Restrizioni di concorrenza

RTS

Anno accademico 2008/9
Sistemi *Real-Time*

Tullio Vardanega, tullio.vardanega@math.unipd.it

Concorrenza esplicita e predicibilità

- Vi è discrepanza tra i paradigmi di concorrenza supportati da linguaggi e sistemi operativi e i modelli concettuali assunti dalle tecniche di analisi
- Mentre possiamo confidare che le tecniche analisi si facciano sempre più sofisticate dobbiamo assicurare che il comportamento dell'eseguibile sia riconducibile a quanto convalidato dall'analisi

Prerequisiti – 1

- Un programma concorrente deve possedere due importanti proprietà
  - ***Safety*** **: i *task* che compongono il sistema non devono mai entrare in uno stato insicuro**
    - *Deadlock*, *livelock*
  - ***Liveness*** **: tutti gli stati desiderabili del sistema devono essere prima o poi raggiunti dai suoi *task***
    - L'esecuzione del sistema deve sempre produrre valore
    - Una esecuzione che non produca valore è in stato di *livelock*
  - **In una sistema concorrente *real-time* la proprietà di *liveness* deve essere raggiunta entro *deadline* fissate**
    - Perciò parliamo di *bounded liveness*

Prerequisiti – 2

- La semantica dei linguaggi di programmazione *mainstream* non è definita in modo formale
- Per provare proprietà del sistema bisogna derivare dal programma un suo modello analizzabile
  - Il legame tra programma e modello deve essere dimostrabile
  - L'uso di specifici stili di programmazione agevola la dimostrazione
- Un approccio basato su prova è preferibile all'uso di *model checking*
  - Ciò incentiva l'uso di *pattern* per i quali esista prova di proprietà

Un modello di concorrenza – 1

- Vogliamo un modello di concorrenza predicibile
  - E dunque analizzabile staticamente
- Per poter utilizzare concorrenza esplicita a programma
  - Per aver pieno controllo della corrispondenza tra i *task* a livello applicazione e i *thread* a livello *kernel*
- Ci servono restrizioni sui costrutti disponibili e sul comportamento ammissibile a tempo d'esecuzione
  - Per introdurre determinismo ove la semantica originaria è non deterministica
  - Per ottenere efficienza prestazionale e certificabilità del sistema

Un modello di concorrenza – 2

- Vogliamo un modello centrato su *fixed-priority scheduling* e *ceiling priority protocol*
  - Da cui derivano gli stati di *thread* studiati in lezione L05
  - E anche le garanzie di *bounded priority inversion* e di assenza di *deadlock* nella condivisione di risorse
- Assumiamo l'uso di *response time analysis*
  - Dobbiamo dunque attingere dal sistema (specifica e realizzazione) tutte le informazioni che servono per alimentarla

Assunzioni di modello – 1

- I *task* non interagiscono direttamente ma lo fanno attraverso *protected object*
  - Un PO fornisce funzioni di controllo (protocollo) di accesso a risorse condivise
    - Secondo il *ceiling priority protocol*
  - Ma anche funzioni di sincronizzazione
    - Essenziali per ottenere *task* con comportamento non periodico
  - E ovviamente anche la combinazione delle due funzioni
- L'analisi statica richiede un insieme statico di *task*
  - Vogliamo pieno controllo sulle modalità e sul tempo di elaborazione dei *task*
  - Ammettiamo solo *task* e PO creati a livello di libreria
    - Dobbiamo proibire i costrutti di creazione dinamica

Assunzioni di modello – 2

- Vogliamo restrizioni note al compilatore
  - Così che la conformità del programma possa essere valutata e garantita
- Vogliamo un profilo di linguaggio
  - Tecnicamente definiamo un *alternative mode of operation*
  - Per essere utile deve essere specificato come elemento standard della stessa specifica del linguaggio
    - Così che un programma conforme al profilo abbia semantica portabile

Restrizioni – 1

```
                          pragma Task_Dispatching_Policy (FIFO_Within_Priorities);
                          pragma Locking_Policy (Ceiling_Locking);
                          pragma Detect_Blocking;
pragma Restrictions
 (No_Abort_Statements,
  No_Dynamic_Attachment,
  No_Dynamic_Priorities,
  No_Implicit_Heap_Allocations,
  No_Local_Protected_Objects,
  No_Local_Timing_Events,
  No_Protected_Type_Allocators,
  No_Relative_Delay,
  No_Requeue_Statements,
  No_Select_Statements,
  No_Specific_Termination_Handlers,
  No_Task_Allocators,             Max_Entry_Queue_Length => 1,
  No_Task_Hierarchy,              Max_Protected_Entries => 1,
  No_Task_Termination,            Max_Task_Entries => 0,
  Simple_Barriers,                No_Dependence => Ada.Asynchronous_Task_Control,
                                  No_Dependence => Ada.Calendar,
                                  No_Dependence => Ada.Execution_Time.Group_Budget,
                                  No_Dependence => Ada.Execution_Time.Timers,
                                  No_Dependence => Ada.Task_Attributes);
```


Restrizioni – 2

- Task_Dispatching_Policy (FIFO_Within_Priorities)
  - Fissa l'algoritmo di *scheduling* come FPS e con ordinamento FIFO tra *job* allo stesso livello di priorità
- Locking_Policy (Ceiling_Locking)
  - Fissa il protocollo di accesso a risorse condivise a *ceiling priority protocol*
- Detect_Blocking
  - Impone che tutte l'esecuzione di operazioni potenzialmente sospensive all'interno di PO causino Program_Error

Restrizioni – 3

- No_Dynamic_Attachment
  - Impone associazione statica tra procedure e interruzioni
- No_Relative_Delay
  - Impone l'uso di sospensioni assolute
- No_*_Allocators
  - Proibisce l'uso di costrutti di creazione dinamica di *task* e di PO
- No_Task_Hierarchy, No_Local_Protected_Object
  - Impone *task* e PO definiti a livello di libreria
- No_Abort_Statements, No_Task_Termination
  - Impone che i *task* non possano terminare

Restrizioni – 4

- No_Select_Statements
  - I *task* non possono avere comportamento non deterministico
- No_Requeue_Statements
  - Per la parte concernente i *task* discende dalla restrizione precedente
  - Per la parte concernente i PO impone attesa deterministica nelle chiamate con guardia
- Simple_Barriers
  - Impone che la valutazione delle espressioni di guardia non abbia effetti laterali

Restrizioni – 5

- **Max_Entry_Queue_Length => 1**
  Max_Protected_Entries => 1
  - **Insieme garantiscono determinismo nell'attesa su una coda di *entry* di PO**
- **Max_Task_Entries => 0**
  - **Impone che i *task* possano comunicare tra loro solo tramite PO**

Effetti delle restrizioni – 1

- ***Static existence model***
  - **L'insieme di *task* e di interruzioni è staticamente fissato, ha proprietà e attributi staticamente determinati ed è completamente operativo al termine dell'elaborazione**
    - Elaborazione = ciò che precede l'avvio dell'esecuzione del *main*
      - Il controllo sull'ordine di elaborazione è problematico in generale ma ancor di più nei programmi concorrenti
      - **pragma** Partition_Elaboration_Policy (Sequential)
        i *task* (tutti dichiarati a livello di libreria) vengono attivati solo dopo il *main*
- ***Static synchronization and communication model***
  - **Diretta conseguenza del precedente requisito**

Effetti delle restrizioni – 2

- ***Deterministic memory usage***
  - **Nessun uso implicito di allocazione dinamica della memoria**
  - **Il bisogno complessivo di memoria deve essere valutabile staticamente per avere certezza che a tempo d'esecuzione non emergano necessità non soddisfabili**
- ***Deterministic execution model***
  - **Nelle attese di sincronizzazione, nel blocco, nelle sospensioni**