(1977 / 71)

Informal and Formal Correctness Proofs for Programs
(For the Critical Section Problem.)

Abstract

Uri Abraham and Shai Ben-David
Ben Gurion University and Cornell University; Toronto University

1. Introduction. Proofs in mathematics can have different aspects, but two are prominent. The first one, of course, is that a proof is used to validate a statement - to establish a theorem. Yet there might be another aspect of a proof: A proof is a way to convey ideas and to explain why things work. So, for exmaple, when we see a proof of Pythagoras' theorem not only are we convinced that the theorem holds, but we also learn something else about triangles; we have a better understanding of right angle triangles. Usually, there is no sharp distinction between those two aspects and they are blended together - sometimes with more emphasis on the one than on the other. So when a mathematician wants to explain the idea of a proof he or she do not present a formal proof, but outline a description of a formal proof or of what can be transformed into such. Usually, arguments are given in order to explain and to communicate, and sometimes examples are given or pictures. Then, if more details are needed, they are provided; until, at least in theory, a complete formal proof can be given - one which is the counterpart of the informal one.

Unfortunately, the situation does not seem to be like that in computer science. The proofs which we have seen for showing the correctness of parallel algorithms are hard to follow. And very often even after the sequence of statements which constitute the proof have been followed line by line, the algorithm remains engimatic as before. (See for example the interesting Manna and Pnueli [1984].)

We feel that there is a need for a way to present intuitive and informal proofs for program correctness, and in a form which can easily be altered to a formal proof if need arises. The frame for correctness proofs should be large and flexible enough to allow both informal and formal proofs. We do not have an answer to this need and our aim is moderate: to explain by means of some examples what we mean by "an intuitive proof" and how a solid formal background can support such a proof. The examples we give are taken from the area known as "the critical section problem". The algorithms we inspect are simple and complex at the same time. Simple in that a few lines suffice to describe them. And complex, because when several processes execute simultaneously it is difficult to imagine and control all possible interleavings and connections between the processes. So one would like a general guiding idea to make sense out of this apparently arbitrary movement.

2. The Critical Section Problem.

This problem arises when several processes that run in parallel, or in a time sharing environment, want from time to time to access a resource (such as a printer) that cannot be shared between two processes. The solution consists of a protocol (a piece of a program) that each process has to execute before and after entering this "critical section", and such that the following hold:

2.1. A safety property: No two processes are at the same time in their critical section.

2.2 A liveness property: (for example) any process that wants to access the critical section can do that after some time.

We shall concentrate here only on the safety property and the case of only two processes.

Let us describe Peterson's algorithm for two processes A and B. There is a global variable that can be read and written by any of the two processes and which can hold any of the two values: A or B. We call this global variable Last_visited. Initially it can be of any value. Then there are two boolean variables, A_Wants_To_Enter and B_Wants_To_Enter, which are initially false. They can be read by both processes but are written only by A and B respectively. Also, each process maintains a local variable for any global one. If X is a global variable then we let $(X)^A$ and $(X)^B$ denote the local variables. Now the protocol for process A is given:

```
ℓ_0   part of program which does not involve the critical section;
ℓ_1   A_Wants_To_Enter := true;
ℓ_2   Last_Visited := A;
ℓ_3   wait until
(*)   (B_Wants_To_Enter)^A = false or (Last_Visited)^A = B;
ℓ_4   critical section;
ℓ_5   A_Wants_To_Enter := false;
ℓ_6   go to ℓ_0;
```

Process B algorithm is symmetrically obtained.

Observe that line ℓ_3 is not an atomic operation, but in fact can be decomposed in the following way:

$\ell_{3.1}$ (B_Wants_To_Enter)^A := B_Wants_To_Enter;

$\ell_{3.2}$ (Last_Visited)^A := Last_Visited;

$\ell_{3.3}$ compute the truth value of (*)

$\ell_{3.4}$ If the result in $\ell_{3.3}$ is false then jump to $\ell_{3.1}$;

Now we want to "explain" Peterson's algorithm in the following intuitive terms. For process A condition (*) for entering the critical section is equivalent to the conjunction of statements 1_A and 2_A.

(1_A) Process A wants to enter the critical section and it knows that, no matter how the programs develop, provided it does not reset its wish to enter, then if process B executes its protocol it finds (or will find) that A wants to enter.

(2_A) A knows that as long as it still wants to enter the critical section, then (1_B) is false. (1_B is the equivalent of 1_A with A and B interchanged.)

In order to prove that this is indeed the case we must set an appropriate frame in which the terms "A know" etc. can be given a clear and definite meaning. We outline this in the next section for a time sharing environment.

3. <u>The Model</u>. The setting is for two processes A and B working in a time-sharing environment, and running some fixed programs. A <u>state</u> is defined to be a description of all relevant information holding at a particular moment. So a state includes the values of all variables, the places of the program counters (of the different processes), and which of the processes was lastly

executing. PC(A) and PC(B) are the values of the program counters of A and B. These values are in $[\ell_0 - \ell_6]$.

On the collection of all states we can define the following relations: (s and t are states)

t is a successor of s ≡ execution of a single step can lead from s to t.

$s < t$ (t develops from s) ≡ there is a computation of zero or more steps that leads from s to t.

$s_1 R_A s_2$ ≡ states s_1 and s_2 are indistinguishable by process A, i.e., have the same values for the local variables of A.

$s_1 R_B s_2$ ≡ as above, but for B.

For any statement φ (in fact we shall use only very simple φ's)

$s_1 <^{\varphi} s_2$ ≡ there is a computation taking state s_1 into s_2 (in zero or more steps) such that all states involved satisfy φ.

There is an initial state from which all states are developed.

Now to these accessibility relations there corresponds the modal operators $\Box_A$ (for R_A), $\Box_B$ (for R_B), $\Box_<$ and $\Box^{\varphi}$ (for $<^{\varphi}$). The meaning of these operators and the definition of the satisfaction relation $s \models \varphi$ is done as usual for Kripke's models. The intuitive meaning of $s \models \Box_A \varphi$ is "in state s, process A knows that φ". (See .)

Now we can write down the precise formulations of 1 and 2.

(1_A) A_Wants_To_Enter = true and $\Box_A$ $\Box$A_Wants_To_Enter = true (if PC(B) $\in [\ell_{3.1} - \ell_4]$ then $(\text{A\_Wants\_To\_Enter})^B$ = true).

(2_A) $\Box_A$ $\Box$A_Wants_To_Enter = true$(\neg(1_B))$.

With this background we can prove the following:

Theorem. For any state s, if process A (or B) is in $\ell_{3.3}$, then (*) is equivalent to 1_A and 2_A (1_B and 2_B).

The proof will be given elsewhere. It is easy to see that the theorem implies the safety property. We feel that this approach explains intuitively as well as rigorously Peterson's algorithm (but of course it does not diminish the ingenuity taken to find that algorithm).

4. On Lamport's Axioms.

In accordance with our aim, we prefer to argue with models and relations rather than through axioms. However, in order to ensure that a formal proof is always present, it is good to have a complete set of axioms. For then what is true can also be derived from the axioms.

Now Lamport [1986, Part I] suggested some axioms and frame to argue about concurrent processes. We shall add an axiom and prove that the resulting set is complete. But first we present some definitions and notions of Lamport.

A realization[1] for a system execution consists of a partial (irreflexive) order, $<$, defined on a set P and of a collection C of non empty subsets of P. The following two relations are then defined on C.

(1) $x \rightarrow y \equiv (\forall p \in x)(\forall q \in y)\ p < q$.

(2) $x \dashrightarrow y \equiv (\exists p,q)(p \in x \text{ and } q \in y \text{ and } p < q)$.

We want to find axioms for the theory of models of the form $(C, \rightarrow, \dashrightarrow)$ arising as above. Lamport suggested the following axioms A_1 to A_4 (and other axioms which do not concern this note). We suggest to add A_5 as well. (It is easy to see that A_5 is not a consequence of A_1-A_4.)

(A_1) The relation $\rightarrow$ is an irreflexive partial ordering.

(A_2) If $A \rightarrow B$ then $A \dashrightarrow B$ and $B \not\dashrightarrow A$.

(A_3) If $A \rightarrow B \dashrightarrow C$ or $A \dashrightarrow B \rightarrow C$ then $A \dashrightarrow C$.

[1]Lamport uses the term model.

(A_4) If $A \to B \dashrightarrow C \to D$ then $A \to D$.

(A_5) If $A \dashrightarrow B \to C \dashrightarrow D$ then $A \dashrightarrow D$.

Now we define, as Lamport does, a system execution to be a structure with abstract relations $\mathscr{S} = (S, \to, \dashrightarrow)$ satisfying axioms A_1 to A_5. A <u>representation</u> for such a structure is a pair $\rho = (\mu, (P,<,C))$ where μ is a function defined on S with values in C, and $(P,<,C)$ is a realization such that the following holds for any $x, y \in S$.

(i) $x \to y$ (in $\mathscr{S}$) $\equiv \mu(x) \to \mu(y)$ (in C).

(ii) $x \dashrightarrow y \equiv \mu(x) \dashrightarrow \mu(y)$.

<u>Theorem</u>. Any system execution has a representation.

<u>Proof</u>. So let $\mathscr{S} = (S, \to, \dashrightarrow)$ be a system execution (i.e., an abstract structure satisfying axioms A_1 to A_5). We need first three definitions: Let $R = (P,<,C)$ be any realization. (So $<$ is a partial ordering and $C \subseteq \mathscr{P}(P) - \{\phi\}$.) Let $\mu : S \to C$ be a function.

<u>Definition</u>. (a) We say that (μ, R) is <u>almost a representation</u> of $\mathscr{S}$ iff

(1) $x \to y \Rightarrow \mu(x) \to \mu(y)$

(2) $x \dashrightarrow y \equiv \mu(x) \dashrightarrow \mu(y)$

(b) We call (μ, R) a <u>partial representation</u> iff

(1) $x \to y \Rightarrow \mu(x) \to \mu(y)$

(2) $x \dashrightarrow y \Leftarrow \mu(x) \dashrightarrow \mu(y)$

(c) We say that a representation (or almost or partial representation) is <u>pairwise disjoint</u> if

$$x \neq y \text{ in } S \Rightarrow \mu(x) \cap \mu(y) = \phi.$$

Now (for clearness) the construction of a representation for $\mathscr{S}$ is done in two stages: In the first stage an almost representation is obtained, and then a representation.

__Definition__. Let $\rho_i = (\mu_i, R_i)$, $i = 1,2$, be two partial representations for ρ. (So $R_i = (P_i, <_i, C_i)$ and $\mu_i : S \to C_i$.) We say that ρ_2 extends ρ_1 ($\rho_1 < \rho_2$) iff:

(1) $P_1 \subseteq P_2$ and $<_1 \subseteq <_2$ (i.e., $a <_1 b$ implies $a <_2 b$).

(2) For all $x \in S$, $\mu_1(x) \subseteq \mu_2(x)$.

For any system execution $\mathscr{S}$, there is the identity partial representation. It is defined as $(\mu, (S, \to, \text{singletones})$ where $\mu(x) = \{x\}$.

Now, let $\rho = (\mu, (P, <, C))$ be any pairwise disjoint partial representation. We want to extend ρ to an almost representation ρ'. For any $x, y \in S$ with $x \dashrightarrow y$, add to P two new elements $a = a(x,y)$, $b = b(x,y)$. Let P' be the set thus obtained, and define:

$$\mu'(x) = \mu(x) \cup \{a \mid a = a(x,y) \text{ for some } y \text{ such that } x \dashrightarrow y\} \cup$$
$$\cup \{b \mid b = b(y,x) \text{ for some } y \dashrightarrow x\}.$$

Now, let $<^*$ be the extension of $<$ obtained by adding all relations $a(x,y) <^* b(x,y)$ for $x \dashrightarrow y$. And let $<^{**}$ be the relation obtained by adding to $<^*$ all pairs in $\mu'(x) \times \mu'(y)$ when $x \to y$ in S. Finally, let $<'$ be the transitive closure of $<^{**}$.

__Claim__. $<'$ is irreflexive partial order on P'.

__Proof__. We make first some observations on $<^{**}$.

(1) If $a <^{**} b$, $a \in \mu(u)$ and $b \in \mu(v)$, then if both a and b are in P, then $a < b$. If exactly one of them is new, then $u \to v$. And if both are new then $u \dashrightarrow v$.

(2) If $a <^{**} b <^{**} c$ are in P', and u, v, w are such that $a \in \mu(u)$, $b \in \mu(v)$ and $c \in \mu(w)$, then either $u \to v \dashrightarrow w$ or $u \dashrightarrow v \to w$.

(3) Hence assume $p <' q$, and let $p = p_0 <^{**} p_1 \dots <^{**} p_{n-1} = q$ be an increasing sequence in $<^{**}$ of minimal length going from p to q. This sequence does not contain three consecutive members of P. So if we let $w_i \in S$ be such that $p_i \in \mu(w_i)$, then the w_i's are connected by arrows that alternate: $\rightarrow$ and $\dashrightarrow$.

Now to show that $<'$ is irreflexive, we have to prove that there are no cycles in $<^{**}$. If there was a cycle, then pick one of minimal length: $p = p_0 <^{**} \dots <^{**} p_{n-1} = p$. Let $w_i \in S$ be with $p_i \in \mu(w_i)$. $n = 2$ is not possible. We can assume that the first arrow, from p_0 to p_1 is $\rightarrow$.

<u>Case 1</u>. n is even. Use A_4 to get $w_0 \rightarrow w_{n-1}$. For example, if $n = 4$ then $w_0 \rightarrow w_1 \dashrightarrow w_2 \rightarrow w_3$ and then $w_0 \rightarrow w_3$. But this is a contradiction, as $w_0 = w_{n-1}$.

<u>Case 2</u>. n is odd. Then by A_4, $w_0 \rightarrow w_{n-2}$. But we also know that $w_{n-2} \dashrightarrow w_{n-1} = w_0$. This contradicts A_2.

<u>Claim</u>. μ', P', $<'$ form an almost representation.

<u>Proof</u>. The only nontrivial part is to show that $\mu(x) \dashrightarrow \mu(y)$ implies $x \dashrightarrow y$. So assume $p \in \mu(x)$, $q \in \mu(y)$ and $p <' q$. Pick a minimal length sequence increasing in $<^{**}$ from p to q. $p = p_0 <^{**} \dots <^{**} p_{n-1} = q$. Let $w_i \in S$ be such that $p_i \in \mu(w_i)$. So $w_0 = x$ and $w_{n-1} = y$.

<u>Case 1</u>. The first arrow is $\rightarrow$. So $p_0 \rightarrow p_1$. Then use of A_4 and A_3 gives that $w_0 \dashrightarrow w_{n-1}$. So $x \dashrightarrow y$.

<u>Case 2</u>. The first arrow is $\dashrightarrow$. Then use of A_5 and A_3 gives that $w_0 \dashrightarrow w_{n-1}$. For example, if $w_0 \dashrightarrow w_1 \rightarrow w_2 \dashrightarrow w_3 \rightarrow w_4$, then $w_0 \dashrightarrow w_3$ by A_5, and then also $w_0 \dashrightarrow w_4$ by A_3. Again $x \dashrightarrow y$.

Now we want to show how to get from an almost representation to a representation for $\mathcal{S}$. So let $\rho = (\mu,(P,<,C))$ be an almost representation which is pairwise disjoint. What could stop ρ from being a representation is that $\mu(x) \to \mu(y)$ can hold for some $x,y \in S$ with $x \to y$. To define P' we add to P new members $a(x)$ for every $x \in S$. We define $\mu'(x) = \mu(x) \cup \{a(x) | x \in S\}$.

Now we define $\overset{*}{<}$ by adding to $<$ the following relations: for any $u,v \in S$ such that $u \to v$ add

$$\mu'(u) \times \mu'(v).$$

We let $<'$ be the transitive closure of $\overset{*}{<}$. Then we have to argue that $<'$ is irreflexive and that we get a representation $(\mu'(P',<',C'))$ for $\mathcal{S}$. All is a consequence of the following observations:

(1) If $a \overset{*}{<} b$, $a \in \mu(z)$ and $b \in \mu(w)$ and a or b is a new element (not in P), then $z \to w$. If both are in P then $a < b$.

(2) Suppose that $p <' q$ and let $p_0 = p \overset{*}{<} p_1 \ldots \overset{*}{<} p_{n-1} = q$ be a minimal length sequence. Say $w_i \in S$ are such that $p_i \in \mu(w_i)$. There are no three consecutive members of P in this sequence. But also there are no three consecutive new members. Because if a_i, a_{i+1}, a_{i+2} are new, then $w_i \to w_{i+1} \to w_{i+2}$, and so $w_i \to w_{i+2}$ and hence $a_i \overset{*}{<} a_{i+2}$ and the sequence could be shortened. It follows that for any i, $w_i \to w_{i+1}$ or $w_i \dashrightarrow w_{i+1}$ and these arrows alternate.

(3) Suppose $p <' q$ and p_i, w_i are as above. We shall prove by induction on n that $w_0 \dashrightarrow w_{n-1}$ and that if both p and q are new, then $w_0 \to w_{n-1}$. For $n = 2$ this follows from (1) above. So now $n > 2$. If either p_{n-1} or p_{n-2} are new, then

$w_{n-2} \to w_{n-1}$, and as $w_0 \dashrightarrow w_{n-2}$ by induction, $w_0 \dashrightarrow w_{n-1}$ by A_3. If both p_{n-1} and p_{n-2} are old, then p_{n-3} is new; so $w_0 \dashrightarrow w_{n-3} \to w_{n-2} \dashrightarrow w_{n-1}$ and A4 implies $w_0 \dashrightarrow w_{n-1}$. Now if p_0 and p_{n-1} are new, then $w_0 \to w_1$ and $w_{n-2} \to w_{n-1}$. But $w_1 \dashrightarrow w_{n-2}$ by induction. So A4 implies $w_0 \to w_{n-1}$.

Now, to show that $<'$ is irreflexive we should prove that $<^*$ has no cycles. But a cycle must contain a new element, and so we can assume that it begin with a new element $p_0 <^* p_1 \ldots <^* p_{n-1} = p_0$. By (3) above $w_0 \to w_0$. A contradiction.

To prove that μ' is a representation, we have to show that

(1) $\mu'(x) \to \mu'(y)$ implies $x \to y$.

(2) $\mu'(x) \dashrightarrow \mu'(y)$ implies $x \dashrightarrow y$. (The other directions are trivials.)

To prove (1) observe that if $\neg(x \to y)$ and $x \neq y$ then $\mu'(x)$ and $\mu'(y)$ contain two new elements $p \in \mu'(x)$ and $q \in \mu'(y)$. So by (3) above $\neg(p <' q)$. For (2), assume $p \in \mu'(x)$ and $q \in \mu'(y)$ and $p <' q$. Then by (3) above, $x \dashrightarrow y$.

References

L. Lamport [1986]. On interprocess communication, Part I, Distributed Computing (1986), 1, pp.77-85.

Z. Manna and A. Pnueli [1984]. Adequate proof principles for invariance and liveness properties of concurrent programs, Science of Computer Programming (1984), 4, pp.257-289.