GI 4TH ANNUAL MEETING OCTOBER 9-12, 1974 | BERLIN

DEADLOCK CONDITIONS IN WELL STRUCTURED MODULAR SYSTEMS

P.Ancilotti, M.Fusani, N.Lijtmaer, C.Thanos
Istituto di Elaborazione dell'Informazione, C.N.R.
Pisa, ITALY

1. Introduction

Communication between program modules is a topic of great interest in the design and production of well structured modular systems. *Modules* may be thought of as logical components of the system,designed to carry out some tasks. To complete the description of a system we need to specify the *connections* between modules,"that are the assumptions which the modules make about each other" [1]. Systems in which the connections between modules contain little information are labeled *well structured* and in fact this property is essential to facilitate the changeability of the system and to prove system correctness [2].

This paper is concerned with some properties of well structured modular systems. In these systems each module is specified in terms of input/output behaviour and input/output interfaces are standardized. A single module becomes,during execution,a *sequential* process,while the whole system allows the *concurrent* execution of several processes which are themselves strictly sequential.

Following the input/output approach,synchronization is achieved by using the *message buffer mechanism* and the only communication primitives are *send* and *receive*. The main goal of this paper is to investigate the properties of such mechanism with respect to deadlock conditions.

Schemata are used as computation models: Cyclic sequential schemata are introduced to represent modules,while the model of the whole software system is obtained by a directed parallel composition of cyclic sequential schemata.

As far as deadlock is concerned, three different kinds of systems will be considered:

i) Data and time independent systems.

ii) Data dependent and time independent systems.

iii) Data and time dependent systems.

We shall show that in the first two types of systems reproducibility of deadlock conditions exists and is related to the connections between modules. Thus,any effort to provide an algorithm to avoid deadlock is a nonsense for systems having statically connected modules. On the basis of these results, an algorithm has been implemented as a procedure of the PSL dynamic nucleus (Pisa Software Laboratory). The PSL [3], is specially convenient to test the applicability of this algorithm since it generates an environment where a user may build and experiment well structured modular software systems.

2. Computation Schemata

" A *computation schema*, or *schema*,represents the manner in which functional elements and decision elements are interconnected, and their actions sequenced, to define an algorithm"[4].

More precisely a schema is a triple $\Sigma=(A,V,C)$, where A is a set of *actors*, V is a set of *variables*,and C is a set of *control sequences* (*Control Set*[5]). The functional elements of a schema are called *operators* and the decision elements are called *deciders*. The set of operators and the set of deciders are denoted by O and $\mathcal{D}$,respectively. These sets are disjoint ($\mathcal{D} \cap O = \Phi$). Both, operators and deciders, are called *actors*. Then, actors are agents capable of transforming values and agents capable of testing values. To each actor a a *domain* X_a is associated, a finite subset of V. Similarly, to each operator o a *range* Y_o is associated, a finite subset of V.

A subset of variables ($I \subseteq V$) is called *schema input* if values are assigned to them before a computation begins.

Actors, operators and deciders,will be considered the units of computational activity, as characterized by their external behaviour. Associated with an operator o there are an *initiation event* $\bar{o}$ and a *termination event* $\underline{o}$. Associated with each decider d, there are an initiation event $\bar{d}$ and either the true or the false termination event,denoted by d_T and d_F respectively.

A *control sequence* of the schema is a string $\sigma = \alpha_1\alpha_2 \ldots \alpha_n \ldots$ of actor initiation and termination events. The *control set* C represents all the allowed sequences of events. The sequences in which operators and deciders are permitted to act, may be specified by a precedence graph [4].

Definition 2.1: Given a control sequence $\sigma = \alpha_1\alpha_2 \ldots \alpha_{i-1}\alpha_i \ldots$, if $\pi_{i-1} = \alpha_1 \ldots \alpha_{i-1}$ is a prefix of σ, after the occurrence of the events in π_{i-1}, the only *feasible event* in σ is α_i, $i \geq 1$.

Note that if $\alpha_i = \underline{a}$ belongs to a control sequence, then $\vec{a}$ must be present in the prefix π_{i-1}.

Definition 2.2: Given a control sequence σ of a schema Σ, a *chain of events* in σ is a string γ of event occurrences in σ : $\gamma = \alpha_i\alpha_j\alpha_h \ldots$ such that $i < j < h < \ldots$.

We will refer to a schema Σ as a *data independent schema* if the set of deciders is empty, otherwise Σ will be called a *data dependent schema*.

A more detailed treatment about schemata is contained in [4,5]. In this paper we will be concerned with the control aspects of computation schemata. In particular, we want to emphasize that our results are concerned with *control* and *sequencing* of a set of cooperating processes, and not with their specific functions. For this purpose we refer to schemata as models of uninterpreted program modules. In this context any control sequence of a schema represents a particular process the program module may give rise to, during execution.

To convert a schema into a specification of a particular program module it is necessary to specify the value of the variables of the schema and a function or predicate for each operator or decider.

Definition 2.4: An *interpretation* of a schema Σ is

i) for each variable $v \in V$, a value set $F(v)$;

ii) for each operator $o \in \mathcal{O}$, a function $f_o: F(v_{x1}) \times F(v_{x2}) \times \ldots \times F(v_{xm}) \rightarrow F(v_{y1}) \times F(v_{y2}) \times \ldots \times F(v_{yn})$, where $X_o = \{v_{x1}, \ldots, v_{xm}\}$ and $Y_o = \{v_{y1}, \ldots, v_{yn}\}$;

iii) for each decider $d \in \mathcal{D}$ a predicate $P_d: F(v_{x1}) \times F(v_{x2}) \times \ldots \times F(v_{xm}) \rightarrow \{\text{true, false}\}$, where $X_d = \{v_{x1}, \ldots, v_{xm}\}$

Our attention will be focalized now on sequential processes. Par-

allel processes will be analized later as a combination of several processes which are themselves strictly sequential.

3. Sequential Schemata

Definition 3.1: A *sequential schema* is a schema $\Sigma=(A,V,C)$ where control sequences of C satisfy the following properties:

i) For each $\sigma \in C$, after the ocurrence of $\bar{a}$, the feasible event is $\underline{a}$;

ii) If an event $\underline{a}$ belongs to many control sequences of C, the feasible event after $\underline{a}$ is the same in all sequences;

iii) If an event $\bar{a}$ belongs to many control sequences of C, the feasible event before $\bar{a}$ is the same in all sequences.

In the precedence graph of a sequential schema, each operator has at most one immediate successor and one immediate predecessor.

Note that if a sequential schema is data independent then there is only one possible control sequence σ in C.

Definition 3.2: A *finite sequential schema* is a sequential schema in which the set of actors is finite.

While the control sequences of this type of schemata allow to model the behaviour of sequential processes, a new type of schema must be introduced to handle cyclic sequential processes.

Definition 3.3: Given a finite sequential schema $\Sigma = (A,V,C)$, let us define a *cyclic sequential schema* $\Sigma^C =(A^C,V^C,C^C)$, where $A^C= A$, $V^C= V$ and $\sigma^C \in C^C$ iff $\sigma^C= \sigma_1 \sigma_2 \ldots \sigma_i \ldots$ with $\sigma_i \in C$ for any integer i. When a σ_i is composed by infinite events, then $\sigma^C= \sigma_1 \ldots \sigma_i$

If a cyclic sequential schema is data independent then there is only one possible control sequence σ^C in C^C: $\sigma^C= \sigma\sigma \ldots \sigma \ldots$ where σ is the unique control sequence of Σ. Note that while variables of schemata may be denoted, generally, by single memory cells, each input variable of a cyclic sequential schema is an array of infinite memory cells. Consecutive readings of an input variable involve consecutive array cells.

4. Parallel Composition of Cyclic Sequential Schemata

In a parallel schema a mechanism capable of describing concurrent, asynchronous activity is needed: This mechanism consists of allowing several initiations before a termination occurs, and of keeping track of

such initiations.

Our attention will be focalized now on schemata whose control sequences model sets of concurrent processes. They can be conceived as a combination of several processes which are themselves strictly sequential.

In this section we introduce the *directed parallel composition of cyclic sequential schemata* to model the behaviour, and to point out special properties, of well structured software systems. In particular we will consider systems composed of several independent modules connected by a message buffer mechanism and where a single module becomes, during execution, a cyclic sequential process.

In order to point out some properties related to concurrency, let us introduce:

i) A special type of variable, called *mailbox*;

ii) Two new types of actors, *send* and *receive*.

Mailbox is a pair (c,N) where c is a memory cell that can assume only integer values between zero and n, and where N is an array of n memory cells. Note that n may be infinite. Here n represents the capacity of the mailbox. The pair of integers (w,n), where w is the actual value of c, is referred to as the *state* of the mailbox.

The actors *send* and *receive* may be defined as follows:

i) The domain of any actor *send* (*receive*) is a set $X_s = \{x,m\}$ ($X_r=\{m\}$) where the variables x and m are a memory cell and a mailbox respectively. The range of any actor *send* (*receive*) is a set $Y_s=\{m\}$ ($Y_r=\{x,m\}$) where the mailbox m is the same in both the domain and the range.

ii) Associated with each actor *send* (*receive*) there are an initiation event $\bar{s}$ ($\bar{r}$) and a termination event $\underline{s}$ ($\underline{r}$).

iii) Given a control sequence $\sigma=\alpha_1..\alpha_i..$ where $\alpha_i=\underline{s}$ ($\underline{r}$), α_i is *feasible* after the occurrence of all the events $\alpha_1,..\alpha_{i-1}$ if the state of the mailbox m is such that $w<n$ ($w>0$). When $\underline{s}$ ($\underline{r}$) occurs the state of m is modified as follows: w:=w+1 (w:=w-1).

Note that *send* and *receive* are the only actors that are allowed to modify the state of a mailbox.

The *state* of a schema is the set of states of all the mailboxes of

the schema. Given an initial state and a control sequence we obtain a sequence of states of the schema. From now on the initial state is assumed to be such that for each mailbox w = 0.

Definition 4.1: A *Directed Parallel Composition* (DPC) of n cyclic sequential schemata $\Sigma_1,\ldots,\Sigma_n$ is a schema $\Sigma^P = (A^P, V^P, C^P)$ where:

i) $A^P = \bigcup_{i=1}^{n} A_i$ and for any pair (Σ_i, Σ_j) of component schemata $A_i \cap A_j = \Phi$

ii) $V^P = \bigcup_{i=1}^{n} V_i$ and for any *ordered* pair (Σ_i, Σ_j) of component schemata $V_i \cap V_j = M_{i,j}$

iii) Any $\sigma^P \in C^P$ can be genetated as follows: Given an n-tuple $\sigma_1,\ldots,\sigma_n$ where $\sigma_i \in C_i$ $(i=1,\ldots,n)$, for each integer $h \geq 0$ if $\pi_h = \alpha_1 \ldots \alpha_h$ is a prefix of σ^P, $\pi_{h+1} = \pi_h \alpha_{h+1}$ is a prefix of σ^P if α_{h+1} is the feasible event of a σ_i after the ocurrence of the events of σ_i in π_h. Moreover, we assume that all the events that become feasible, must occur in σ^P at most after a finite number of event occurrences.

$M_{i,j}$ is a set, possibly empty, whose elements are mailboxes belonging to the range of actors *send* of the schema Σ_i and to the domain of actors *receive* of Σ_j. Mailbox $m_{i,j} \in M_{i,j}$ is said to *link* Σ_i with Σ_j. If $M_{i,j} = \Phi$ then Σ_i and Σ_j are not directly connected.

A DPC Σ^P can be represented by a *Connection Graph* G whose nodes are the component schemata and whose arcs are links among schemata, that is mailboxes.

We introduce now, some definitions on the connection graph and on deadlock.

Definition 4.2: A *terminal schema* of a DPC Σ^P is a schema Σ_i for which only one mailbox links Σ_i to some other schemata of Σ^P.

Definition 4.3: A *path* in the connection graph G of a DPC Σ^P is a collection of distinct schemata $\Sigma_1,\ldots,\Sigma_k$ linked through k-1 mailboxes: $m_{1,2}, m_{2,3}, \ldots, m_{k-1,k}$.

Definition 4.4: A *semipath* in G is a collection of distinct schemata $\Sigma_1,\ldots,\Sigma_n$ linked through k-1 mailboxes: $m_{1,2}$ *or* $m_{2,1}$, $m_{2,3}$ *or* $m_{3,2}$, $\ldots$, $m_{k-1,k}$ *or* $m_{k,k-1}$. If Σ_k coincides with Σ_i the semipath is called a *semicycle*.

Definition 4.5: A mailbox is called *one-to-one* if it links only two

schemata.

Definition 4.6: Given a control sequence σ^p of a DPC Σ^p, a *deadlock* occurs in σ^p if at least one component schema Σ_k and an integer h exist such that for each integer t>h, $\sigma_t \notin \sigma_k$. Here σ_k is the control sequence of Σ_k in σ^p and σ_k is said a *deadlocked control sequence*. The first event of σ_k that follows the last event of σ_k in σ^p is called the *blocking event* of σ_k.

Note that the blocking events are unfeasible terminations of actors *send* or *receive* related to a mailbox whose state is (n,n) or (0,n) respectively.

Definition 4.7: For any blocking event e_i belonging to a deadlocked control sequence σ_i, if there is an event e_j belonging to a control sequence σ_j such that the occurrence of e_j in σ^p makes feasible e_i, then e_j is called an *awaking event* of e_i.

Obviously if σ_i is deadlocked in σ^p then also σ_j is deadlocked.

Definition 4.8: A deadlock in a control sequence σ^p of a DPC Σ^p is called a *partial deadlock* if σ^p is infinite. Otherwise it is called a *total deadlock*.

Definition 4.9: A DPC Σ^p is *well formed* if:

i) Any mailbox links at least two schemata.

ii) The connection graph G is connected.

From now on, any DPC will be tacitly considered well formed.

In the following sections distinctions among different kinds of DPC will be pointed out in order to find relevant properties of systems under deadlock conditions.

5. Data and Time Independent Systems

Definitions 5.1: A DPC Σ^p of n cyclic sequential schemata $\Sigma_1 \ldots \Sigma_n$ is of *type 1* if all the component schemata are *data independent* and all the mailboxes are one-to-one.

Any DPC Σ^p of type 1 is such that all its component schemata have only one control sequence. Furthermore, since Σ^p is well formed, it can be proved that if a deadlock occurs in some σ^p, for any blocking event e_i belonging to a deadlocked control sequence σ_i *there exists* one component schema Σ_j whose control sequence σ_j contains the awaking event

of e_i. Thus, also σ_j is deadlocked in σ^p. In other words a *unique* deadlocked control sequence *can not exist* in σ^p.

More generally, the following lemma can be proved:

Lemma 5.1: Given a control sequence σ^p of a DPC Σ^p of type 1, if a deadlock occurs in σ^p, then for any Σ_i whose control sequence σ_i is deadlocked, every schema Σ_j such that a path exists in G from Σ_i to Σ_j, has the control sequence deadlocked.

In particular, if all the mailboxes of Σ^p have a finite capacity, then the following theorem can be proved:

Theorem 5.1: Given a control sequence σ^p of a DPC Σ^p of type 1, where all the mailboxes of Σ^p have a finite capacity, if a deadlock occurs in σ^p, then it is a total deadlock.

Now we give a necessary condition for the occurrence of a deadlock, both a partial and a total deadlock, in a control sequence σ^p of a DPC of type 1. This condition is related to the connection topology of the component schemata. For a proof of this and the following theorems, see [6].

Theorem 5.2: Given a control sequence σ^p of a DPC Σ^p of type 1, the existence of a semicycle in the connection graph G of Σ^p is a necessary condition for a deadlock in σ^p.

Let us state now, an important property of any DPC of type 1:

Theorem 5.3: Given a DPC of type 1, if a particular control sequence σ^p_x *is* finite and is composed of k events, then every control sequence σ^p is finite and is composed of the same number k of events.

The theorem 5.3 asserts that for a DPC of type 1, a total deadlock is independent of the particular control sequence. It only depends on both, the topology of the connections among component schemata and the structure of each component schema. Modular systems modelled by DPC of type 1 give rise, during execution, to a family of cooperating asynchronous processes. For these type of modular systems, a deadlock condition depends neither on the input data nor on the relative speeds of the component processes. For this reason we call this type of systems: *Data and time independent systems.*

6. Data Dependent and Time Independent Systems

Definition 6.1: A DPC Σ^p of n cyclic sequential schemata is of *type 2*

if all the mailboxes in Σ^p are one-to-one.

Any DPC of type 1 is a particular case of DPCs of type 2, where all the component schemata are data independent. When deciders are present, it is no longer true that for any cyclic sequential schema there is only one possible control sequence. Therefore, for a DPC of type 2, if a deadlock occurs in some control sequence σ^p, it is not true that for a blocking event e_i of a deadlocked control sequence σ_i there exists a Σ_j whose control sequence σ_j contains the awaking event of e_i. In particular, a unique deadlocked σ_i can exist in σ^p. Thus, lemma 5.1 and theorem 5.1 are no longer true for a DPC of type 2. For the same reason also theorem 5.2 is false, in general. However we can prove that it is still true only for total deadlocks.

Theorem 6.1: Given a control sequence σ^p of a DPC Σ^p of type 2, a semicycle in the connection graph G of Σ^p is a necessary condition for a total deadlock in σ^p.

Theorem 5.3 is a consequence of the characteristics of a DPC of type 1, that is all the mailboxes are one-to-one and for each component schema there is only one control sequence. Here, theorem 5.3 is no longer true. However, an analogue theorem can be proved, if we choose a particular interpretation of actors *send* and *receive*, namely, values are written by *send* in the cells of the mailbox vector N, one at a time, and they are read by *receive* in exactly the same order. In other words the vector N is a FIFO queue. For any DPC of type 2, let us denote by Ω the set of interpretations for which, actors *send* and *receive* have the above specification.

We can prove that for any DPC of type 2, given an interpretation $\omega \in \Omega$ and an input assignement, there is only one possible control sequence for any component schema. Then, the following theorem can be proved

Theorem 6.2: Given a DPC of type 2 if, for some interpretation $\omega \in \Omega$ and for some input assignement, a particular control sequence σ^p_κ is finite and composed by k events, then, for the same interpretation and for the same input assignement, every control sequence σ^p is finite and composed by the same number k of events.

This theorem enables us to say that in any modular system modelled by a DPC of type 2 a total deadlock is independent of the relative speeds of the component processes. However, a total deadlock depends on

the input data. In other words if, for certain input data, a total deadlock occurs during a computation, then, for the same input data, the same deadlock condition occurs in any computation. For this reason we call this type of systems: *Data Dependent and Time Independent Systems*.

We want to emphasize that the above results hold only for an interpretation $\omega \in \Omega$. However, considering a mailbox as a FIFO queue is not a strong limitation.

A *DPC* of type 2 is a particular case of a *DPC* were all the mailboxes are one-to-one. If a *DPC* is not of type 2, theorem 6.2 is no longer true. In other words, for a modular system modelled by a general type of *DPC* a deadlock depends on the input data *and* on the relative speeds of the component processes. We call this type of systems *Data and Time dependent*.

7. Conclusions

This work arised from the design of the PSL [3], a software laboratory which aids an user to build well structured software systems. The above results can be tested on the PSL. In particular an algorithm for deadlock detection has been implemented [6].

Systems modelled by *DPCs* of type 2 are particularly interesting. In fact, they are sufficiently general. Furthermore, for any interpretation $\omega \in \Omega$, they belong to the class of Patil's β systems [7]. That is, they are functinal systems.

8. References

[1] D.L.Parnas:" Information Distribution Aspects of Design Methodology" *IFIP, Proceedings - Ljubljana, Yugoslavia 1971.*

[2] D.L.Parnas:" On the problem of producing well structured programs". *Computer Science Research Review 1971-72. Carnegie Mellon University.*

[3] P.Ancilotti, R.Cavina, M.Fusani, F.Gramaglia, N.Lijtmaer, E.Martinelli, C.Thanos: " Designing a Software Laboratory" *Proceedings 8th Yugoslav International Symposium on Information Processing, Bled, October 1973.*

[4] J.B.Dennis: *Course Notes "Computation Structures"* Department of Electrical Engineering, MIT 1967.

[5] J.P.Linderman:*"Productivity in Parallel Computation Schemata"*,MIT Project MAC - TR - 111, December 1973.

[6] P.Ancilotti, M.Fusani, N.Lijtmaer: *"Interprocess Communication: Deadlock Conditions"* IEI-CNR Internal Report (to be printed).

[7] S.S.Patil: "Closure properties of interconnections of determinate systems" *Record - Project MAC Conference on Concurrent Systems and Parallel Computation. ACM ,N.Y.1970.*