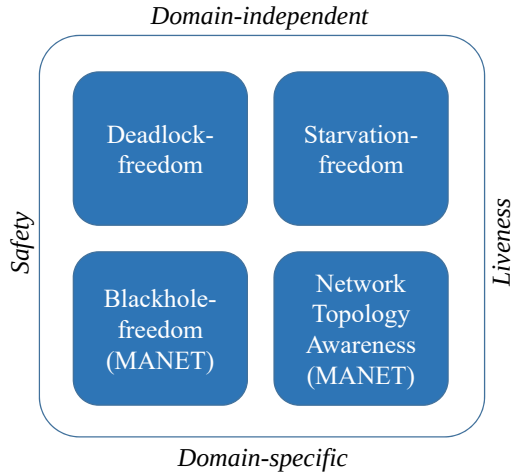
Analisi di starvation e deadlock mediante ASM

Gennaro Vessio
gennaro.vessio@uniba.it

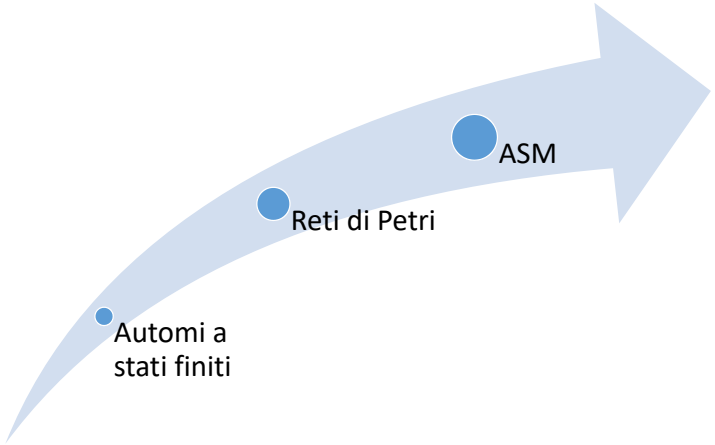
Contesto

- La verifica di un sistema software consiste nel verificare se esso soddisfa o meno determinate proprietà
- Una proprietà esprime un comportamento desiderato del sistema
- L'approccio formale propone di condurre tale analisi su un modello formale del sistema

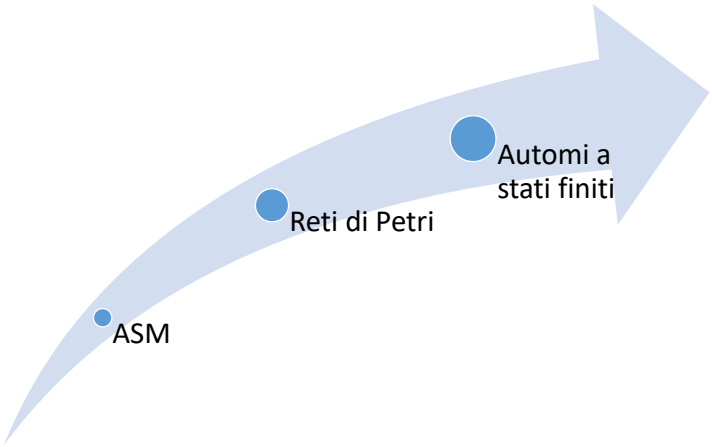
Classi di proprietà



Potere espressivo



Potere decisionale



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

Soluzione

- Numerosi formalismi, p.e. le reti di Petri, sono dotati di caratterizzazioni operazionali di proprietà
  - Reachability
  - Reversibility
  - …
- *Dotare le ASM di un framework analogo?*

Starvation

Concetto generale

- 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
  - …

Definizioni informali (classiche)

1972	**Dijkstra**	*"Although all individual eating actions take only a finite period of time, a person may be kept hungry for the rest of his days"*
1983	**Lamport**	*"A process eventually receives service"*
1985	**Alpern & Schneider**	*"A process makes progress infinitely often"*

Definizioni informali (più recenti)

2005	**Pnueli, Podelski & Rybalchenko**	*"The* [starvation-freedom] *property relies on justice assumptions that none of the processes idles forever in some location"*
2007	**Tanenbaum**	*"Some policy is needed to make a decision about who gets which resource when. This policy, although seemingly reasonable, may lead to some processes never getting service even though they are not deadlocked"*
2008	**Baier & Katoen**	*"Each waiting process will eventually enter its critical section"*

Condizione necessaria

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

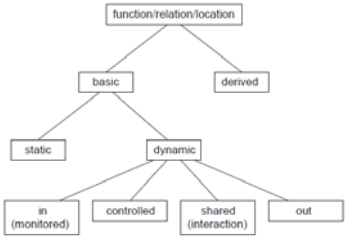
⇩

Necessary condition (ASM-based statement)
1. A *move* of an agent a_i ($i = 1, \ldots, n$) requires a service provided by a_j ($j = 1, \ldots, n, j \neq i$)
2. The agent a_i is forced to wait for the desired service

Dipendenza

- La (1) cattura la dipendenza da altri agenti
  - *Come esprimere tale dipendenza?*

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 (2) cattura l'inattività dell'agente fintanto che attende
  - *Come esprimere tale inattività?*
  - Occorre esprimere il ritorno ciclico verso uno o più stati computazionali

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

Predicati sugli stati

- Una struttura algebrica può modellare oggetti di arbitraria complessità ⇨ comprendere la semantica degli stati di una ASM è difficile
- Il concetto di predicato torna 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 $s \vDash \varphi$

Regola vulnerabile

- Lo stato di una ASM può cambiare (o non cambiare) solo come conseguenza dell'applicazione di regole

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:

1. Its condition includes a logical conjunction with one or more risky functions
2. 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*

Sketch of proof. 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 □

- Non vale il viceversa! *Perché?*

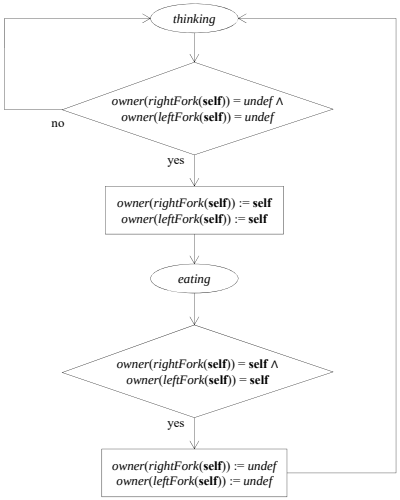
Cinque filosofi a cena

- Cinque filosofi siedono a cena di fronte a un piatto di spaghetti
- Ciascun filosofo, nella sua vita, pensa o mangia, rigorosamente con due forchette
- Problema: a ogni filosofo è stata servita un'unica forchetta

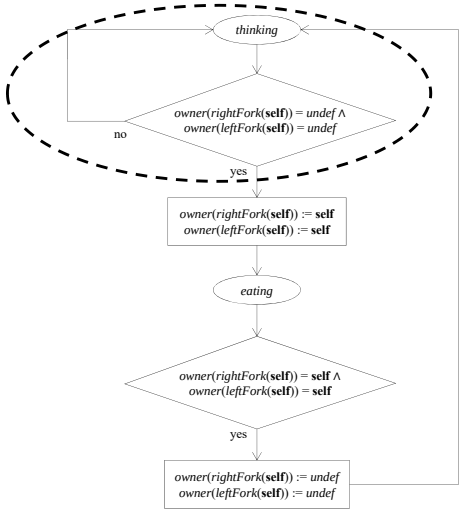
Cinque filosofi a cena

- *Cosa succede se non neghiamo la resource holding?*
- *Cosa succede se neghiamo la resource holding?*

Modellazione in ASM



Modellazione in ASM



Analisi del modello

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

Deadlock

Concetto generale

- In generale, il deadlock è la situazione di stallo in cui tutti i processi sono vicendevolmente bloccati in attesa
  - –Attesa di una risorsa condivisa
  - –Attesa di un messaggio
  - –…

Condizione necessaria

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 $p_1, \ldots, p_n$ such that p_i is waiting for a resource held by p_{i+1} mod n (*circular wait*)

Condizione necessaria

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 $a_1, \ldots, a_n$ such that a_i is waiting for a resource held by a_{i+1} mod n (*circular wait*)

- *E la 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* = $\{a_1, \ldots, a_n\}$, and let *Resources* = $\{r_1, \ldots, r_n\}$ 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* ∪ {*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 $s \vDash \varphi$, exists. We call φ *holding predicate*

No preemption e 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_k, resulting by the composition of the local states $s_1(a_1)$, $s_2(a_2)$, …, 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 $i \neq j$

Di nuovo i filosofi a cena

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


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
- *Come risolvere il problema?*

Sviluppi futuri

- La nozione di deadlock è più generale del deadlock concernente le sole risorse condivise
  - Occorre generalizzare
  - Occorre dimostrare formalmente quando una DASM è deadlock-free

Riferimenti

1. 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
2. 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
3. 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