Information Processing Letters 40 (1991) 141–142
North-Holland

8 November 1991

Preserving liveness: Comments on "Safety and liveness from a methodological point of view"

Martín Abadi
Digital Equipment Corporation, Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, USA

Bowen Alpern
IBM, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, USA

Krzysztof R. Apt
Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, Netherlands

Nissim Francez and Shmuel Katz
Department of Computer Science, Technion – Israel Institute of Technology, Technion City, Haifa 32000, Israel

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

Fred B. Schneider *
Department of Computer Science, Cornell University, 4130 Upson Hall, Ithaca, NY 14853, USA

Communicated by D. Gries
Received 9 January 1991
Revised 26 June 1991 and 19 August 1991

Keywords: Distributed systems, specification methodology, safety, liveness property

Dederichs and Weber [4] define what it means for a property to be a liveness property with respect to a safety property. They argue that specifications should be written in the form $P \cap Q$, where Q is a liveness property with respect to the safety property P. They also criticize Alpern and Schneider's general definitions of safety and liveness [2]:

> Alpern and Schneider's characterizations are problematic, since they permit a certain kind of anomaly.

The anomaly is that a liveness property, which should constrain only infinite behavior, can implicitly rule out some finite behaviors.

* Supported by Office of Naval Research Contract N00014-86-K-0092, National Science Foundation No. CCR-8701103, and DARPA/NSF Grant No. CCR-9014363.

We agree that most reasonable specifications will be written in the form recommended by Dederichs and Weber. As observed by Abadi and Lamport [1], who called specifications having this form *machine closed*, one tries to write liveness properties that "[do] not rule out any finite behavior." As pointed out by Apt, Francez, and Katz [3], who defined a fairness condition for a programming language to be *feasible* if it produces machine-closed specifications for all programs, feasibility is necessary to "prevent a scheduler from 'painting itself into a corner' ".

We disagree with Dederichs and Weber's contention that non-machine-closed specifications should be avoided. We believe that it is neither desirable nor possible to do so.

Abadi and Lamport's completeness result [1] requires that only the lower-level implementation be machine closed, suggesting that there is no need for high-level specifications to be machine closed. Indeed, the general specification of serializability given by Lamport [5] achieves its simplicity by not being machine closed.

Even if one tried to forbid non-machine-closed specifications, they would arise in proofs that one specification implements another. A state-based proof that a lower-level specification Z implements a higher-level specification X is usually done in two steps. One first adds history and prophecy variables to Z to obtain an equivalent specification Y [1], and then one proves $Y \Rightarrow \overline{X}$, where $\overline{X}$ is obtained from X by substituting concrete realizations for abstract variables [5]. Surprisingly, it turns out that each of these steps can destroy machine closure, so Y and $\overline{X}$ need not be machine closed even if Z and X are. Although there are alternatives to using history and prophecy variables, substitution of concrete entities for abstract ones is fundamental, and it is likely that non-machine-closed specifications will arise in any approach that handles liveness.

As Dederichs and Weber observe, arbitrary liveness properties are "problematic". However, the problem lies in the nature of liveness, not in its definition.

Once cannot avoid complexity by definition.
Stephen Jay Gould

References

[1] M. Abadi and L. Lamport, The existence of refinement mappings, *Theoret. Comput. Sci.* **82** (2) (1991) 253–284; A preliminary version appeared in: *Proc. Third Ann. Symp. on Logic In Computer Science* (IEEE Computer Society, Edinburgh, Scotland, 1988) 165–177.

[2] B. Alpern and F.B. Schneider, Defining liveness, *Inform. Process. Lett.* **21** (4) (1985) 181–185.

[3] K.R. Apt, N. Francez and S. Katz, Appraising fairness in languages for distributed programming, *Distributed Comput.* **2** (1988) 226–241.

[4] F. Dederichs and R. Weber, Safety and liveness from a methodological point of view, *Inform. Process. Lett.* **36** (1) (1990) 25–30.

[5] L. Lamport, A simple approach to specifying concurrent systems, *Comm. ACM* **32** (1) (1989) 32–45.