Run-Time Detection of Tasking Deadlocks in Real-Time Systems with the Ada 95 Annex of Real-Time Systems

Jingde Cheng

Saitama University

Outline of the Talk

- Background and Motivation
- Basic Notions and Terminology about Deadlocks and Livelocks
- Tasking Deadlocks in Real-Time Systems with the Ada 95 Annex of Real-Time Systems
- The Task-Wait-For Graph (An Extension to the Old One)
- Run-time Detection of all Types of Tasking Deadlocks
- Concluding Remarks

Tasking Deadlocks in Ada Programs

- **Deadlock as an old and eternal issue in concurrent systems**
  - **Deadlock is always a serious issue in concurrent systems, and, of course, may occur in any concurrent system with Ada.**
- **Tasking deadlocks**
  - **A tasking deadlock in a concurrent Ada program is a situation where some tasks form a circular waiting relation at some synchronization points that cannot be resolved by the program itself (including the behavior of other tasks), and hence these tasks can never proceed with their computation by themselves.**

Strategies for Handling deadlocks

- **Deadlock prevention**
  - **The developers of a system guarantee that deadlocks can never occur in the system.**
- **Deadlock avoidance**
  - **The developers of a system make the system able to detect potential deadlocks in advance and take some action to ensure that the deadlocks will not occur.**
- **Deadlock resolution**
  - **A system is developed such that it allows deadlocks to occur and then take some action to detect and resolve the deadlocks.**
- **Which is better?**
  - **Which deadlock handling strategy is suitable for a system depends on the characteristics of that system.**

Deadlock Detection

- **Deadlock detection as a common technique in handling deadlocks**
  - **Deadlock prevention needs a deadlock detection method to analyze the target program to determine whether the system is free of deadlock or not.**
  - **Deadlock avoidance and deadlock resolution need some method to detect deadlocks before or after their occurrences during the execution of the target program.**
- **Requirements for an ideal deadlock detection method for Ada programs**
  - ***Completeness*: the method must be able to detect any tasking deadlock in any arbitrary Ada program.**
  - ***Soundness*: the method must do not report any nonexistent tasking deadlock in any arbitrary Ada program.**
  - ***Efficiency*: the method must be able to be implemented such that it can detect tasking deadlocks in any arbitrary Ada program using a tractable space in a tractable time.**

Motivation and Goal

- **The state-of-the-art of tasking deadlock detection in Ada programs**
  - **None of those tasking deadlock detection methods and tools proposed until now can satisfy all of the three basic requirements completely.**
  - **For the core Ada 95, the current best result is our run-time detection method which can certainly detect all types of tasking deadlocks, without report of any nonexistent tasking deadlock, in a class of Ada 95 programs that are livelock-free. (Ada-Europe 1996, Ada-UK 1998)**
  - **For the full Ada 95, no work has been reported.**
- **Goal of this work**
  - **Develop a highly portable tasking deadlock detector for Ada 95 programs such that it can act as a component of any real-time system with the Ada 95 annex of real-time systems to detect all types of tasking deadlocks in the system at run-time.**

Basic Notions and Terminology about Deadlocks and Livelocks

- **Task and task type**
  - **We use the term '*task*' to represent an execution of a task unit, and use the term '*task type*' and/or '*single task*' to represent the source code of a task unit.**
- **Blocked tasks**
  - **A task in an Ada program is said to be *blocked* in an execution state of the program if it is waiting at a synchronization point in its thread of control for synchronization with one or more other tasks or even itself and this waiting state will be kept until either the synchronization has occurred or the task is aborted.**

Basic Notions and Terminology about Deadlocks and Livelocks

- **A *tasking deadlock* in an Ada program is an execution state of the program where some synchronization waiting relations among some tasks blocked at some synchronization points form a cycle that cannot be resolved by the program itself (including the behavior of other tasks), and hence these blocked tasks can never proceed with their computation by themselves. Any of the blocked tasks involved in the cycle is said to be *deadlocked*.**
- **A blocked task in an execution state is said to be *deadlock-blocked* if it is waiting for synchronization with a deadlocked task but not involved in the cycle the deadlocked task is involved in.**
- **Note that a task waiting for synchronization with several tasks may be deadlocked in a tasking deadlock and at the same time be blocked by another tasking deadlock.**

Basic Notions and Terminology about Deadlocks and Livelocks

- **Obviously, from the viewpoint of deadlock resolution, to break the waiting of a task blocked by a tasking deadlock cannot change the deadlock state of any deadlocked task in the tasking deadlock, and hence has no effect on resolution of that tasking deadlock.**
- **Therefore, if a tasking deadlock detection method does not explicitly distinguish deadlocked tasks from deadlock-blocked tasks, then the detection method cannot work well for tasking deadlock resolution.**

Basic Notions and Terminology about Deadlocks and Livelocks

- A *tasking livelock* in an Ada program is an infinite series of execution states of the program where each member of a group of tasks keeps forever rendezvousing with only tasks in the group and hence can never respond to a synchronization request from any task outside the group. Any of the tasks involved in the group is said to be *livelocked*.
- A blocked task in an execution state is said to be *livelock-blocked* if it is waiting for synchronization with a livelocked task.
- Note that a task waiting for synchronization with several tasks may be blocked by a tasking deadlock and at the same time be blocked by a tasking livelock.

Basic Notions and Terminology about Deadlocks and Livelocks

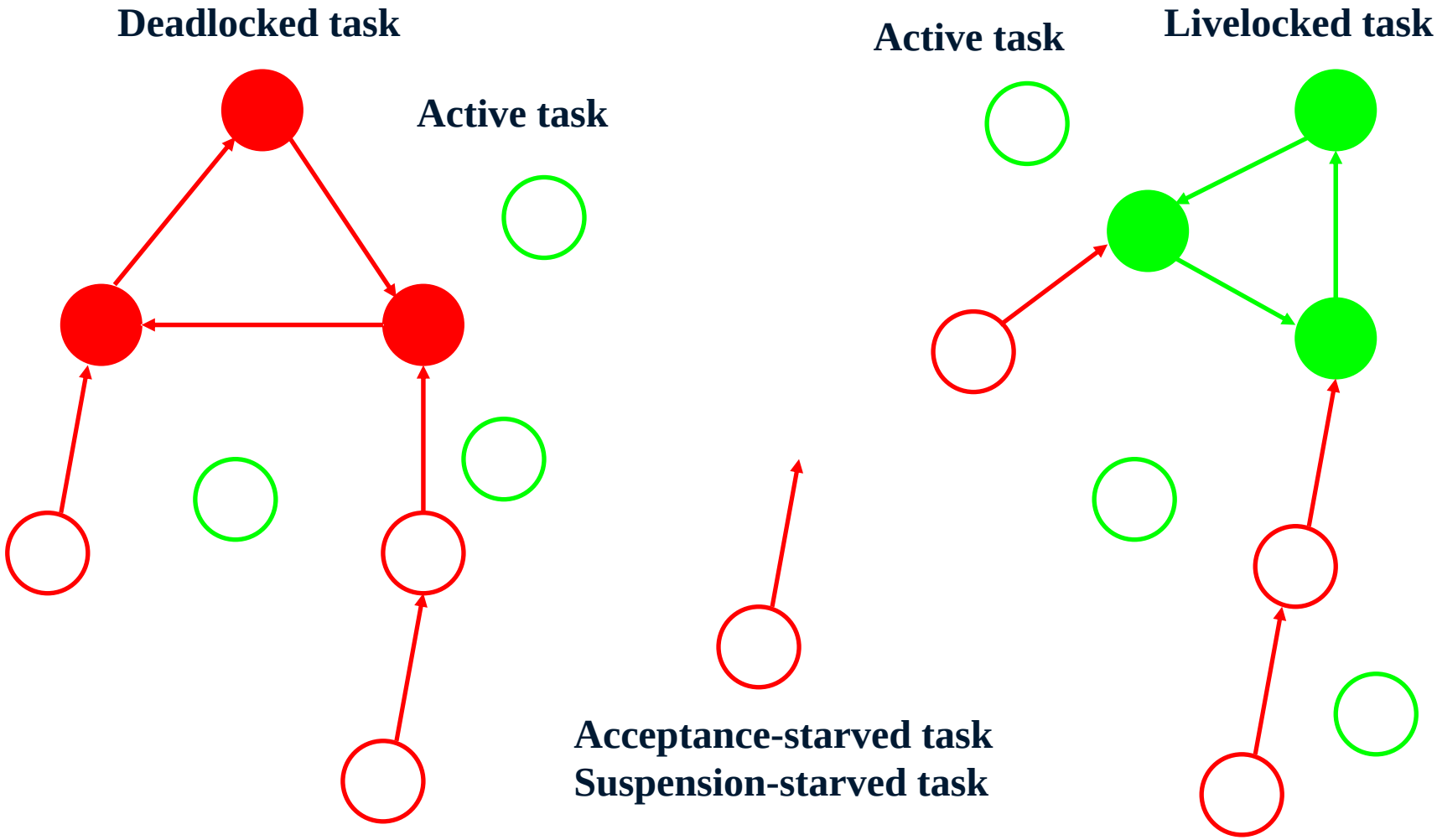
- **Both livelock and deadlock may occur in the same Ada program. In this case, the detection of the tasking deadlock may depend upon the ability to detect the tasking livelock. At present, how to detect a livelock in a concurrent program is a completely open problem.**
- **From the viewpoint of deadlock resolution, if a local tasking deadlock and a local tasking livelock exist simultaneously during an execution of an Ada program, to break the waiting of a livelock-blocked task cannot change the deadlock state of any deadlocked task in the tasking deadlock, and hence has no effect on resolution of that tasking deadlock.**
- **Therefore, if a tasking deadlock detection method does not explicitly distinguish deadlocked tasks from livelock-blocked tasks, then the detection method cannot work well for deadlock resolution.**

Basic Notions and Terminology about Deadlocks and Livelocks

- **Besides deadlocks and livelocks, a task may be blocked forever when it is waiting for accepting an entry call from some unspecified task or tasks even if such tasks are never existent. In an execution state of an Ada program, a blocked task waiting for accepting an entry call from other tasks is said to be *acceptance-starved* if it is nether deadlocked, nor deadlock-blocked, nor livelock-blocked, and there does not exist an entry call to one of its entries from any other task.**

- **Similarly, in an execution state of an Ada program, a blocked task waiting for suspension until the value of the suspension object is true is said to be *suspension-starved* if it is neither deadlocked, nor deadlock-blocked, nor livelock-blocked, and no task has a statement to set the value of the suspension object true.**

Examples of Blocked, Deadlocked, and Livelocked Tasks



Synchronization Waiting Relations between Tasks

- **Synchronization Waiting Relations**
  - **(1) Activation waiting (Ada 83)**
  - **(2) Finalization waiting (Ada 83)**
  - **(3) Completion waiting (Ada 83)**
  - **(4) Acceptance waiting (Ada 83)**
  - **(5) Entry-calling waiting (Ada 83)**
  - **(6) Protection waiting (Ada 95)**
  - **(7) Protected-entry-calling waiting (Ada 95)**
  - **(8) Suspension waiting (Ada 95 annex of real-time systems)**
- **Common property of the synchronization waiting relations**
  - **The waiting task in any waiting relation cannot change its own waiting state if there is not an event, including the execution of an abort statement, in the execution of its partner or partners.**

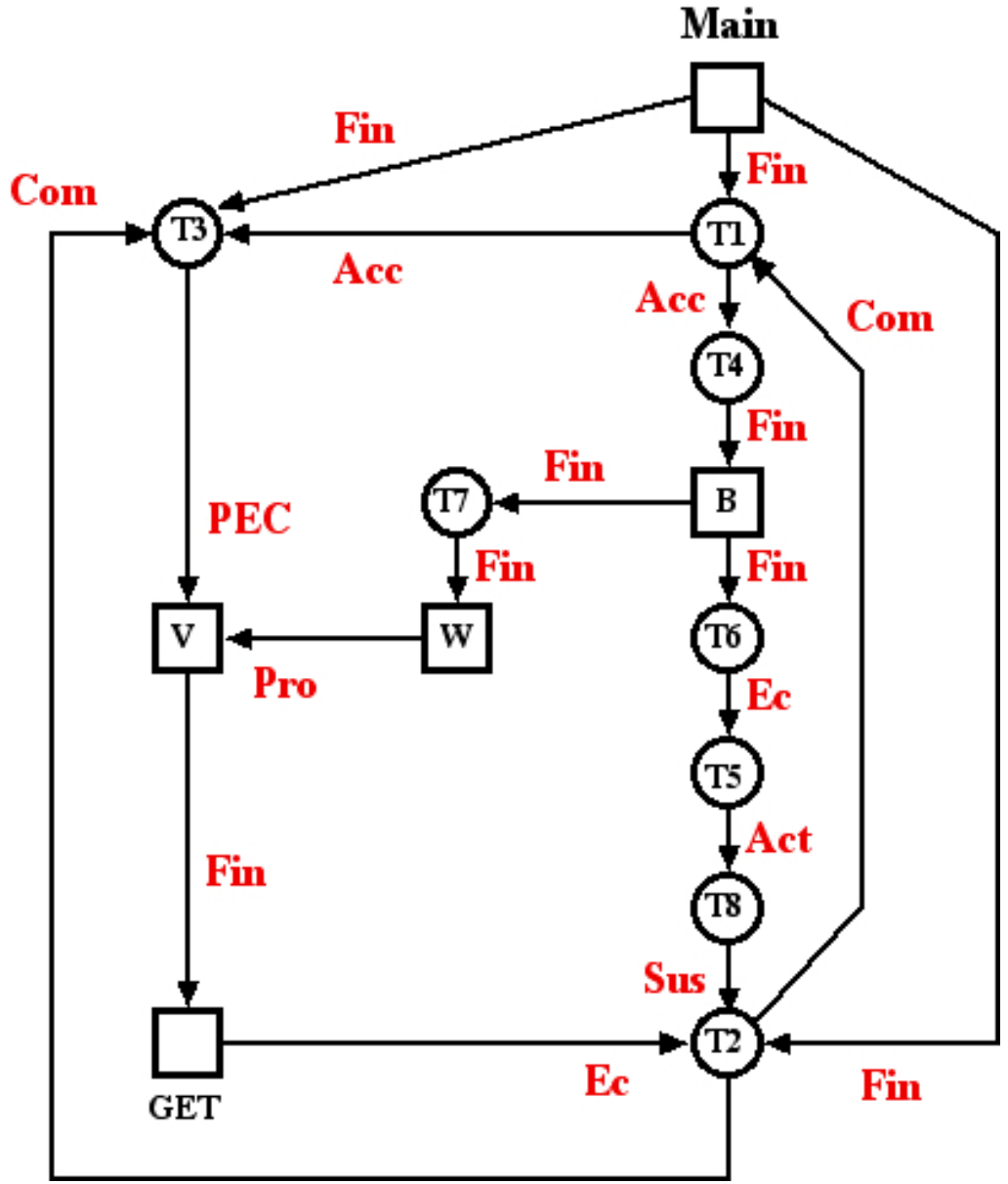
Tasking Deadlocks in the Full Ada 95 Programs

- **Tasking deadlocks**
  - **A circular waiting relation formed among some tasks implies that a tasking deadlock might have occurred there.**
  - **If the circular waiting relation is stable, then a tasking deadlock has certainly occurred there.**
- **Tasking deadlock classification [Cheng, 1990]**
  - **If we consider that different combinations of the waiting relations form different tasking deadlocks, then we can completely classify tasking deadlocks.**
  - **For Ada 83 programs there may be at most 31 different types of tasking deadlocks, while there are only 18 different types of tasking deadlocks which may really occur in Ada 83 programs.**
  - **For the full Ada 95 programs, there may be at most 255 different types of tasking deadlocks.**

The Task-Wait-For-Graph [Cheng, 1990-2006]

- **Tasking objects**
  - 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 underway
- **TWFG**
  - An arc-classified digraph whose vertexes represent tasking objects and whose different types of arcs represent different synchronization waiting relations

The Task-Wait-For-Graph of the Example Program



Run-time Detection of all Types of Tasking Deadlocks

- **States of tasks**
  - **Proposition 1 In an execution state of an Ada program, if a task is blocked, then it must keep one and only one of the following five states: deadlocked, deadlock-blocked, livelock-blocked, acceptance-starved, or suspension-starved.**
- **Arc-type exclusiveness of TWFG**
  - **Proposition 2 For any vertex v of a TWFG (P, t), all outgoing arcs of v are type exclusive, i.e., if out-degree(v)>0 then all outgoing arcs of v must be a subset of any one of Act, Fin, Com, Acc, EC, Pro, PEC, and Sus of the TWFG (P, t).**

Run-time Detection of all Types of Tasking Deadlocks

- **The necessary condition for occurrences of tasking deadlocks**
  - **Proposition 3 For any Ada program P, if a tasking deadlock occurs at time t in an execution of P, then there exists a cycle in TWFG (P, t) such that each arc of the cycle corresponds to a synchronization waiting relation involved in the tasking deadlock.**
- **The sufficient condition for occurrences of tasking deadlocks**
  - **Proposition 4 For any TWFG (P, t), if (1) there exists a cycle, and (2) for every OR-waiting path starting from every OR-waiting vertex in the cycle, either it is a part of a cycle or it is stable, then a tasking deadlock occurs in the execution of P at time t.**

Run-time Detection of all Types of Tasking Deadlocks

- **Run-time detection of tasking deadlocks**
  - **Having TWFGs as a formal representation for the waiting state of task synchronization in an execution of an Ada program, a run-time detector for tasking deadlocks can work by monitoring the tasking behavior of the program, managing a TWFG for the program, detecting cycles in the TWFG, and reporting detected tasking deadlocks.**
- **Monitoring the tasking behavior of an Ada program**
  - **The source program transformation approach: A target program P is transformed by a preprocessor into another Ada program P', called the subject program of P, such that P' preserves the tasking behavior of P and during its execution P' will communicate with a run-time monitor when each tasking event of P occurs in P' and pass information about the tasking event to the run-time monitor.**
  - **The run-time environment support approach: A run-time monitor is implemented as a component of the underlying run-time support environment of Ada and information concerning tasking events in a target Ada program is provided directly by the run-time monitor.**

Concluding Remarks

- **Our works**
  - **Although the work and results presented in this paper are an extension of our previous work, we have made some important improvements on basic notions, terminology, formal definitions on our previous ones in order to provide a standard reference on Ada 95 tasking deadlocks for designers and developers of real-time systems with the Ada 95's annex of real-time systems.**
  - **We are developing a tasking behavior monitor with general-purpose as an Ada 95 generic package such that its instances can be used as permanent components in various target systems which need to monitor tasking behavior at run-time by themselves based on the self-measurement principle.**

Concluding Remarks

- **Future works**
  - **Designing and developing complete, sound, and efficient detection methods and tools for run-time detection of all tasking deadlocks.**
  - **Sloving the open problem how to detect tasking deadlocks when tasking livelocks occur simultaneously.**
  - **Investigating tasking deadlocks in Ada 2005 programs.**

Thanks

Thank you very much for your attention