Information Processing Letters 35 (1990) 277–279
North-Holland

15 September 1990

LETTER TO THE EDITOR

COMMENTS ON "ON THE PROOF OF A DISTRIBUTED ALGORITHM": ALWAYS-TRUE IS NOT INVARIANT

A.J.M. VAN GASTEREN * and G. TEL **

Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands

Communicated by W.M. Turski
Received 18 June 1990

We would like to report an error in a proof given by J.-P. Verjus in [3]. More importantly, we would like to point out a difference between the notions "always-true" and "invariant", which are identified in [3]. In his paper, Verjus claims that invariance proofs by contradiction are shorter than invariance proofs by the established method, called the "induction method" in [3]. To illustrate his idea, he proposes an alternative correctness proof for a termination detection algorithm given by Dijkstra, Feijen, and Van Gasteren [2]. We show that his—operational—proof is incorrect and incomplete (Section 1); moreover, what he shows is that a predicate is always-true, which is strictly weaker than invariance (Section 2). Instead of illustrating the merits of proofs by contradiction, the proof strongly supports our opinion that operational arguments should be avoided.

1. Analysis of Verjus' proof

The algorithm consists of (atomic) actions $\mathbf{R}_0$ through $\mathbf{R}_4$, described in [3]. The predicate to be proved invariant is $\mathbf{P}_0 \vee \mathbf{P}_1 \vee \mathbf{P}_2$, where

$P_0 \equiv \forall i,\ t < i < n$: m_i is passive,

$P_1 \equiv \exists j,\ 0 \leqslant j \leqslant t$: m_j is black,

$P_2 \equiv$ the token is black.

To prove that this predicate always holds during every execution of the algorithm, the author assumes a state in which the negation of the predicate holds and arrives at a contradiction as follows. (We number the sentences for easy referencing.)

(0) Let m_i be the highest numbered active machine.

(1) The machine m_i was passive (see $\mathbf{R}_0$) and it has transmitted the token.

(2) It has been reactivated by receiving a message from a machine m_j with a number lower than its own ($j < i$).

(3a) This machine m_j has become black (see $\mathbf{R}_1$) and,

(3b) by virtue of $\mathbf{R}_2$ and $\neg\mathbf{P}_2$ (the token is white),

(3c) $j \leqslant t$, which contradicts $\neg\mathbf{P}_1$. □

The proof contains the following two errors.

The conclusion $j < i$ in sentence (2) is drawn erroneously, probably from the fact that according to the rules m_j was active when it sent the message to m_i, and m_i is the highest numbered active machine. It is possible, however, that machine m_i has been activated by a machine m_j with $j > i$ that has since become passive: by (0), m_i is the highest numbered active machine in the *current* state, but this does not exclude that in an *earlier* state there was an active machine with an even higher number.

The conclusion $j \leqslant t$, meaning that m_j has not (yet) propagated the token, in sentence (3c) is drawn erroneously, in the following way. Machine

* The work of this author was supported by the Netherlands' Organization for Scientific Research (NWO) under contract no. NF-63/62-518 (Project STOP).
** The work of this author was supported by the ESPRIT II Basic Research Actions Program of the EC under contract no. 3075 (Project ALCOM).

m_j has become black (3a), the token is white (3b), and black machines transmit the token black ($\mathbf{R}_2$). It is possible, however, that m_j transmitted the token *before* becoming black, in which case $j > t$ but blackening of the token has not necessarily taken place. This execution sequence was overlooked, and it is not excluded by any of the assumptions previously made.

It is not possible for us to provide a *counterexample* to show that the conclusions drawn in the proof are wrong. The invariance of the predicate was demonstrated in [2] and thus the conclusions, although drawn erroneously, are themselves vacuously correct. We have clearly shown, however, that the author has not succeeded, as he claims, in providing an alternative proof for the invariance.

The errors are a consequence of an inaccurate formulation of the proof obligations. The author aims at establishing that a state in which the invariant does not hold *cannot exist* (see p. 145, first column). This formulation is not correct, because in general such states *do* exist; what needs to be shown is that they are *not reachable* from the initial state by a sequence of state transitions. The latter formulation *explicitly* relates the proof obligations to the state transitions of the algorithm. Verjus' proof obligations seem to be independent of the transitions, but his proof introduces them *implicitly* by reasoning about execution sequences.

Finally we comment on what Verjus writes on p. 145: "From a combinatorial point of view, this static method results in a great simplification." We think that, rather than the claimed efficiency of his method, the following two circumstances contribute to the brevity of his proof. First, as we have shown above, the proof neglects possible execution sequences. Second, it does not show in detail that in the cases considered the execution sequences are as stated. For example, there is no proof that m_j does not turn white between becoming black and propagating the token. Such a proof would involve *all* possible state transitions. Indeed, any invariance proof involves all these transitions. Thus, if Verjus' proof were completed it would probably become much longer; we therefore disagree with the cited claim.

2. Always-true is not invariant

Verjus claims to show the invariance of the predicate, but in fact he only shows that the predicate is not falsified in any execution of the program. In this section we demonstrate that the latter is weaker than the former, and that the stronger notion is to be preferred.

In our model a program is a tuple $V = (S, I, R)$, where S is a set of *states*, I a subset of S of *initial states* and R a set of *actions* (see, for example, [1]).

Definition 1. A predicate P on S is an *invariant* (of program V) if

(1) P holds in every initial state; and

(2) P is maintained by every action, that is, $\forall r \in R$: $\{P\}\, r\, \{P\}$.

A predicate P on S is *always-true* (in program V) if it holds in each reachable state (that is, a state reachable from an initial state by a finite sequence of actions).

Theorem 2. *If P is an invariant, then P is always-true.*

Theorem 2 is proved by induction on the length of a sequence of actions leading to a reachable state. (Consequently, some authors use the term "inductive proof" for a proof that P is invariant.) The converse of Theorem 2 is not true. To see this, consider the following example program **A** with one integer variable k, which is 0 initially. There is only one action:

if $k = 1$ **then** $k := 2$.

Let **F** be the predicate $k < 2$. **F** is *always-true* in **A**, but **F** is not an invariant, because the (unreachable) state $k = 1$, where **F** holds, is transformed into the state $k = 2$, where **F** does not hold. Thus the notion *always-true* is weaker than the notion of invariance.

We believe that of the two notions invariance is the more useful one, because *always-true* is not preserved under composition of programs. For programs $V = (S, I, R_V)$ and $W = (S, I, R_W)$, the (parallel) *composition* of V and W is $(S, I, R_V \cup R_W)$.

Theorem 3. *If P is an invariant of V and P is an invariant of W, then P is an invariant of the composition of V and W.*

Proof. By the definition of invariance and distribution of universal quantification over set union. □

The notion *always-true*, however, is not preserved under composition. To see this, consider program **A** above and program **B** with the same states and initial states as **A** and also one action:

$k := 1.$

Predicate **F** above is *always-true* in **A** and *always-true* in **B**, but it is not *always-true* in the composition of **A** and **B**.

3. Conclusion

Verjus advocates a proof method based on reasoning by contradiction, illustrating it with an example. His method, however, encourages the use of operational arguments, which, to our experience, bears the risk of making errors. The example proof supports this experience, as is shown in Section 1. We also refuted the claim of efficiency of the method. Furthermore, proofs by Verjus' method show that a predicate is *always-true* rather than that it is an invariant. This makes the method invalid for parallel program composition, as is shown in Section 2. To conclude, the method has none of the advantages claimed by Verjus; on the contrary, compared to the established method of proving invariance it has only disadvantages.

References

[1] K.M. Chandy and J. Misra, *Parallel Program Design, A Foundation* (Addison-Wesley, Reading, MA, 1988).

[2] E.W. Dijkstra, W.H.J. Feijen and A.J.M. Van Gasteren, Derivation of a termination detection algorithm for distributed computations, *Inform. Process. Lett.* **16** (1983) 217–219.

[3] J.-P. Verjus, On the proof of a distributed algorithm, *Inform. Process. Lett.* **25** (1987) 145–147.