Tasking Deadlocks in Programs with the Full Ada 95

Yasushi Tojo Sinsuke Nara Yuichi Goto Jingde Cheng

Department of Information and Computer Sciences
Saitama University
Saitama 338-8570, Japan
{tojo, nara, gotoh, cheng}@aise.ics.saitama-u.ac.jp

Abstract

This article presents some examples of tasking deadlocks concerning tasking synchronization waiting relations defined in Ada 95's Real-Time Systems Annex.

1 Introduction

A tasking deadlock in a concurrent Ada program is a situation where some tasks form a circular waiting relation, which cannot be resolved by the program itself, at some synchronization and/or communication points and hence can never proceed with their computation.

We have proposed a method for detecting tasking deadlocks at run-time in Ada 95 programs [5] and developed a run-time tasking deadlock detector based on the method [8]. However, the detecting method did not take Ada 95's annexes into account. Ada 95's Real-Time Systems Annex provides a way for synchronization of tasks by suspension objects. Since the synchronization may involve waiting relations among tasks, a program using the annex may have some tasking deadlocks that are not reported until now.

In this article, we present some examples of tasking deadlocks concerning tasking synchronization waiting relations defined in Ada 95's Real-Time Systems Annex.

2 Tasking Synchronization Waiting Relations in Programs with the Full Ada 95

The core Ada 95 defines 7 types of tasking synchronization waiting relations, i.e., activation waiting relation, finalization waiting relation, completion waiting relation, acceptance waiting relation, entry-calling waiting relation, protection waiting relation, and protected-entry-calling waiting relation [1, 5, 7].

Ada 95's annexes define additional specialized facilities for 14 distinct application areas, but there is no synchronization waiting relation except the suspension waiting relation defined in Real-Times Systems Annex, i.e., Annex D [7]. Annex D.10 Synchronous Task Control defines a language-defined private semaphore (suspension object), which can be used for two-stage suspend operations and as a simple building block for implementing higher-level queues. Using a suspension object, it is possible to achieve mutual exclusion between two tasks by simply encapsulating their critical sections. Suspension objects can also be used inside protected objects to implement condition synchronization [2].

Suspension waiting may result in tasking deadlocks in a program with the full Ada 95. A task that issues Suspend_Until_True operation on a suspension object is blocked until the value of the suspension object becomes true. Some tasks which can make value of suspension objects true can resolve suspension waiting. If no task has the possibility to resolve suspension waiting, a task in suspension waiting state will be blocked forever and may be involved a tasking deadlock.

3 Examples of Tasking Deadlocks in Programs with the Full Ada 95

Five example programs are presented in this section. We present each program by its Ada 95 code and its Task-Wait-For Graph. Three programs have tasking deadlocks with suspension waiting. A program has a temporary cycle of synchronization waiting but it does not lead to a tasking deadlock. Another program has a tasking deadlock and a tasking livelock simultaneously.

3.1 Task-Wait-For Graph

A Task-Wait-For Graph (TWFG for short) is a kind of arc-classified digraph to represent tasking waiting state in an execution of an Ada program [4]. The TWFG explicitly represents various types of waiting relations in an execution of an Ada program. The notion of TWFG was originally proposed for classification and detection of tasking deadlocks in Ada 83 programs [3, 4] and was extended to deal with tasking deadlocks in Ada 95 programs [5]. In a TWFG, vertices indicate tasking objects. A tasking object in an execution state of a concurrent Ada 95 program is any of the fol-

lowing: a task whose activation has been initiated and whose state is not terminated, a block statement that is being executed by a task, a subprogram that is being called by a task, a protected subprogram that is being called by a task, a protected object on which a protected action is undergoing, and a suspension object that is being waited by a task. Arcs indicate synchronization waiting relations which are binary relations between tasking objects. In a TWFG of a programs with the full Ada 95, they are 8 types of arcs: activation waiting arc, finalization waiting arc, completion waiting arc, acceptance waiting arc, entry-calling waiting arc, protection waiting arc, protected-entry-calling waiting arc, and suspension waiting arc, respectively, corresponding to an activation waiting relation, finalization waiting relation, completion waiting relation, acceptance waiting relation, entry-calling waiting relation, protection waiting relation, protected-entry-calling waiting relation, and suspension waiting relation.

3.2 Example Program 1: Tasking Deadlocks with all Types of Synchronization Waiting Relations

This program has 8 types of synchronization waiting relations. These synchronization waiting relations form three cycles that cannot be resolved by the program itself. Therefore, threee tasking deadlocks must occur in this program. The three synchronization waiting cycles in this program are as follows:

$$T_3 \rightarrow_{PEC} V \rightarrow_{Fin} GET \rightarrow_{EC} T_2 \rightarrow_{Com} T_3$$

$$T_1 \rightarrow_{Acc} T_4 \rightarrow_{Fin} B \rightarrow_{Fin} T_7 \rightarrow_{Fin} W \rightarrow_{Pro} V \rightarrow_{Fin} GET \rightarrow_{EC} T_2 \rightarrow_{Com} T_1$$

$$T_1 \rightarrow_{Acc} T_4 \rightarrow_{Fin} B \rightarrow_{Fin} T_6 \rightarrow_{EC} T_5 \rightarrow_{Act} T_8 \rightarrow_{Sus} S \rightarrow_{Sus} T_2 \rightarrow_{Com} T_1$$

Here the affixing characters show types of arcs. $\rightarrow_{Act}$, $\rightarrow_{Fin}$, $\rightarrow_{Com}$, $\rightarrow_{Acc}$, $\rightarrow_{EC}$, $\rightarrow_{Pro}$, $\rightarrow_{PEC}$, and $\rightarrow_{Sus}$ denote activation waiting arc, finalization waiting arc, completion waiting arc, acceptance waiting arc, entry-calling waiting arc, protection waiting arc, protected-entry-calling waiting arc, and suspension waiting arc, respectively.

Example Program 1

```
with Ada.Synchronous_Task_Control;
use  Ada.Synchronous_Task_Control;
procedure Main is
   type ITEM is new Integer;
   task T1 is entry E1; end T1;
   task T2 is entry E2; end T2;
   task T3;
   S : Suspension_Object;
   function GET return ITEM is
   begin
      T2.E2; return 0;
   end GET;
   protected V is
      Procedure W(X: in ITEM);
      entry R(X: out ITEM);
      private Var: ITEM := 0;
   end V;
   protected body V is
      procedure W(X: in ITEM) is
      begin Var := X; end W;
      entry R(X: out ITEM) when TRUE is
      begin X := GET; end R;
   end V;
   task body T1 is
      task T4;
      task T5 is entry E5; end T5;
      task body T4 is
      begin
         B:
         declare
            task T6;
            task T7;
            task body T6 is
            begin T5.E5; end T6;
            task body T7 is
               Y: ITEM;
            begin V.W(Y); end T7;
         begin null; end B;
         T1.E1;
      end T4;
      task body T5 is
         task T8;
         task body T8 is
         begin
            Set_False(S);
            Suspend_Until_True(S);
         end T8;
      begin accept E5; end T5;
   begin accept E1; end T1;
   task body T2 is
   begin
      select
         when FALSE => accept E2;
      or
         terminate;
      end select;
      Set_True(S);
   end T2;
   task body T3 is
      Z: ITEM;
   begin V.R(Z); T1.E1; end T3;
begin null; end Main;
```

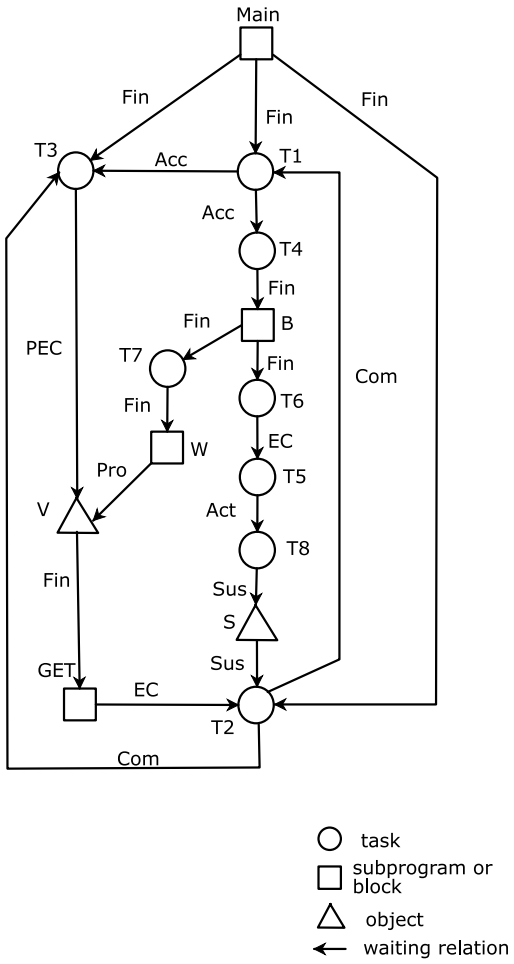

Fig. 1: TWFG of example program 1

3.3 Example Program 2: A Tasking Deadlock with Only Suspension Waiting

In this program some suspension waiting relations form a cycle and these is no task that can resolve the suspension waiting relations. Therefore a tasking deadlock occurs in this program. The synchronization waiting cycle in this program is as follows:

$$T_1 \rightarrow_{Sus} S_1 \rightarrow_{Sus} T_3 \rightarrow_{Sus} S_3 \rightarrow_{Sus} T_2 \rightarrow_{Sus} S_2 \rightarrow_{Sus} T_1$$

Example Program 2

```
with Ada.Synchronous_Task_Control;
use  Ada.Synchronous_Task_Control;
procedure Main is
   task T1;
   task T2;
   task T3;
   S1 : Suspension_Object;
   S2 : Suspension_Object;
   S3 : Suspension_Object;
   task body T1 is
   begin
      Set_False(S1);
      Suspend_Until_True(S1);
      Set_True(S2);
   end T1;
   task body T2 is
   begin
      Set_False(S2);
      Suspend_Until_True(S2);
      Set_True(S3);
   end T2;
   task body T3 is
   begin
      Set_False(S3);
      Suspend_Until_True(S3);
      Set_True(S1);
   end T3;
begin null; end Main;
```

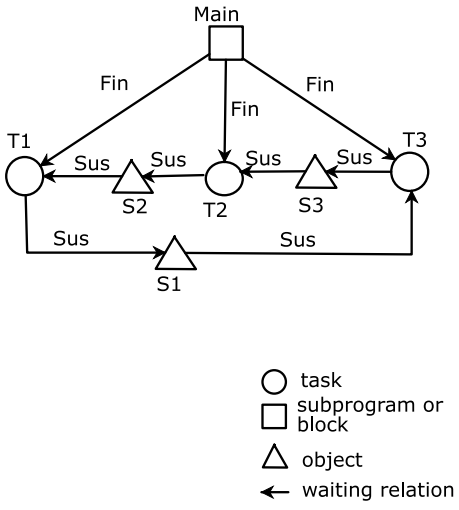

Fig. 2: TWFG of example program 2

3.4 Example Program 3: A Cycle of Synchronization Waiting that is not a Tasking Deadlock

This program shall form a cycle of synchronization waiting relations. However the cycle is not a tasking deadlock, because suspension waiting is resolved by a task that is not in the cycle. The synchronization waiting cycle in this program is as follows:

$$T_1 \rightarrow_{Act} T_3 \rightarrow_{Fin} A \rightarrow_{Fin} T_4 \rightarrow_{Sus} S_1 \rightarrow_{Sus} T_2 \rightarrow_{Com} T_1$$

Example Program 3

```
with Ada.Synchronous_Task_Control;
use  Ada.Synchronous_Task_Control;
procedure Main is
   task T1;
   task T2 is entry E2; end T2;
   S1 : Suspension_Object;
   task body T1 is
      task T3;
      task body T3 is
      begin
         A:
         declare
            task T4 is entry E4; end T4;
            task T5;
            procedure GET is
            begin
               T4.E4;
            end GET;
            task body T4 is
            begin
               Set_False(S1);
               Suspend_Until_True(S1);
               accept E4;
            end T4;

            task body T5 is
            begin
               Set_True(S1);
               GET;
            end T5;
         begin null; end A;
      end T3;
   begin null; end T1;
   task body T2 is
   begin
      select
         when FALSE => accept E2;
      or
         terminate;
      end select;
      Set_True(S1);
   end T2;
begin null; end Main;
```

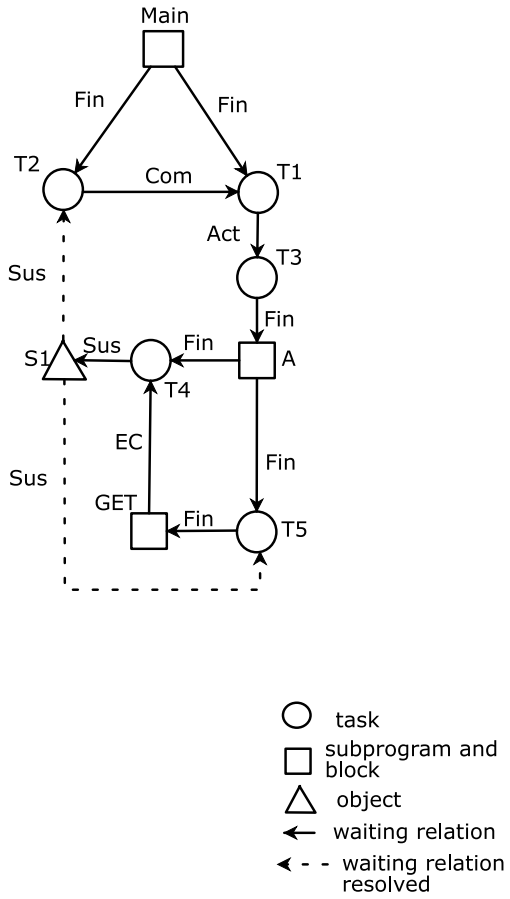

Fig. 3: TWFG of example program 3

3.5 Example Program 4: Tasking Deadlocks by modifying Program 3

This program is just like Program 3 but tasking deadlocks will occur in this program. These tasking deadlocks are caused by exchanging the order of two lines of code in T5 in the programs. The synchronization waiting cycles in this program are as follows:

$$T_1 \rightarrow_{Act} T_3 \rightarrow_{Fin} A \rightarrow_{Fin} T_4 \rightarrow_{Sus} S_1 \rightarrow_{Sus} T_2 \rightarrow_{Com} T_1$$

$$T_5 \rightarrow_{Fin} GET \rightarrow_{EC} T4 \rightarrow_{Sus} S_1 \rightarrow_{Sus} T5$$

Example Program 4

```
with Ada.Synchronous_Task_Control;
use  Ada.Synchronous_Task_Control;
procedure Main is
   task T1;
   task T2 is entry E2; end T2;
   S1 : Suspension_Object;
   task body T1 is
      task T3;
      task body T3 is
      begin
         A:
         declare
            task T4 is entry E4; end T4;
            task T5;
            procedure GET is
            begin
               T4.E4;
            end GET;
            task body T4 is
            begin
               Set_False(S1);
               Suspend_Until_True(S1);
               accept E4;
            end T4;

            task body T5 is
            begin
               GET;
               Set_True(S1);
            end T5;
         begin null; end A;
      end T3;
   begin null; end T1;
   task body T2 is
   begin
      select
         when FALSE => accept E2;
      or
         terminate;
      end select;
      Set_True(S1);
   end T2;
begin null; end Main;
```

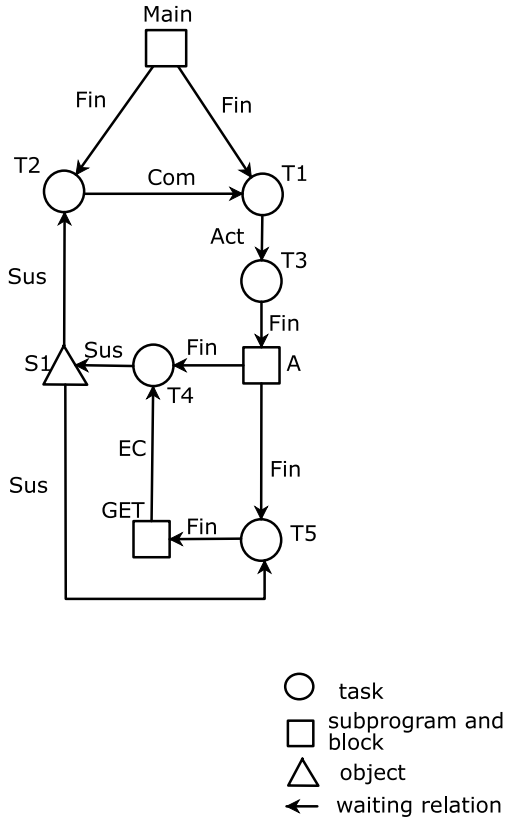

Fig. 4: TWFG of example program 4

3.6 Example Program 5: A Tasking Deadlock and a Tasking Livelock

This program has a tasking deadlock and a tasking livelock simultaneously. T1 and T2 form a suspension waiting cycle. However all other tasks are involved in a tasking livelock. The synchronization waiting cycle in this program is $T_1 \rightarrow_{Sus} S_1 \rightarrow_{Sus} T_2 \rightarrow_{Sus} S_2 \rightarrow_{Sus} T_1$. Livelocked tasks are T3 and T4. This type of tasking deadlock cannot be detected by using the TWFG, because the TWFG cannot handle tasking livelocks.

Example Program 5

```
with Ada.Synchronous_Task_Control;
use  Ada.Synchronous_Task_Control;
procedure Main is
   task T1;
   task T2;
   task T3 is entry E3; end T3;
   task T4 is entry E4; end T4;
   S1 : Suspension_Object;
   S2 : Suspension_Object;
   task body T1 is
   begin
      Set_False(S1);
      Suspend_Until_True(S1);
      Set_True(S2);
   end T1;
   task body T2 is
   begin
      Set_False(S2);
      Suspend_Until_True(S2);
      Set_True(S1);
   end T2;
   task body T3 is
   begin
      loop
         T4.E4;
         accept E3;
      end loop;
      Set_True(S1);
   end T3;
   task body T4 is
   begin
      loop
         accept E4;
         T3.E3;
      end loop;
   end T4;
begin null; end Main;
```

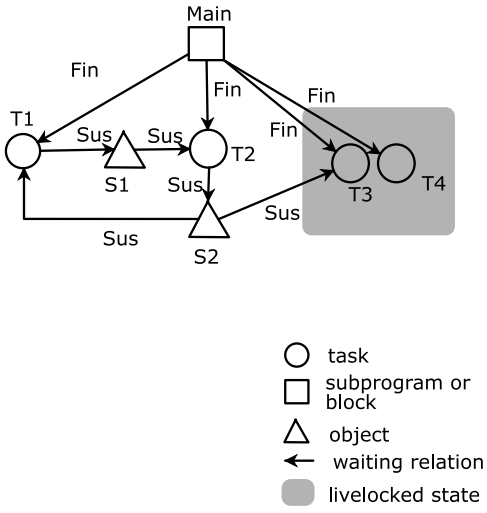

Fig. 5: TWFG of example program 5

4 Concluding Remarks

We have presented some examples of tasking deadlocks concerning tasking synchronization waiting relations defined in Ada 95's Real-Time Systems Annex.

We have extended the method of detecting tasking deadlocks at run-time for the core Ada 95 to the full Ada 95 by introducing suspension waiting arcs into the TWFG, and developed a new tasking deadlock detector which can detect all types of tasking deadlock. We will present our tasking deadlock detector in another paper.

References

[1] Barnes, J.: Programming in Ada 95 (2nd Edition), Addison-Wesley, 1998.

[2] Burns, A., Wellings, A.: Concurrency in Ada (2nd Edition), Cambridge, 1998.

[3] Cheng, J.: A Classification of Tasking Deadlocks, ACM Ada Letters, Vol.10, No.5, pp.110-127, 1990.

[4] Cheng, J.: Task-Wait-For Graphs and Their Application to Handling Tasking Deadlocks, Proc. 3rd ACM Annual TRI-Ada Conference, pp.376-390, 1990.

[5] Cheng, J., Ushijima, K.: Tasking Deadlocks in Ada 95 Programs and Their Detection, in A. Strohmeier (ed.), "Reliable Software Technologies - Ada-Europe '96," Lecture Notes in Computer Science, Vol. 1088, pp.135-146, Springer-Verlag, 1996.

[6] Conn, R.: Software Version Description(SVD) and Software User's Manual(SUM) Source Code Analysis Tool Construction Domain-Specific Kit(SCATC DSK), Public Ada Library, 1998.

[7] International Organization for Standardization Information Technology - Programming Language - Ada, ISO/IEC 8652:1995(E), 1995.

[8] Nonaka, Y., Cheng, J., Ushijima, K.: A Tasking Deadlock Detector for Ada 95 Programs, Ada User Journal, Vol.20, No.1 pp.79-92, 1999.