Information Processing Letters 21 (1985) 181–185
North-Holland

7 October 1985

DEFINING LIVENESS*

Bowen ALPERN and Fred B. SCHNEIDER

Department of Computer Science, Cornell University, 405 Upson Hall, Ithaca, NY 14853, U.S.A.

Communicated by David Gries
Received 5 November 1984
Revised 20 February 1985

A formal definition for liveness properties is proposed. It is argued that this definition captures the intuition that liveness properties stipulate that 'something good' eventually happens during execution. A topological characterization of safety and liveness is given. Every property is shown to be the intersection of a safety property and a liveness property.

Keywords: Liveness, absolute liveness, uniform liveness, safety, property, topology, concurrency, semantics

1. Introduction

An execution of a concurrent program can be viewed as an infinite sequence of states:

$\sigma = s_0 s_1 \ldots$.

Each state after s_0 results from executing a single atomic action in the preceding state. (For a terminating execution, an infinite sequence is obtained by repeating the final state.) A *property* is a set of such sequences. Since a program also defines a set of sequences of states, we say that a property *holds* for a program if the set of sequences defined by the program is contained in the property.

It is useful to distinguish two classes of properties, since they are proved using different techniques. A proof that a program satisfies a *safety property* rests on an invariance argument [4], while a proof that a program satisfies a *liveness property* depends on a well-foundedness argument [6,7]. Safety and liveness were first described in [2]. The defining characteristic of safety properties was recently formalized in [3]. This paper gives a formal characterization of liveness properties and shows that all properties are the intersection of safety and liveness properties.

2. Safety properties

Informally, a safety property stipulates that some 'bad thing' does not happen during execution [2]. Examples of safety properties include mutual exclusion, deadlock freedom, partial correctness, and first-come-first-serve. In *mutual exclusion*, the proscribed 'bad thing' is two processes executing in critical sections at the same time. In *deadlock freedom* it is deadlock. In *partial correctness*, it is terminating in a state not satisfying the postcondition after having been started in a state that satisfies the precondition. Finally, in *first-come-first-serve,* which states that requests are serviced in the order they are made, the 'bad thing' is servicing a request that was made after one not yet serviced.

* This work was supported, in part, by NSF Grant DCR-8320274. F.B. Schneider was also supported by an IBM Faculty Development Award.

We now formalize safety.[1] Let S be the set of program states, S^ω the set of infinite sequences of program states, and S^* the set of finite sequences of program states. An execution of any program can be modeled as a member of S^ω. We call elements of S^ω *executions* and elements of S^* *partial executions* and write $\sigma \models P$ when execution σ is in property P. Finally, let σ_i denote the partial execution consisting of the first i states in σ.

For P to be a safety property, if P does not hold for an execution, then at some point some 'bad thing' must happen. Such a 'bad thing' must be irremediable because a safety property states that the 'bad thing' never happens during execution. Thus, P is a *safety property* if and only if the following holds.

Safety

$$(\forall \sigma : \sigma \in S^\omega : \sigma \not\models P \Rightarrow (\exists i : 0 \leqslant i : (\forall \beta : \beta \in S^\omega : \sigma_i \beta \not\models P))).$$

There are two things to notice about this definition. First, the definition does not restrict a 'bad thing' except to require that it be *discrete*—if the 'bad thing' happens during an execution, then there is an identifiable point at which it happens. Second, a safety property can never require that something happens sometime, as opposed to always. Thus, the definition merely stipulates that a safety property unconditionally prohibits a 'bad thing' from occurring and if it does occur, there is an identifiable point at which this can be recognized.

3. Liveness properties

Informally, a liveness property stipulates that a 'good thing' happens during execution [2]. Examples of liveness properties include starvation freedom, termination, and guaranteed service. In *starvation freedom*, which states that a process makes progress infinitely often, the 'good thing' is making progress. In *termination*, which asserts that a program does not run forever, the 'good thing' is completion of the final instruction. Finally, in *guaranteed service*,[2] which states that every request for service is satisfied eventually, the 'good thing' is receiving service.

The thing to observe about a liveness property is that no partial execution is irremediable: it always remains possible for the required 'good thing' to occur in the future.[3] We take this to be the defining characteristic of liveness since if some partial execution were irremediable, then it would be a 'bad thing'; liveness properties cannot proscribe a 'bad thing', they can only prescribe a 'good thing'.

We now formalize liveness. A partial execution α is *live* for a property P if and only if there is a sequence of states β such that $\alpha\beta \models P$. A *liveness property* is one for which every partial execution is live. Thus, P is a liveness property if and only if the following holds.

Liveness

$$(\forall \alpha : \alpha \in S^* : (\exists \beta : \beta \in S^\omega : \alpha\beta \models P)).$$

Again, there are two things to notice about this definition. First, the definition does not restrict what a 'good thing' can be; it does not even require that the 'good thing' be discrete. In starvation freedom the 'good thing'—progress—is an infinite collection of discrete events. In this way, 'good things' are fundamentally different from 'bad things'. Second, a liveness property cannot stipulate that some 'good thing' *always* happens, only that it eventually happens.

We believe that no definition of liveness can be more permissive than the one given above. Suppose, by way of contradiction, that P is a liveness property that does not satisfy our definition. There

[1] This formalization differs slightly from the one proposed in [3]. Under Lamport's assumption that properties are preserved under finite repetition of individual states ('stuttering'), both definitions are equivalent [1]. A property that is not invariant with respect to stuttering is "the value of x differs in any two successive states". We believe this should be considered a safety property, and it meets our definition but not Lamport's.

[2] This is called *responsiveness* in [5].

[3] "While there's life there's hope."—*Cicero*.

must be some partial execution α such that

$(\forall \beta : \beta \in S^{\omega} : \alpha\beta \not\models P)$.

Clearly, α is a 'bad thing' proscribed by P. Thus, P is in part a safety property and not a liveness property, as was assumed.

Obviously, definitions for liveness more restrictive than ours are possible. One candidate we have investigated is the following.

Uniform Liveness

$(\exists \beta : \beta \in S^{\omega} : (\forall \alpha : \alpha \in S^{*} : \alpha\beta \models P))$.

P is a uniform-liveness property if and only if there is a single execution (β) that can be appended to every partial execution (α) so that the resulting sequence is in P. Another definition has been proposed by Sistla [10].

Absolute Liveness

$(\exists \gamma : \gamma \in S^{\omega} : \gamma \models P)$
$\wedge (\forall \beta : \beta \in S^{\omega} : \beta \models P \Rightarrow (\forall \alpha : \alpha \in S^{*} : \alpha\beta \models P))$.

P is an absolute-liveness property if and only if it is nonempty and any execution (β) in P can be appended to any partial execution (α) to obtain a sequence in P.

It is instructive to contrast these formal definitions. L is a liveness property if any partial execution α can be extended by some execution β so that $\alpha\beta$ is in L—the choice of β may depend of α. U is a uniform-liveness property if there is a single execution β that extends all partial executions α such that $\alpha\beta$ is in U. And, A is an absolute-liveness property if it is nonempty and any execution β in A can be used to extend all partial executions α. Any absolute-liveness property is a uniform-liveness property and any uniform-liveness property is a liveness property.

While absolute liveness characterizes an interesting class of properties, we do not believe it includes all properties that should be considered liveness. Any *leads-to* property (e.g., guaranteed service) is not an absolute-liveness property. Such properties[4] are characterized as follows.

Leads-to. *Any occurrence of an event of type* E_1 *is eventually followed by an occurrence of an event of type* E_2.

When E_2 is satisfiable, such properties are liveness properties—E_2 is the prescribed 'good thing' [2]. To see that a leads-to property is not an absolute-liveness property, consider an execution β in which no event of type E_1 or E_2 happens. Leads-to holds on β. However, appending β to a partial execution consisting of a single event of type E_1 yields an execution that does not satisfy the property.

We also believe that uniform liveness does not correctly capture the intuition for liveness. An example of a liveness property that is not a uniform-liveness property is characterized as follows.

Predictive. *If* A *initially holds, then after some partial execution* B *always holds; otherwise, after some partial execution* B *never holds.*

We believe this to be a liveness property, because it requires some 'good thing' (either 'always B' or 'always $\neg$B') to happen eventually. It is not a uniform-liveness property since there is no *single* sequence that can succesfully extend all partial executions.

4. Other properties

Many properties are neither safety nor liveness. For example, any property characterized as follows: [5]

Until. *Eventually an event of type* E_2 *will happen and all preceding events are of type* E_1.

is the intersection of a safety property and a liveness property. The safety property is ''$\neg E_1$ before E_2' does not happen' and the liveness property is 'E_2 eventually happens'. Total correctness is also the intersection of a safety property (partial correctness) and a liveness property (termination). In fact, every property is the intersection of a

[4] These are the *eventuality properties* of Manna and Pnueli [5].

[5] In temporal logic this property is denoted by E_1 $\mathcal{U}$ E_2.

safety property and a liveness property. The proof of this (below) depends on a topological characterization of safety and liveness proposed by Plotkin[6] [9], who was motivated by Smyth [11].

There is a natural topology of S^ω in which safety properties are exactly the closed sets, and liveness properties (as defined above) are exactly the dense sets. The basic open sets of this topology are the sets of all executions that share a common prefix. As usual, an open set is the union of basic open sets, a closed set is the complement of an open set, and a dense set is one that intersects every nonempty open set. It is now possible to prove the following.

Theorem 1. *Every property* P *is the intersection of a safety property and a liveness property.*

Proof. Let $\bar{P}$ be the smallest safety property containing P and let L be $\neg(\bar{P} - P)$. Then,

$$\begin{aligned} L \cap \bar{P} &= \neg(\bar{P} - P) \cap \bar{P} \\ &= (\neg\bar{P} \cup P) \cap \bar{P} \\ &= (\neg\bar{P} \cap \bar{P}) \cup (P \cap \bar{P}) \\ &= P \cap \bar{P} = P. \end{aligned}$$

It remains to show that L is dense, and hence a liveness property. By way of contradiction, suppose there is a nonempty open set O contained in $\neg L$ and thus L is not dense. Then, $O \subseteq (\bar{P} - P)$. Consequently, $P \subseteq (\bar{P} - O)$. The intersection of two closed sets is closed, so $\bar{P} - O$ is closed and thus a safety property. This contradicts the hypothesis that $\bar{P}$ is the smallest safety property containing P. □

An obvious corollary of this is the following.

Corollary 1.1. *If a notation* Σ *for expressing properties is closed under complement, intersection, and topological closure, then any* Σ*-expressible property is the intersection of a* Σ*-expressible safety property and a* Σ*-expressible liveness property.*

[6] Plotkin nevertheless is unhappy with our definition of liveness because it is not closed under intersection.

Thus, in order to establish that every property P expressible in a temporal logic can be given as the conjunction of a safety property and a liveness property expressed in the logic, is suffices to show that the smallest safety property containing P is also expressible in the logic.

Plotkin has shown that any property that can be expressed in temporal logic can be written as the conjunction of two temporal logic expressible liveness properties [8]. In fact, a more general result can be proven.

Theorem 2. *If* $|S| > 1$, *then any property* P *is the intersection of two liveness properties.*

Proof. By hypothesis, there are two states a and b in S. Let L_a (respectively L_b) be the set of all executions with tails that are an infinite sequence of a's (respectively b's). Both L_a and L_b are liveness properties and $L_a \cap L_b = \Phi$. Now,

$$\begin{aligned} &(P \cup L_a) \cap (P \cup L_b) \\ &\quad = (P \cap P) \cup (P \cap L_a) \cup (P \cap L_b) \cup (L_a \cap L_b) \\ &\quad = P. \end{aligned}$$

The union of any set and a dense set is dense, so $P \cup L_a$ and $P \cup L_b$ are liveness properties and the theorem is proven. □

As before, there is an obvious corollary.

Corollary 2.1. *If a notation* Σ *for expressing properties is closed under intersection and there exist* Σ*-expressible liveness properties with empty intersection, then any* Σ*-expressible property is the intersection of two* Σ*-expressible liveness properties.*

Topology also provides a convenient framework for investigating the closure of safety and liveness under boolean operations. Safety properties (closed sets) are closed under union and intersection. Liveness properties (dense sets) are closed only under union. Neither is closed under complement. Finally, the only property that is both safety and liveness is S^ω itself.

5. Conclusion

A formal definition of liveness should be as general as possible without doing violence to the intuition. We have argued that no definition that is less restrictive than ours corresponds to this intuition. Any argument for a more restrictive formal definition should include an example of a property that meets our definition, but that does not seem to be a liveness property. It appears to us that any such definition will unduly restrict what a 'good thing' can be, but we cannot prove this.

It seems naive to hope for a proof that a formal definition for liveness (or safety) is correct, because 'good things' and 'bad things' are not well-defined concepts. However, the simple topological characterization of the definitions suggests that they do indeed capture fundamental distinctions.

Acknowledgment

We wish to thank A. Demers, D. Gries, M. Krentel and E. Mallison for comments on an early draft of this paper. Comments from G. Plotkin on two separate occasions forced us to understand the consequences of our liveness definition and are gratefully acknowledged. Thanks are also due to J. Hook, L. Lamport, A. Pnueli, and A.P. Sistla for helpful discussions and encouragement concerning this work.

References

[1] B. Alpern, F.B. Schneider and A.J. Demers, A note on safety without stuttering, In preparation, 1985.

[2] L. Lamport, Proving the correctness of multiprocess programs, IEEE Trans. Software Engineering SE-3 (2) (1977) 125–143.

[3] L. Lamport, Basic concepts, in: Advanced Course on Distributed Systems—Methods and Tools for Specification, Lecture Notes in Computer Science 190 (Springer, Berlin–Heidelberg, 1984).

[4] L. Lamport and F.B. Schneider, The 'Hoare Logic' of CSP and all that, ACM Trans. Programming Languages and Systems 6 (2) (1984) 281–296.

[5] Z. Manna and A. Pnueli, Temporal verification of concurrent programs: The temporal framework for concurrent programs, in: R.S. Boyer and J.S. Moore, eds., The Correctness Problem in Computer Science (Academic Press, London, 1981).

[6] Z. Manna and A. Pnueli, Verification of concurrent programs: Proving eventualities by well-founded ranking, Tech. Rept. STAN-CS-82-915, Dept. of Computer Science, Stanford Univ., 1982.

[7] S. Owicki and L. Lamport, Proving liveness properties of concurrent programs, ACM Trans. Programming Languages and Systems 4 (3) (1982) 455–495.

[8] G. Plotkin, Private communication, 1983.

[9] G. Plotkin, Private communication, 1984.

[10] A.P. Sistla, Characterization of safety and liveness properties in temporal logic, Extended abstract, Univ. of Massachusetts at Amherst, MA, 1984.

[11] M.B. Smyth, Power domains and predicate transformers: A topological view, in: Proc. ICALP '83, Lecture Notes in Computer Science 154 (Springer, Berlin–New York, 1983) 662–675.