in [2] by using the following program:

```
(1A) $b := true$;
     do $b \rightarrow print(0)$
       [] $b \rightarrow print(1)$; $b := false$
     od
```

If the **do** is fair, the program will print a string of 0's and then eventually print a 1 and halt. Dijkstra argues, however, that one can claim to have implemented (1A) by writing

```
(1B) $b := true$;
     do $b \rightarrow print(0)$ od
```

since no finite experiment can refute this claim. Thus, fairness of the **do** is a void obligation.

However, the same argument applies to termination, the property asserting that a program eventually halts. One can claim to implement the program fragment

(2A) $x := 1$

by writing

(2B) **do** $true \rightarrow$ **skip** **od**

Program (2A) terminates. Although (2B) does not terminate, no finite experiment can refute the claim that it does terminate, so termination is a void obligation. We have shown that termination is "an unworkable notion" using the same argument as was used in [2] to dismiss fairness.

[1]Schneider is supported in part by the Office of Naval Research under contract N00014-86-K-0092, the National Science Foundation under Grant No. CCR-8701103, and Digital Equipment Corporation. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author and do not reflect the views of these sponsors.

We can even take this one step further. A *liveness property* is a property that asserts something good must eventually happen. (See [1] for a formal definition of liveness.) Termination is the archetypal liveness property—the "good thing" is the program terminating. The argument above can be applied to any liveness property L to conclude that L is "an unworkable notion", because in L the good thing happening is a void obligation.

However, there are good reasons why one should be reluctant to ignore liveness properties. All properties of a program are the conjunction of liveness properties and safety properties [1]. Safety properties assert that some bad thing does not happen during execution. (Again, see [1] for a formal definition.) Partial correctness is the archetypal safety property—the "bad thing" in partial correctness is termination with the wrong answer. Safety properties do not allow us to distinguish between programs (1A) and (1B) or between (2A) and (2B). Every safety property satisfied by (1A) is also satisfied by (1B); every safety property satisfied by (2A) is also satisfied by (2B). Liveness properties do allow us to distinguish between programs (1A) and (1B) and between (2A) and (2B). Program (1A) with a fair **do** satisfies the liveness property

LP1: Eventually print a 1 and terminate.

Program (1B) does not. Program (2A) satisfies termination, a liveness property, but program (2B) does not.

From a practical point of view, liveness properties are unsatisfactory because, while they assert that something must happen, they do not say when. It is of little use to know that a program will eventually terminate without knowing if it will terminate within a century. (Terminating within a century is a safety property; it asserts that the program will not still be running when some clock reads one century from now.) However, we cannot prove that a Pascal program will terminate within a century because the definition of Pascal says nothing about execution speed. Any terminating Pascal program has a correct implementation that runs for more than a century. The strongest assertion that can hold for any valid implementation is termination, which is a liveness property.

Fairness is an especially benign type of liveness property because it usually appears in a property as the antecedent of an implication. For program (1A), fairness is the weakest antecedent that guarantees the printing of a 1 without forbidding the printing of a 0. In practice, one would want a scheduling requirement for the **do** that bounds the number of times each of the two guarded commands can be executed in succession. Any such requirement implies fairness, which in turn implies that a 1 is eventually printed. Fairness is the property common to all such schedulers.

Our conclusion is that anyone who accepts the argument of [2], that fairness can be ignored, must also be prepared to ignore termination and all other liveness properties. And, ignoring liveness properties means that one is unable to distinguish between programs that ought to be distinguishable.

[1] Alpern, B., and F.B. Schneider. Defining liveness. *Information Processing Letters* 21, 4 (October 1985), 181-185.

[2] Dijkstra, E.W. Position paper on "Fairness". EWD 1013. Reprinted in *Software Engineering Notes 13*, 2 (April 1988), 18-20.