Preserving Liveness: Comments on "Safety and Liveness from a Methodological Point of View"

Martín Abadi*, Bowen Alpern†, Krzysztof R. Apt‡, Nissim Francez§, Shmuel Katz§, Leslie Lamport*, and Fred B. Schneider¶

January 9, 1991
revised June 26, 1991 and August 19, 1991

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.

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' ".

*Digital Equipment Corporation, Systems Research Center
†I.B.M., T. J. Watson Research Center
‡C.W.I.
§Department of Computer Science, The Technion
¶Computer Science Department, Cornell University (Supported by Office of Naval Research contract N00014-86-K-0092, National Science Foundation Grant No. CCR-8701103, and DARPA/NSF Grant No. CCR-9014363.)

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.

One cannot avoid complexity by definition.
Stephen Jay Gould

References

[1] Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991. A preliminary version appeared in *Proceedings of the Third Annual Symposium on Logic In Computer Science*, pages 165-177, IEEE Computer Society, Edinburgh, Scotland, July 1988.

[2] Bowen Alpern and Fred B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.

[3] Krzysztof R. Apt, Nissim Francez, and Shmuel Katz. Appraising fairness in languages for distributed programming. *Distributed Computing*, 2:226–241, 1988.

[4] Frank Dederichs and Rainer Weber. Safety and liveness from a methodological point of view. *Information Processing Letters*, 36(1):25–30, October 1990.

[5] Leslie Lamport. A simple approach to specifying concurrent systems. *Communications of the ACM*, 32(1):32–45, January 1989.