Synthesizing a General Deadlock Predicate

By

David R. O'Hallaron
Paul F. Reynolds, Jr.

Computer Science Report # TR-85-01

March 22, 1985

Submitted to Information Processing Letters

Synthesizing a General Deadlock Predicate

DAVID R. O'HALLARON
PAUL F. REYNOLDS, JR.

University of Virginia, Charlottesville, Virginia

Satisfiability of the deadlock predicate constructed by the semaphore invariant method is a necessary condition for total deadlock in *PV* programs. Clarke has developed a technique, based on a view of resource invariants as fixpoints of a functional, for constructing a deadlock predicate such that satisfiability is a necessary and sufficient condition for total deadlock. We describe a technique for synthesizing a general deadlock predicate such that satisfiability is a necessary and sufficient condition for both total and partial deadlock. Our method constructs a strongest resource invariant using Clarke's fixed point functional. We then use this strongest resource invariant and an inverse fixed point functional to construct a general deadlock predicate.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming; D.2.4 [**Software Engineering**]: Program Verification; D.4.5 [**Operating Systems**]: Process Management — *deadlocks*

General Terms: Theory, Verification

Additional Keywords and Phrases: Semaphore invariant method, static deadlock detection, deadlock predicate, fixpoint

---

1. Introduction

The semaphore invariant method [Hab72, Hab75] uses a resource invariant called the semaphore invariant to construct the total deadlock predicate [OwG76], the satisfiability of which is a necessary condition for total deadlock in a *PV* program. The semaphore invariant method is an attractive method because the total deadlock predicate is compact and can be easily synthesized from the program text, as first described by E. M. Clarke in [Cla80], and later generalized by S. D. Carson in [Car84]. However, the semaphore invari-

---

Authors' present address: Department of Computer Science, University of Virginia, Charlottesville, Virginia, 22903.

ant method is incomplete for two reasons.

(1) Because of the possibility of feasible yet unreachable total deadlock states, satisfiability of the deadlock predicate is not a sufficient condition for total deadlock[Cla80].

(2) The total deadlock predicate cannot be used to detect partial deadlocks.

Clarke has addressed the first problem by developing an iterative procedure to synthesize from the program text a strongest resource invariant that is satisfied only by reachable states. With this strongest resource invariant, satisfiability of the total deadlock predicate becomes a necessary and sufficient condition for total deadlock.

We address the second problem by presenting a technique for synthesizing from the text of a program a deadlock predicate, the satisfiability of which is a necessary and sufficient condition for total and partial deadlock. We call this predicate the *general deadlock predicate*. Our method is a variant of Clarke's technique for generating strongest resource invariants.

Section Two defines basic concepts. Section Three describes the semaphore invariant method, and Section Four describes Clarke's technique for generating strongest resource invariants. Section Five describes our technique for constructing a general deadlock predicate from the program text. Section Six discusses some remaining challenges.

2. Program Model

We define a PV program as a collection of N cyclic processes

$$\textbf{var } L\textbf{; cobegin } P_1 \,//\, P_2 \,//\, \cdots \,//\, P_N \textbf{ end}$$

where L is a list of S semaphores and their initial nonnegative values. Each P_i is a process of the form

$$P_i : \textbf{cycle } S_i^1; S_i^2; \cdots ; S_i^{k_i} \textbf{ endcycle.}$$

Each S_i^k is a statement of the form

$$\textbf{when } \sigma > 0 \textbf{ do } \sigma \leftarrow \sigma - 1$$

denoted in the program text by $P(\sigma)$, or of the form

$$\textbf{when } \text{true} \textbf{ do } \sigma \leftarrow \sigma + 1$$

denoted in the program text by $V(\sigma)$.

As in [Cla80], we say that a *program state* τ is an ordered list $(pc_1, \ldots, pc_N; s) \in \Sigma$, where pc_i is the *program counter* for process i with $1 \leqslant pc_i \leqslant k_i$, and where s is the *program store* representing the values of the semaphores at state τ. The *initial state* has the form $(1, \ldots, 1; s_0)$ where s_0 is the store reflecting the initial values of the semaphores. We say that $\sigma(s)$ is the value of semaphore σ in store s, while $A(s)$ will be the new store resulting when statement A is executed in store s.

A *computation* is a sequence of states $\tau_j, \tau_{j+1}, \ldots, \tau_k, \cdots$ where consecutive states $\tau_k = (pc_1^k, \ldots, pc_N^k; s_k)$ and $\tau_{k+1} = (pc_1^{k+1}, \ldots, pc_N^{k+1}; s_{k+1})$ are related as follows: There exists an $m, 1 \leqslant m \leqslant N$, such that

(1) $$pc_i^{k+1} = \begin{cases} pc_i^k & \text{if } i \neq m \\ pc_i^k + 1 & \text{if } i = m \text{ and } pc_i^k < k_i \\ 1 & \textit{otherwise} \end{cases}$$

(2) if statement pc_m^k is $P(\sigma)$ then $\sigma(s_k) > 0$ and $s_{k+1} = A(s_k)$.

(3) if statement pc_m^k is $V(\sigma)$ then $s_{k+1} = A(s_k)$.

If τ_k and τ_{k+1} are states in a computation, then τ_k is an *immediate predecessor* of state τ_{k+1} and τ_{k+1} is an *immediate successor* of τ_k. A state τ is *reachable from* state τ' if and only if there exists a computation where τ follows τ' in the sequence. A state τ is *reachable* if and only if τ is in some computation starting from the initial state. Process i is *blocked* at statement pc_i^k in state $\tau_k = (pc_1^k, \ldots, pc_N^k; s_k)$ if and only if statement pc_i^k is a $P(\sigma)$ and $\sigma(s_k) = 0$. Process i is *deadlocked* at state τ if and only if process i is blocked at all states reachable from τ. We say that τ is a deadlock state if at least one process is

deadlocked at τ. If τ is a deadlock state and $k < N$ processes are deadlocked at τ, then τ is a *partial* deadlock state; otherwise τ is a *total* deadlock state.

$SP[A](U)$ denotes the *strongest postcondition* [Cla80] corresponding to the statement A and the *precondition* U. If we associate with predicate U the set of states that satisfy it, then $SP[A](U)$ is defined by

$$SP[A](U) = \{(pc_1,\ldots,pc_N;A(s)) \mid (pc_1,\ldots,pc_N;s) \in U\}.$$

We let $sp[A](V)$ denote the *strongest precondition* corresponding to the statement A and the *postcondition* V. If we associate with predicate V the set of states that satisfy it, then $sp[A](V)$ is defined by

$$sp[A](V) = \{(pc_1,\ldots,pc_N;s) \mid (pc_1,\ldots,pc_N;A(s)) \in V\}.$$

Notice that $sp[A](SP[A](U)) = U$.

3. Semaphore Invariant Method

The semaphore auxiliary method of static deadlock detection uses auxiliary variables, the semaphore invariant, statement preconditions, and blocking conditions to generate the total deadlock predicate. We briefly discuss each of these.

Let σ be a semaphore initialized to σ_0. Let σ_i^P and σ_i^V be auxiliary variables, initially zero, such that σ_i^P is incremented when process i acquires σ, and σ_i^V is incremented when process i releases σ. Then the semaphore invariant for σ is defined by

$$I_\sigma \equiv \sigma = \sigma_0 - \sum_{i=1}^{N}\sigma_i^P + \sum_{i=1}^{N}\sigma_i^V.$$

A state $(pc_1,\ldots,pc_N;s)$ is uniquely determined by the values of the auxiliary variables. If the semaphore invariant for any semaphore in a program is violated by the auxiliary variable values associated with a state, then the state is called *infeasible*. Notice that infeasible states are always unreachable while feasible states are not always reachable.

Associated with each process i is a set of auxiliary variables V_i, where $\sigma_i^P \in V_i$ if process i requests σ, and where $\sigma_i^V \in V_i$ if process i releases σ. The *precondition* of the *jth* statement in the *ith* process, $pre(S_i^j)$, describes the relationships that exist among the auxiliary variables in V_i just before S_i^j is executed.

Associated with each statement S_i^j is a *blocking condition*, b_i^j, that describes the conditions under which S_i^j is blocked. If S_i^j is $V(\sigma)$ then b_i^j is simply the predicate $(false)$. If S_i^j is $P(\sigma)$ then b_i^j is the predicate $(\sigma = 0)$.

Let S be the number of semaphores and let k_i be the number of statements in process i. Then the total deadlock predicate is defined by

$$D = \bigwedge_{i=1}^{N} \left\{ \bigvee_{j=1}^{k_i} pre(S_i^j) \wedge b_i^j \right\} \wedge \left\{ \bigwedge_{\zeta=1}^{S} I_\zeta \right\}$$

and is satisfied by feasible states where all processes are deadlocked. Satisfiability of the total deadlock predicate is a necessary condition for total deadlock. Unfortunately, satisfiability of the total deadlock predicate is not a sufficient condition for total deadlock because of the possibility of feasible yet unreachable states. As we shall see in the next section, Clarke solves this problem with an iterative technique that strengthens resource invariants such as the semaphore invariant so that all unreachable states are excluded.

4. Strengthening the Semaphore Invariant

Clarke's iterative technique for strengthening resource invariants such as the semaphore invariant is based on a view of a resource invariant as a *fixpoint* of a functional F. A similar technique that does not use resource invariants is presented in [Kel77]. The reader is directed to Clarke's paper for a more theoretical treatment.

The fixpoint functional $F : 2^\Sigma \rightarrow 2^\Sigma$ is defined by

$$F(J) = J_0 \vee J \vee \left\{ \bigvee_{i=1}^{N} \left\{ \bigvee_{k=1}^{k_i} SP[S_i^k](pre(S_i^k) \wedge \neg b_i^k \wedge J) \right\} \right\},$$

where the predicate J_0 describes the initial state of the program. Intuitively we think of $F(J)$ as the union of the set of states that satisfy J with the reachable immediate successors of the states that satisfy J.

The strongest resource invariant is defined by

$$F^*(\text{false}) = \bigcup_{i=0}^{\infty} F^i(\text{false}),$$

where $F^0(\text{false}) = \text{false}$ and $F^{i+1}(\text{false}) = F(F^i(\text{false}))$. Intuitively, we think of the strongest resource invariant as being satisfied by the set of all states reachable from the initial state.

Clarke has noted that the strongest resource invariant $F^*(J)$ converges in a finite number of steps when the program has a finite number of states or when J is a reasonably good approximation to $F^*(J)$. This implies that $F^*(\text{false})$ converges in a finite number of steps only when the program has a finite number of states. Clarke has addressed this problem in [Cla80]; we discuss it briefly in Section Six and propose a potentially useful alternative.

Given the strongest resource invariant $F^*(\text{false})$ Clarke constructs a strongest total deadlock predicate

$$D = \bigwedge_{i=1}^{N} \left\{ \bigvee_{j=1}^{k_i} pre(S_i^j) \wedge b_i^j \right\} \wedge F^*(\text{false})$$

which is satisfied by those reachable states where all processes are deadlocked. Its satisfiability is a necessary and sufficient condition for total deadlock.

5. Synthesizing a General Deadlock Predicate

Clarke's technique addresses one of the problems with the semaphore invariant method by strengthening the semaphore invariant so that satisfiability of the deadlock predicate is a necessary and sufficient condition for total deadlock. However, this total deadlock predicate cannot be used to detect partial deadlock. In this section, we present a technique for constructing a general deadlock predicate, the satisfiability of which is a necessary and sufficient condition for both total and partial deadlock.

We start with a weak predicate called the *blocking predicate.* Let P be the powerset of $\{1, \ldots, N\}$. Let P^i be the ith set in P. Let $A_k = \{P^i \mid k \in P^i\}$. Let A_k^i be the ith set in A_k, and let X_k^i be the complement of A_k^i. Then the blocking predicate for process α is

$$B_\alpha = \bigvee_{i=1}^{|A_\alpha|} \left[\bigwedge_{j \in A_\alpha^i} \left(\bigvee_{k=1}^{k_j} pre(S_j^k) \wedge b_j^k \right) \wedge \left(\bigwedge_{j \in X_\alpha^i} \left(\bigvee_{k=1}^{k_j} pre(S_j^k) \wedge \neg b_j^k \right) \right) \right] \wedge F^*(\text{false})$$

The predicate B_i is satisfied by the set of reachable states where process i is blocked and the predicate $U_i = \neg B_i \wedge F^*(\text{false})$ is satisfied by the set of reachable states where process i is not blocked. The blocking predicate for all processes

$$B = \bigvee_{i=1}^{N} B_i ,$$

is satisfied by the set of reachable states where at least one process is blocked. Since blocking is a necessary condition for deadlock, the satisfiability of B is a necessary condition for deadlock. For the satisfiability of B to be a sufficient condition for deadlock, we must strengthen it to include only those reachable states where at least one process is blocked forever. Our strengthening technique uses the inverse of Clarke's fixed point functional.

The inverse fixpoint functional $G : 2^\Sigma \rightarrow 2^\Sigma$ is defined by

$$G(J) = J \vee \left\{ \bigvee_{i=1}^{N} \left\{ \bigvee_{k=1}^{k_i} sp[S_i^k](post(S_i^k) \wedge J) \wedge F^*(\text{false}) \right\} \right\}$$

Intuitively we think of $G(J)$ as the union of the set of states that satisfy J with the set of reachable immediate predecessors of the states that satisfy J. Notice from the definition of G that

$$G(F(\text{false})) = \text{false}$$

and

$$G(F^{i+1}(\text{false})) = F^i(\text{false}),\ 1 \leqslant i \leqslant \infty.$$

Suppose we iteratively apply the inverse fixpoint functional G to the predicate $U_i = \neg B_i \wedge F^*(\text{false})$ to get

$$G^*(U_i) = \bigcup_{k=1}^{\infty} G^k(U_i).$$

If $F^*(\text{false})$ converges in a finite number of steps, then the program has a finite number of states, and $G^*(U_i)$ converges in a finite number of steps. For now, we assume a finite number of states. $G^*(U_i)$ is satisfied by the union of the set of all reachable states where process i is unblocked with the set of all reachable states where process i is blocked but eventually becomes unblocked. Thus $\neg G^*(U_i) \wedge B_i$ is satisfied by the set of reachable states where process i is blocked forever.

Given $G^*(U_i)$ for all processes, we can define Clarke's strongest total deadlock predicate by

$$D = \bigwedge_{i=1}^{N} \neg G^*(U_i) \wedge B_i,$$

which is satisfied by those reachable states where all processes are deadlocked. Its satisfiability is a necessary and sufficient condition for total deadlock. Furthermore, we can define the general deadlock predicate by

$$D^* = \bigvee_{i=1}^{N} \neg G^*(U_i) \wedge B_i,$$

which is satisfied by all reachable states where at least one process is deadlocked. Its satisfiability is a necessary and sufficient condition for both partial and total deadlock.

6. Further Challenges

To simplify our presentation of the general deadlock predicate, we have limited our discussion to cyclic *PV* programs with a finite number of states. However, there exist useful cyclic *PV* programs with an infinite number of states. For these programs the strongest resource invariant $F^*(\text{false})$ cannot be obtained through a finite number of applications of the fixed point functional. Clarke has developed a more powerful iterative technique for obtaining strongest resource invariants based on the notion of *widening*. This technique is guaranteed to converge in a finite number of steps when the strongest resource invariant is the conjunction of a finite number of inequalities. This condition on convergence raises a number of questions.

First, we would like to know if Clarke's more powerful iterative technique will converge for all cyclic *PV* programs. If not, can we characterize those programs for which it *will* converge?

Second, Carson [Car84] has pointed out that the number of unique deadlocks in a cyclic *PV* program is finite. Thus one need only analyze a finite subset of the potentially infinite states in a program to accurately predict if the program can deadlock. Determining the size of this finite subset of states is called the *finite modeling problem.* Solutions to the finite modeling problem have been found for restricted classes of *PV* programs in [Car84] and [OHR85]. Furthermore, Carson has shown that the problem is decidable for total deadlocks in general *PV* programs.

That the number of unique deadlocks is finite suggests that it is unnecessary to obtain the strongest resource invariant in order to perform accurate deadlock detection

with the general deadlock predicate. Rather we need only iteratively apply the fixed point functional until we obtain an approximate strongest resource invariant $\hat{F}^*(\text{false})$ that admits a finite set of states large enough to include all unique deadlocks. Thus, if we can solve the finite modeling problem, then we also solve the problem of generating the general deadlock predicate for arbitrary *PV* programs.

7. Conclusions

We have presented a technique for synthesizing a general deadlock predicate from the text of a cyclic *PV* program such that the satisfiability of the predicate is a necessary and sufficient condition for both total and partial deadlock. Our method is based on Clarke's method for generating strongest resource invariants.

Our method constructs a strongest resource invariant, $F^*(\text{false})$ using Clarke's fixed point functional. We then use $F^*(\text{false})$ and an inverse fixed point functional to strengthen a blocking predicate such that satisfiability of the strengthened predicate is a necessary and sufficient condition for both total and partial deadlock.

References

[Car84] S. D. Carson, Geometric Models of Concurrent Programs, PhD Dissertation, University of Virginia, 1984.

[Cla80] E. M. Clarke, Synthesis of Resource Invariants for Concurrent Programs, *ACM Transactions on Programming Languages and Systems 2*, 3 (July 1980), 338–358.

[Hab72] A. N. Habermann, Synchronization of Communicating Processes, *Communications of the ACM 15*, 3 (March 1972), 171–176.

[Hab75] A. N. Habermann, Path Expressions, Technical Report, Department of Computer Science, Carnegie–Mellon University, June 1975.

[Kel77] R. M. Keller, Generalized Petri Nets as Models for System Verification, Technical Report, University of Utah, 1977.

[OHR85] D. R. O'Hallaron and P. F. Reynolds, Jr., Finite Models of Cyclic Concurrent Programs, Technical Report 85–01, Department of Computer Science, University of Virginia, 1985.

[OwG76] S. Owicki and D. Gries, Verifying Properties of Parallel Programs: An Axiomatic Approach, *Communications of the ACM 19*, 5 (May 1976), 279–284.