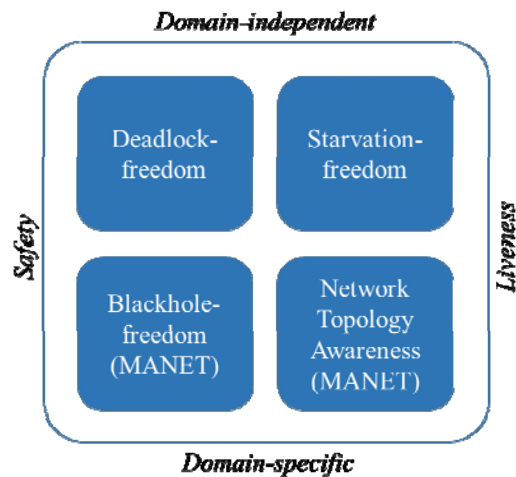


Analisi di starvation e deadlock mediante ASM

Nota

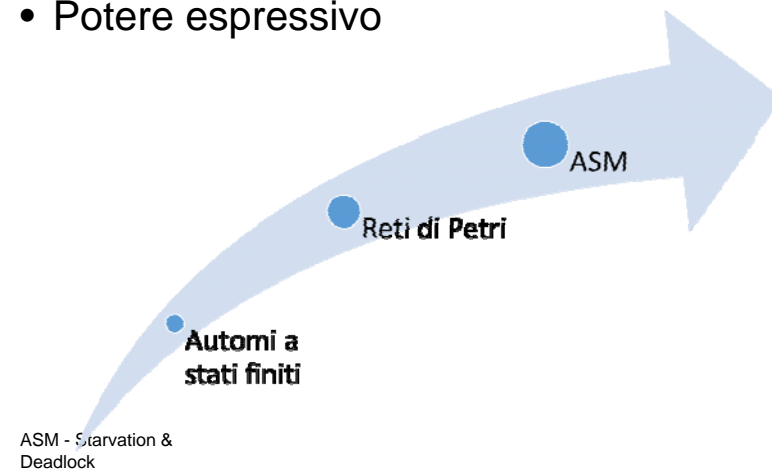
- Questo modulo è realizzato modificando opportunamente :
 - materiale prodotto dal Dr. Gennaro Vessio per il seminario tenuto per il corso dell'a.a. 2016-17, rivisto dal docente
 - articoli di letteratura citati in bibliografia

Classi di Proprietà



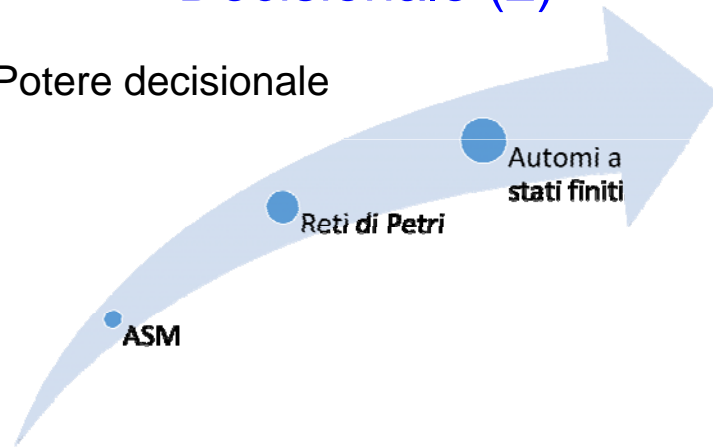
Potere Espressivo vs Potere Decisionale (1)

- Potere espressivo



Potere Espressivo vs Potere Decisionale (2)

- Potere decisionale



ASM - Starvation & Deadlock

5

Problema

- Allo stato dell'arte, l'analisi di proprietà con le ASM fa ricorso a tecniche basate su model checking
 - Ma, la traduzione della ASM in esame nell'input richiesto dal model checker causa perdita di potere espressivo
- Alcuni formalismi permettono caratterizzazioni operazionali di proprietà
- Obiettivo: Dotare le ASM di un framework analogo

ASM - Starvation & Deadlock

6

RICHIAMO: Predicati sugli Stati

- Una ASM può modellare oggetti di arbitraria complessità
 - comprendere la semantica degli stati di una ASM è difficile
- Il concetto di predicato è utile per esprimere la condizione di attesa

Definition (predicate over ASM state). A predicate φ over an ASM state s is a first-order formula defined over the locations in s such that $s \models \varphi$

ASM - Starvation & Deadlock

7

STARVATION

8

Generalità

- In letteratura non esiste una definizione formale e univoca di starvation
- In generale, è l'**impossibilità** da parte di un processo concorrente di compiere progressi computazionali
 - Attesa di una risorsa condivisa
 - Attesa di un messaggio
 - ...

Condizione Necessaria

Necessary condition (general statement)

1. The execution of a process p_i ($i = 1, \dots, n$) requires a service provided by p_j ($j = 1, \dots, n, j \neq i$)
2. The process p_i is forced to wait for the desired service



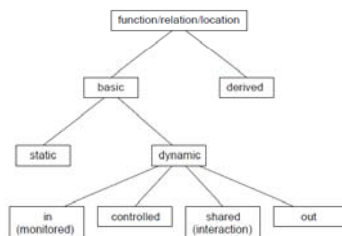
Necessary condition (ASM-based statement)

1. A move of an agent a_i ($i = 1, \dots, n$) requires a service provided by a_j ($j = 1, \dots, n, j \neq i$)
2. The agent a_i is forced to wait for the desired service

Dipendenza

- La condizione (1) esprime il concetto di **dipendenza** da altri agenti

Definition (risky function). A *risky function* is: (i) a monitored or shared function; or (ii) a controlled or derived function *determined* by a risky function



Inattività

- La condizione (2) esprime il concetto di **inattività** dell'agente durante l'attesa
 - Per esprimere l'inattività abbiamo bisogno di esprimere il ciclico **ritorno** verso uno stesso stato

Definition (risky predicate). Let D be a DASM including the agents a_i and a_j . A *risky predicate* φ_i over a state s of $ASM(a_i)$ is a first-order predicate defined over (at least) a risky location in s such that $s \models \varphi_i$, and its truth value depends also on $ASM(a_j)$

Vulnerable Rule

- Lo stato di una ASM cambia solo come effetto dell'applicazione di una rule

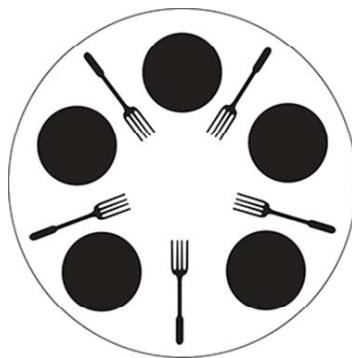
Definition (vulnerable rule). Let φ_r be a risky predicate such that a state s , which models φ_r , exists. A rule is said to be *vulnerable* if:

- Its condition includes a logical conjunction with one or more risky functions
- The truth value of φ_r changes only as a result of the execution of the updates of some vulnerable rule in s

Teorema

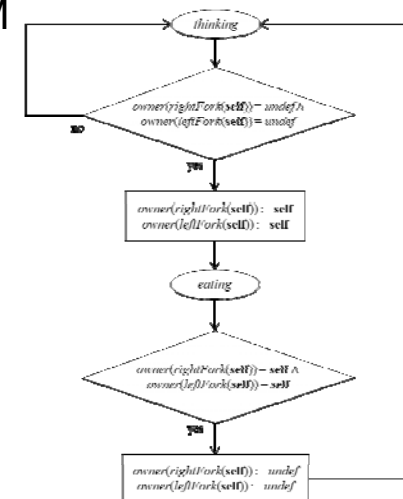
- Theorem** (starvation freedom) *Let D be a DASM composed by the family of pairs $(a_i, ASM(a_i))$. If each $ASM(a_i)$ is without vulnerable rules, then D is starvation-free*
 - Dimostrazione (schema): The absence of vulnerable rules implies that at least one of the two sub-conditions of the ASM-based necessary condition for starvation is negated
- Attenzione: non vale il viceversa

Esempio: 5 filosofi (1)



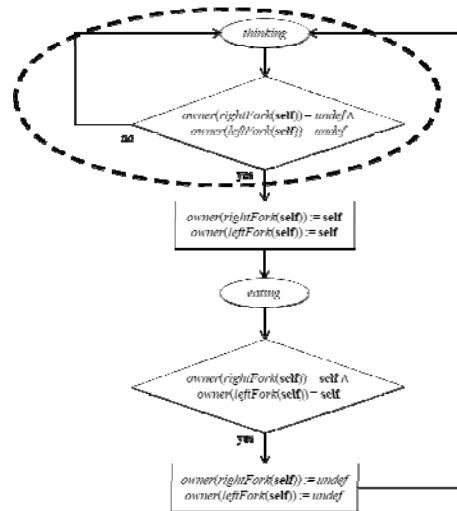
Esempio: 5 filosofi (2)

- Modello ASM



Esempio: 5 filosofi (3)

- Modello ASM



ASM - Starvation & Deadlock

17

Esempio: 5 filosofi (4)

- Analisi del modello
- È possibile definire due predicati sugli stati:
 - thinking**: $\neg(\text{owner}(\text{rightFork}(\text{self})) = \text{self} \vee \text{owner}(\text{leftFork}(\text{self})) = \text{self})$
 - eating**: $\text{owner}(\text{rightFork}(\text{self})) = \text{self} \wedge \text{owner}(\text{leftFork}(\text{self})) = \text{self}$
- owner* è una funzione di rischio
- thinking* è un predicato di rischio
- La prima regola è vulnerabile
- Il modello **soffre** di starvation

ASM - Starvation & Deadlock

18

DEADLOCK

Generalità

- È una situazione di stallo in cui tutti i processi sono vicendevolmente bloccati in attesa
 - di una risorsa condivisa
 - di un evento
 - di un messaggio
 - ...

19

ASM - Starvation & Deadlock

20

Condizione Necessaria (1)

Necessary condition (general statement)

1. Processes claim exclusive control of the resources they require (*mutual exclusion*)
2. Processes hold resources already allocated to them while waiting for the other requested resources (*resource holding*)
3. Resources cannot be removed from the processes holding them until they are used to completion (*no preemption*)
4. A circular chain of processes exists so that each process holds (at least) one resource that is requested by the next process in the chain. In other words, there is a set of processes p_1, \dots, p_n such that p_i is waiting for a resource held by $p_{i+1} \bmod n$ (*circular wait*)

Condizione Necessaria (2)

Necessary condition (ASM-based statement)

1. Agents hold resources already allocated to them while waiting for the other requested resources (*resource holding*)
2. Resources cannot be removed from the agents holding them until they are used to completion (*no preemption*)
3. A circular chain of agents exists so that each agent holds (at least) one resource that is requested by the next agent in the chain. In other words, there is a set of agents a_1, \dots, a_n such that a_i is waiting for a resource held by $a_{i+1} \bmod n$ (*circular wait*)

Manca il concetto di mutua esclusione!

Funzioni di Ownership

- Abbiamo bisogno di esprimere il concetto di **possesso** di una risorsa

Definition (ownership function). Let a DASM be given, including the set $Agents = \{a_1, \dots, a_n\}$, and let $Resources = \{r_1, \dots, r_n\}$ be a set of shared resources. An *ownership function* is a function *owner*, with a variable number of arguments in $Resources$ and values in $Agents \cup \{undef\}$, such that it is: (i) a shared function; or (ii) a derived function defined over an ownership function.

Resource Holding

Resource holding. Let φ be a predicate over the states, defined over an ownership 1-ary function *owner* of $ASM(a_i)$, such that it is in the form $owner(r_i) = a_i \wedge \neg(owner(r_j) = a_i)$, with $i \neq j$. The fulfillment of such a predicate indicates that the agent a_i is waiting for a resource r_j while another requested resource r_i has already been allocated to it. Resource holding holds if, for every $ASM(a_i)$, representing each process of the system, a state s , such that $s \models \varphi$, exists. We call φ *holding predicate*

No Preemption & Circular Wait

No preemption. For every $ASM(a_i)$, representing each process of the system, there is no rule updating an ownership location $owner(r_i)$ to a_i if $owner(r_i)$ does not evaluate to *undef*

Circular wait. A circular chain arises when a global state S_p , resulting by the composition of the local states $s_1(a_1), s_2(a_2), \dots$, each satisfying a holding predicate φ , exists. When each $ASM(a_i)$ satisfies φ , for each agent a_i , $owner(r_i) = a_i \wedge \neg(owner(r_j) = a_i)$, with $i \neq j$

Esempio: 5 filosofi (1)

```
PhilosopherProgram(p_i) = {  
  if owner(rightFork(self)) = undef then  
    owner(rightFork(self)) := self  
  if owner(rightFork(self)) = self  $\wedge$   
    owner(leftFork(self)) = undef then  
    owner(leftFork(self)) := self  
  if owner(rightFork(self)) = self  $\wedge$   
    owner(leftFork(self)) = self then {  
    Eat(self)  
    owner(rightFork(self)) := undef  
    owner(leftFork(self)) := undef  
  }  
}
```

Esempio: 5 filosofi (2)

- Analisi del modello
- È possibile definire tre predicati sugli stati:
 - **thinking**: $\neg(owner(rightFork(self)) = self \vee owner(leftFork(self)) = self)$
 - **holdingRightFork**: $owner(rightFork(self)) = self \wedge \neg(owner(leftFork(self)) = self)$
 - **eating**: $owner(rightFork(self)) = self \wedge owner(leftFork(self)) = self$
- *holdingRightFork* è un holding predicate
- la no preemption è soddisfatta
- una catena circolare emerge quando tutti gli agenti eseguono la loro prima regola
- Il modello **soffre** di deadlock

Sviluppi

- La nozione di deadlock è più generale di quella illustrata, che considera solo risorse condivise
 - Si deve generalizzare
 - Si deve dimostrare la proprietà di deadlock-freedom per DASM

Bibliografia

- Bianchi A., Pizzutilo S., Vessio G. Applying Predicate Abstraction to Abstract State Machines. Enterprise, Business-Process and Information Systems Modeling, LNBIP 214, pp. 283-292, Springer, 2015
- Bianchi A., Pizzutilo S., Vessio G. Reasoning on Starvation in AODV using Abstract State Machines. Journal of Theoretical and Applied Information Technology, 84(1), pp. 140-149, 2016
- Bianchi A., Pizzutilo S., Vessio G. Towards an ASM-based Characterization of the Deadlock-freedom Property. In: 11th International Conference on Software Paradigm Trends (ICSOFT-PT 2016), Lisbon, Portugal, pp. 123-130, SciTePress, 2016
- Bianchi A., Pizzutilo S., Vessio G., "An ASM-based characterisation of starvation-free systems", International Journal of Parallel, Emergent and Distributed Systems, 2017, pp.1-17