Verifikation nebenläufiger Programme
Wintersemester 2004/05

Ulrich Hannemann Jan Bredereke

Example

In this section we illustrate the method of Owicki & Gries by proving mutual exclusion of different algorithms. First we investigate the general structure of mutual exclusion algorithms where two processes should be coordinated, leading to the development of Dijkstra's mutual exclusion algorithm [Dij65, Dij68] and a proof of its correctness using the method of Owicki & Gries.

The mutual exclusion property we want to establish for these algorithms cannot be expressed directly as a partial-correctness formula. Consequently, we do not use the whole of the method of Owicki & Gries, since that proves partial-correctness formulae of the $\{pre\}P_1||\ldots||P_n\{post\}$-type, but only part of it. We only need the notion of inductive assertion networks in the way these are constructed according to the method of Owicki & Gries, since these express properties which hold at the locations of processes. We shall prove that these properties, when applied to the locations of the critical sections of each process, imply mutual exclusion.

In general, the mutual exclusion problem is present in a system of concurrently executing processes where some common resources can only be used by one process at-a-time. For simplicity, we model these processes as infinite loops alternating between a *critical section* and a *non-critical section*. The general idea to enforce this property is to introduce *pre- and postprotocols* to the involved processes to ensure mutual exclusion.

In order to model mutual exclusion algorithms in terms of transition diagrams for two processes we use the following conventions.

- For each process P_i, $i = 1, 2$ we introduce a boolean variable cs_i as *observable,* with the intention that the value of cs_i is tt if and only if P_i is in its critical section.
- A program never starts in its critical section; we thus identify the beginning of the noncritical section with the entry location.
- Infinitely looping programs never reach their exit location; this explains the transition $(s, false \rightarrow id, t)$.

We first consider systems which are composed of two concurrent processes P_1 and P_2. We say that they *satisfy the mutual exclusion property* if their parallel execution $P_1 \| P_2$ ensures that $\neg(cs_1 \wedge cs_2)$ always holds, i.e., P_1 and P_2 are never in their critical sections simultaneously – for the transition diagrams as in Figure 1 this obviously does not hold.

Example 1: Consider the following approach: For each process P_1 and P_2 a (boolean) variable req_i, $i = 1, 2$ is introduced to indicate that process P_i requests access to its critical section. These variables are shared, but process

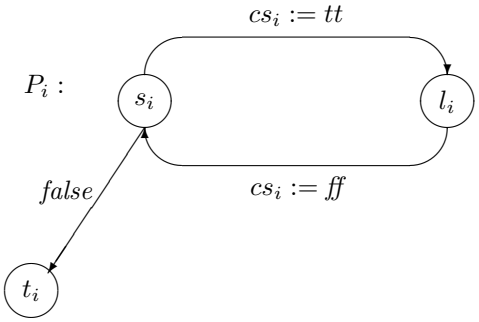

Figure 1: General structure of a mutual exclusion algorithm.

P_i has exclusive write-access to its own request flag req_i. Process P_j, $i \neq j$, can read req_i, but it cannot change the value of req_i.

To be sure that mutual exclusion holds we require that no process enters the critical section while the other process has already requested entrance to its critical section. Although we have satisfied the mutual exclusion requirement

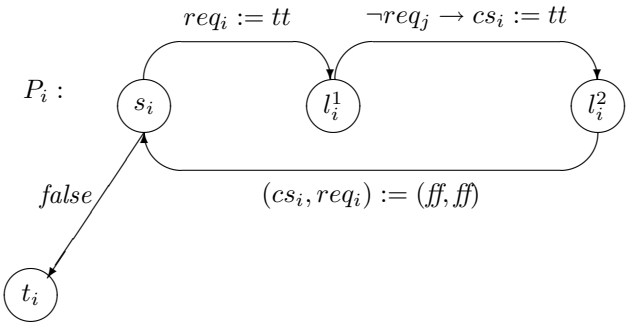

Figure 2: A deadlock solution.

(not formally proven), this "solution" suffers from the possibility of running into a "deadlock" situation when P_i sets its request flag and does not enter the critical section immediately, but instead P_j sets its request flag; then both req_1 and req_2 are $true$ and neither P_1 nor P_2 can proceed. □

Example 2: (Dijkstra's Mutual Exclusion Algorithm) To avoid this possibility of deadlock, another location is added to each process, setting req_i to ff during a certain period so that P_j can reach its critical section and set its flag req_j to ff after executing that section. Now we want to establish that $P_1 \| P_2$ in Figure 3 satisfies the mutual exclusion property. As precondition we characterize a safe state for the program variables, stating that no process is in its critical section and no process requested access to its critical section.

$$\varphi = \neg cs_1 \wedge \neg cs_2 \wedge \neg req_1 \wedge \neg req_2$$

Since the transition diagram resulting from parallel composition in terms of their product would consist of 25 location and 60 transitions, we give a proof

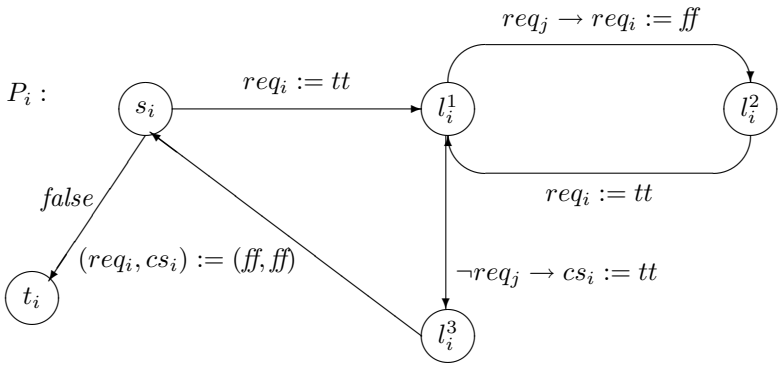

Figure 3: Dijkstra's mutual exclusion algorithm.

using the method of Owicki and Gries to reduce the required effort.

Proof

We have the following assertion network $\mathcal{Q}_i$ for P_i, $i = 1, 2$, where $j = 1, 2$, $i \neq j$.

$$\mathcal{Q}(s_i) \stackrel{\text{def}}{=} \neg cs_i \wedge \neg req_i \wedge (cs_j \to req_j) \wedge (\neg(cs_i \wedge cs_j))$$
$$\mathcal{Q}(l_i^1) \stackrel{\text{def}}{=} \neg cs_i \wedge req_i \wedge (cs_j \to req_j) \wedge (\neg(cs_i \wedge cs_j))$$
$$\mathcal{Q}(l_i^2) \stackrel{\text{def}}{=} \neg cs_i \wedge \neg req_i \wedge (cs_j \to req_j) \wedge (\neg(cs_i \wedge cs_j))$$
$$\mathcal{Q}(l_i^3) \stackrel{\text{def}}{=} cs_i \wedge req_i \wedge (cs_j \to req_j) \wedge (\neg(cs_i \wedge cs_j))$$
$$\mathcal{Q}(t_i) \stackrel{\text{def}}{=} false$$

Local correctness: We have to check the following local verification conditions of which only the fourth one is nontrivial.

- $\models \mathcal{Q}(s_i) \to \mathcal{Q}(l_i^1) \circ (req_i := tt))$.
- $\models \mathcal{Q}(l_i^1) \wedge req_j \to \mathcal{Q}(l_i^2) \circ (req_i := ff))$.
- $\models \mathcal{Q}(l_i^2) \to \mathcal{Q}(l_i^1) \circ (req_i := tt))$.
- $\models \mathcal{Q}(l_i^1) \wedge \neg req_j \to \mathcal{Q}(l_i^3) \circ (cs_i := tt))$.

  Here we have to particularly check that our invariant $\neg(cs_i \wedge cs_j)$ is maintained. Now $\mathcal{Q}(l_i^1) \wedge \neg req_j$ implies that $(cs_j \to req_j) \wedge \neg req_j$ and therefore $\neg cs_j$. Since cs_j is not changed by this transition, we have established that $\neg(cs_i \wedge cs_j)$ holds in l_i^3.
- $\models \mathcal{Q}(l_i^3) \to \mathcal{Q}(s_i) \circ (cs_i, req_i := ff, ff))$.
- $\models \mathcal{Q}(s_i) \wedge false \to \mathcal{Q}(t_i)$.

Interference freedom: We only consider the interesting case in which the transition from l_i^1 to l_i^3 is taken in P_i and prove that the assertion network associated with the locations of P_j remains interference free.

- $\models \mathcal{Q}(l_i^1) \wedge \mathcal{Q}(s_j) \wedge \neg req_j \to \mathcal{Q}(s_j) \circ (cs_i := tt))$.

  The only crucial clause is $\neg(cs_1 \wedge cs_2)$. Since $\mathcal{Q}(s_j)$ implies $\neg cs_j$ and cs_j is not changed by this particular transition, $\mathcal{Q}(s_j)$ holds afterwards.

- $\models \mathcal{Q}(l_i^1) \wedge \mathcal{Q}(l_j^1) \wedge \neg req_j \rightarrow \mathcal{Q}(l_j^1) \circ (cs_i := tt))$.

  Now since $\mathcal{Q}(l_j^1)$ implies req_j, we conclude from the premise that $req_j \wedge \neg req_j$ has to hold to enable the transition in this particular case. This turns out to be false, and hence the verification condition above is satisfied trivially.

- $\models \mathcal{Q}(l_i^1) \wedge \mathcal{Q}(l_j^2) \wedge \neg req_j \rightarrow \mathcal{Q}(l_j^2) \circ (cs_i := tt))$.

  Since $\mathcal{Q}(l_j^2)$ is the same as $\mathcal{Q}(s_j)$ this condition is already proven above.

- $\models \mathcal{Q}(l_i^1) \wedge \mathcal{Q}(l_j^3) \wedge \neg req_j \rightarrow \mathcal{Q}(l_j^3) \circ (cs_i := tt))$.

  Again, $\mathcal{Q}(l_j^3)$ implies req_j, and analogously to the case of $\mathcal{Q}(l_j^1)$ this verification condition is satisfied trivially.

- $\models \mathcal{Q}(l_i^1) \wedge \mathcal{Q}(t_j) \wedge \neg req_j \rightarrow \mathcal{Q}(t_j) \circ (cs_i := tt))$.

  Since $\mathcal{Q}(t_j) \equiv false$, this verification condition holds trivially.

We have to check 60 verification conditions for the interference freedom test and 12 local verification conditions – but due to the symmetry of the processes only half of them have to be actually carried out. This is an improvement over the direct product. Additionally, the verification conditions for the Owicki & Gries method are better structured and usually simpler than those for the product of the processes.

As indicated in Session 8, the above assertion network for P is inductive, and, by Lemma 4.2, it is also invariant. This means that for every computation $\eta : \langle l_0; \sigma_0 \rangle \longrightarrow \langle l_1; \sigma_1 \rangle \longrightarrow \langle l_2; \sigma_2 \rangle \longrightarrow \ldots$ we have $\models \mathcal{Q}_{l_i}(\sigma_i)$. The proof above indicates that $P_1 \| P_2$ satisfies the mutual exclusion property, since every assertion implies $\neg(cs_1 \wedge cs_2)$.

■

References

[Dij65] E.W. Dijkstra. Solution of a problem in concurrent programming control. *CACM*, 8(9), 1965.

[Dij68] E.W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages*, pages 43–112. Academic Press, New York, 1968.