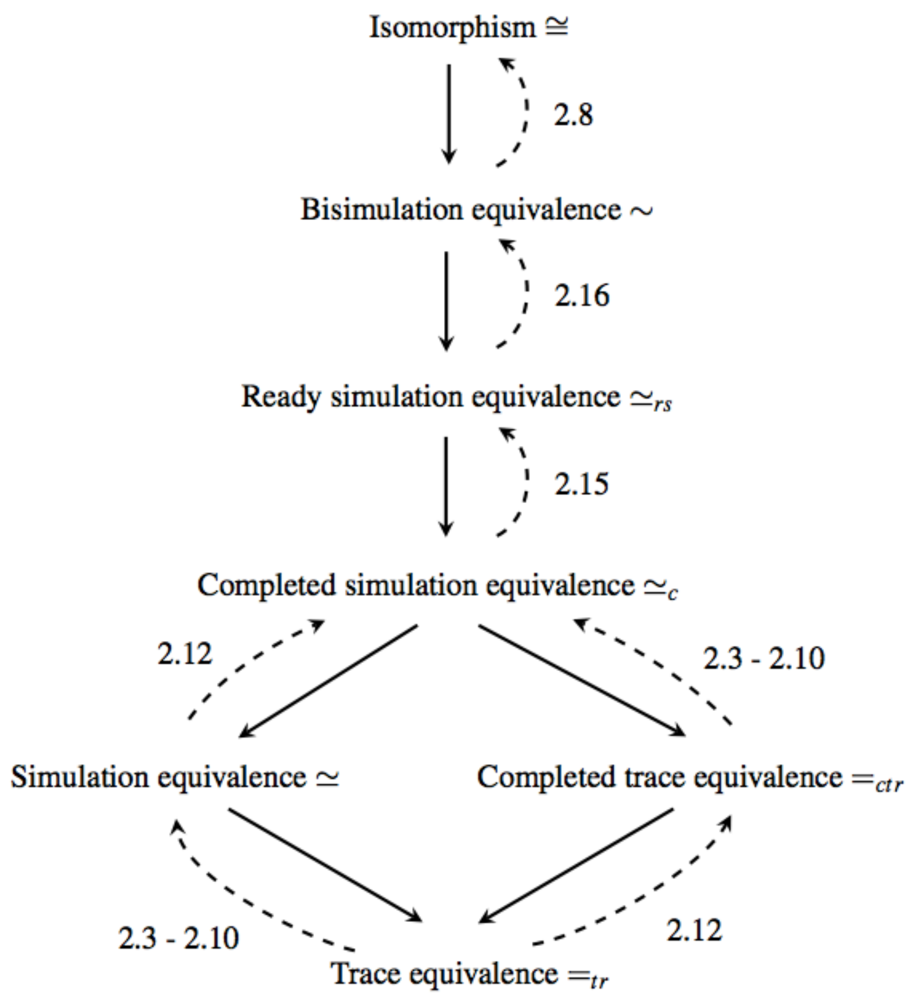
Lezione 4 MSC
Equivalenze forti – prima parte (di 3)

Roberto Gorrieri

Gerarchia di equivalenze forti



Isomorfismo di LTS's

- Dato che gli LTS's sono praticamente grafi …

Definition 2.8. Let $TS_1 = (Q_1, A_1, \rightarrow_1)$ and $TS_2 = (Q_2, A_2, \rightarrow_2)$ be two labeled transition systems. An *isomorphism* is a bijection $f : Q_1 \rightarrow Q_2$ that preserves transitions:

$$q \xrightarrow{\mu}_1 q' \quad \text{if and only if} \quad f(q) \xrightarrow{\mu}_2 f(q')$$

for all $q, q' \in Q_1$ and $\mu \in A_1 \cup A_2$. If there exists an isomorphism between TS_1 and TS_2 then we say that TS_1 and TS_2 are *isomorphic*, denoted $TS_1 \cong TS_2$.

This definition can be lifted to rooted labeled transition systems by requiring that the isomorphism f preserves also the initial states, i.e., $f(q_1) = q_2$, if q_1 and q_2 are the initial states of TS_1 and TS_2, respectively. □

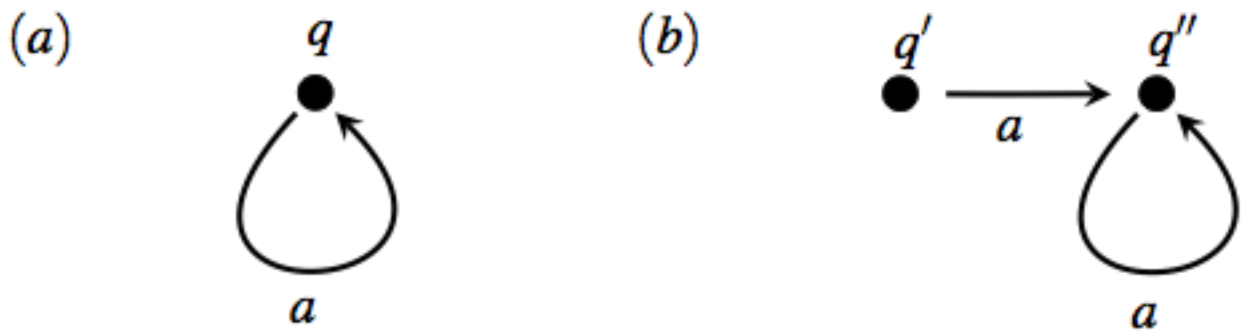
Remark 2.2. Observe that if $TS_1 = (Q_1, A_1, \rightarrow_1)$ and $TS_2 = (Q_2, A_2, \rightarrow_2)$ are isomorphic, then they are also isomorphic to $TS_1' = (Q_1, A, \rightarrow_1)$ and $TS_2' = (Q_2, A, \rightarrow_2)$, where $A = A_1 \cap A_2$. □

Esercizi

- Esercizio: dimostra che l'isomorfismo tra grafi è una relazione d'equivalenza (cioè riflessiva, simmetrica e transitiva).
- Prova che un lts finitely-branching con un numero finito di stati è (isomorfo a) un finite-state lts.

Isomorfismo: troppo concreto

- Distingue lts's che dovrebbero essere eguagliati



- Inoltre è poco pratico: riconoscere se due grafi sono isomorfi è un problema NP (che non si sa se sia in P) e il migliore algoritmo noto è esponenziale nel numero degli stati.

Equivalenza di linguaggio

- Poiché gli LTSs sono praticamente automi, proviamo con l'equivalenza tipica degli automi
- Un automa, oltre ad uno stato iniziale, ha un insieme designato di stati finali; una sequenza di simboli (una stringa) è riconosciuta se c'è un cammino dallo stato iniziale ad uno dei suoi stati finali che "legge" quella stringa.
- Due automi sono equivalenti se riconoscono le stesse stringhe (language equivalence).

Equivalenza a tracce (1)

- Mentre per gli automi verifichiamo se la sequenza σ può portare l'automa in uno stato finale, per gli LTSs verifichiamo se è in grado di eseguire quella sequenza interattivamente.
- Quindi, se un LTS esegue una sequenza σ , necessariamente è anche in grado di eseguire un qualunque prefisso di σ ; questo implica che implicitamente assumiamo che tutti gli stati siano finali.

Equivalenza a tracce (2)

Definition 2.9. (Trace equivalence) Let $(Q, A, \rightarrow)$ be an LTS and let $q \in Q$. A *trace* of q is a sequence of actions $\mu_1 \mu_2 \dots \mu_n$ (possibly empty, when $n = 0$) such that there exists a path

$$q \xrightarrow{\mu_1} q_1 \xrightarrow{\mu_2} \dots q_{n-1} \xrightarrow{\mu_n} q_n.$$

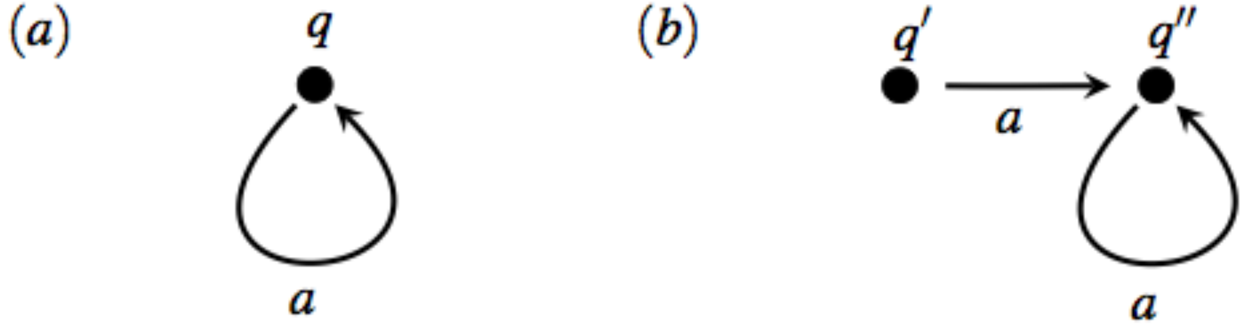
In other words, according to Exercise 2.4, the set of *traces* of q is

$$Tr(q) = \{\sigma \in A^* \mid \exists q' \in Q.\, q \xrightarrow{\sigma}{}^{*} q'\}.$$

Two states $q_1, q_2 \in Q$ are *trace equivalent* if $Tr(q_1) = Tr(q_2)$, and this is denoted as $q_1 =_{tr} q_2$. This definition can be extended to rooted LTSs as follows. The set $Tr(TS)$ of traces of the rooted LTS $TS = (Q, A, \rightarrow, q_0)$ is $Tr(q_0)$. Two rooted LTSs, TS_1 and TS_2, are *trace equivalent* if $Tr(TS_1) = Tr(TS_2)$. $\square$

Esempio

- Tr(q) = {ε, a, aa, aaa, aaaa, ...} = a*
  - esercizio: provarlo per induzione sulla lunghezza della traccia.
- Tr(q’) = Tr(q’’) = Tr(q) = $\{a^n \mid n \in N\}$ = a*



Equivalenza a tracce (3)

Trace preorder: $q_1 \leq_{tr} q_2$ iff $Tr(q_1) \subseteq Tr(q_2)$

Examples: $a.0 \leq_{tr} a.b.0$ $a.0 \leq_{tr} a.0 + b.0$

Trace equivalence: $q_1 =_{tr} q_2$ iff

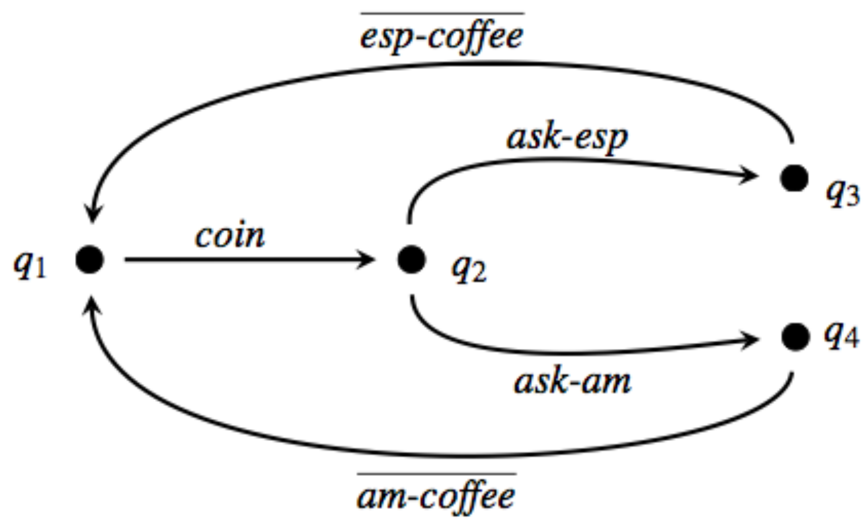
($q_1 \leq_{tr} q_2$ and $q_2 \leq_{tr} q_1$) iff $Tr(q_1) = Tr(q_2)$

Example: $a.b.0+b.a.0 =_{tr} a.0 \mid b.0$

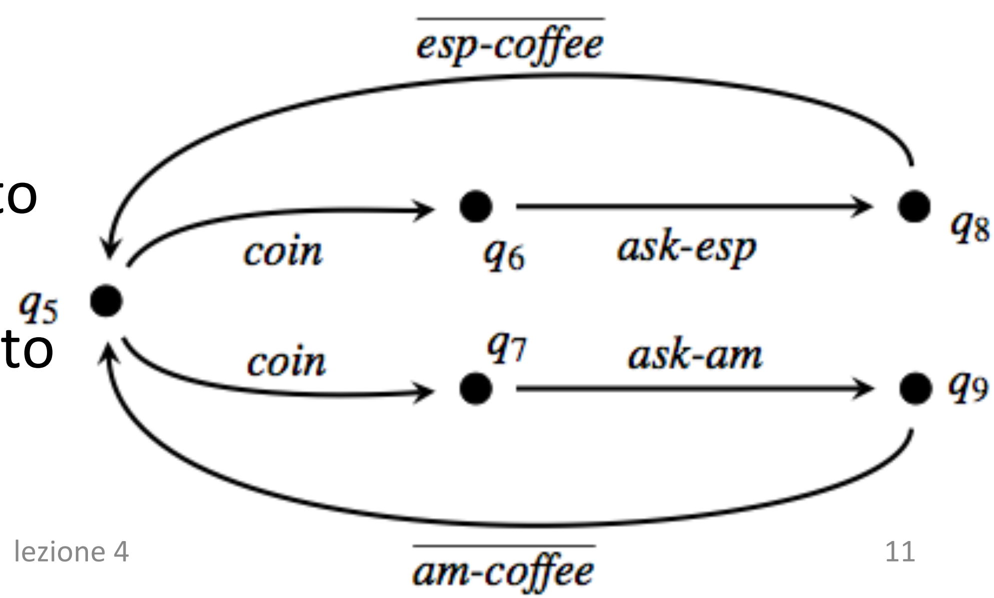
Exercise: Prove that relation $\leq_{tr} \subseteq Q \times Q$ is reflexive and transitive, i.e. the trace preorder is a preorder. Check that $\leq_{tr}$ is not (anti)symmetric.

Exercise: Prove that relation $=_{tr} \subseteq Q \times Q$ is reflexive, symmetric and transitive, i.e. trace equivalence is an equivalence relation.

Trace eq. utile per sistemi reattivi?



$Tr(q_1) = Tr(q_5)$, quindi le due sono trace equivalent.



Punto di scelta diverso: nella prima l'utente dopo aver inserito la moneta, può decidere la bevanda; la seconda, al momento dell'inserzione, decide quale opzione offrire all'utente.

Nondeterminismo irrilevante per trace equivalence (1)

Exercise 2.17. (Nondeterminism vs. Determinism for trace equivalence) Given a nondeterministic, finite-state, rooted lts $TS_1 = (Q, A, \rightarrow_1, q_0)$, one can build a finite-state rooted lts $TS_2 = (R, A, \rightarrow_2, \{q_0\})$, where $R = \wp(Q) \setminus \{\emptyset\}$ is the set of all nonempty subsets of Q and the transition relation $\rightarrow_2 \subseteq R \times A \times R$ is defined as follows: for any $P \in R$ and for any $a \in A$, we have that $P \xrightarrow{a}_2 P'$ if $P' = \{q' \in Q \mid \exists q \in P. q \xrightarrow{a}_1 q'\}$ is nonempty.[3] Prove that TS_2 is deterministic and that the two lts's are trace equivalent, i.e., $Tr(TS_1) = Tr(TS_2)$. Apply the construction to the nondeterministic lts of Figure 2.10, rooted in q_5, and compare the result with the deterministic lts in Figure 2.3.

---

[3] This simple construction is not optimal, as it may generate states that are unreachable from the initial state $\{q_0\}$. It is an easy exercise (try it!) to optimize it in order to get only reachable states.

Nondeterminismo irrilevante per trace equivalence (2)

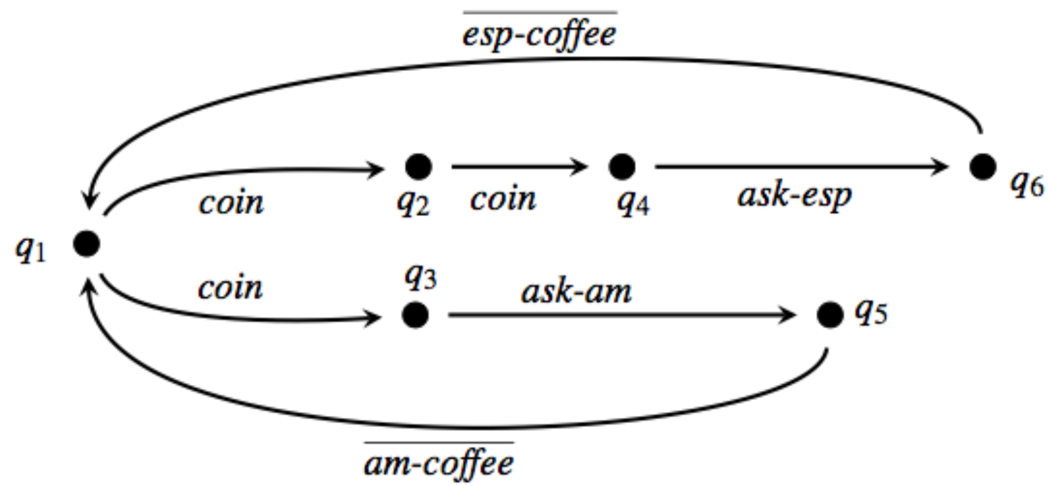
This construction is a slight variant of the famous *Rabin-Scott subset construction* for nondeterministic finite-state automata [RS59] (see, e.g., [Sip06, HMU01] for a gentle introduction). In principle, it may be extended also to lts's with infinitely many states, but the resulting deterministic lts's may have uncountably many states (which is excluded in Definition 2.2), as the powerset of a countable set may be uncountable.

This exercise proves that, from an expressiveness point of view, nondeterminism is inessential for trace equivalence, as, given a nondeterministic lts, it is always possible to find a deterministic one exhibiting the same set of traces. (Observe that, on the contrary, it is not possible to find a deterministic lts isomorphic to a nondeterministic lts.) □

- Esercizio: considera a.b.0 + a.c.0 e rendilo deterministico

Un’altra vending machine

- Supponiamo che servano due monete per l’espresso, mentre una sola per am-coffee
- Se l’utente ha una sola moneta, cosa può capitare? Deadlock (dell’utente)!



- Esercizio: (1) Modellare una vending machine corretta rispetto ai punti di scelta. (2) Modificarla in modo che possa avere am-coffe anche dopo che 2 monete sono state inserite, mantenendone una in credito. (3) Questi due modelli sono trace equivalent?

Trace eq. insensibile al deadlock

- Uno stato q è un deadlock se da q non parte nessuna transizione. Ad esempio, q_3, q_4 e q_7 qui sotto.
- $Tr(q_1) = Tr(q_5) = \{\varepsilon, a, ab\}$ ma il secondo dopo aver fatto *a*, non va mai in deadlock.

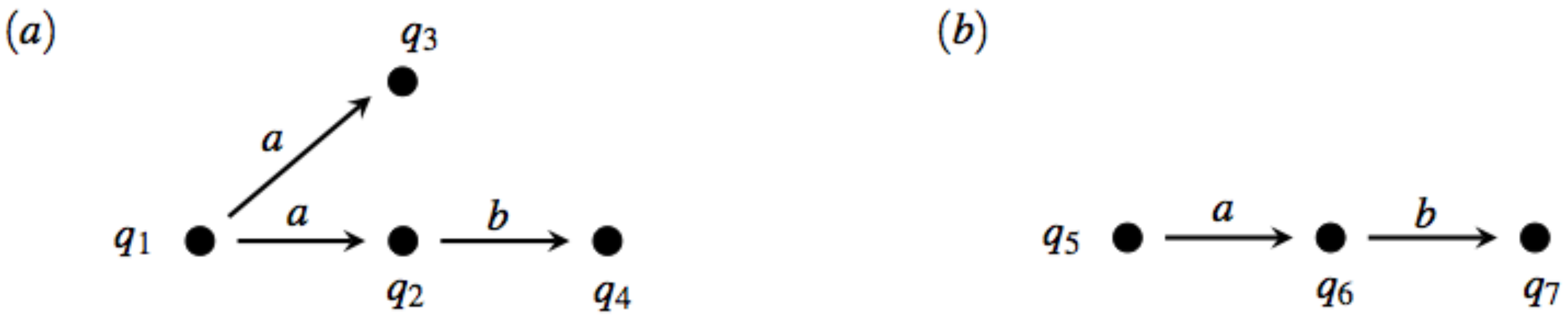


Fig. 2.12 Deadlock is not observable

Completed trace equivalence

- Se osserviamo le tracce massimali (dette complete), allora possiamo osservare il deadlock e distinguere i due sistemi del lucido precedente.

Definition 2.11. (Completed trace equivalence) Let $TS = (Q, A, \rightarrow)$ be a transition system. A *completed trace* for state $q \in Q$ is a (possibly empty) sequence of actions $\mu_1 \dots \mu_n$ such that there exists a path $q_1 \xrightarrow{\mu_1} \dots q_n \xrightarrow{\mu_n} q_{n+1}$ such that $q_1 = q$ and q_{n+1} is a deadlock: $q_{n+1} \not\rightarrow$. Hence, the set of *completed traces* of a state $q \in Q$ is

$$CTr(q) = \{\sigma \in A^* \mid \exists q' \in Q.\, q \xrightarrow{\sigma}{}^* q' \wedge q' \not\rightarrow\}.$$

Two states $q_1, q_2 \in Q$ are completed trace equivalent if $Tr(q_1) = Tr(q_2)$ and $CTr(q_1) = CTr(q_2)$, and this is denoted as $q_1 =_{ctr} q_2$. □

- $CTr(q_1) = \{a, ab\}$ $CTr(q_5) = \{ab\}$

Esercizio

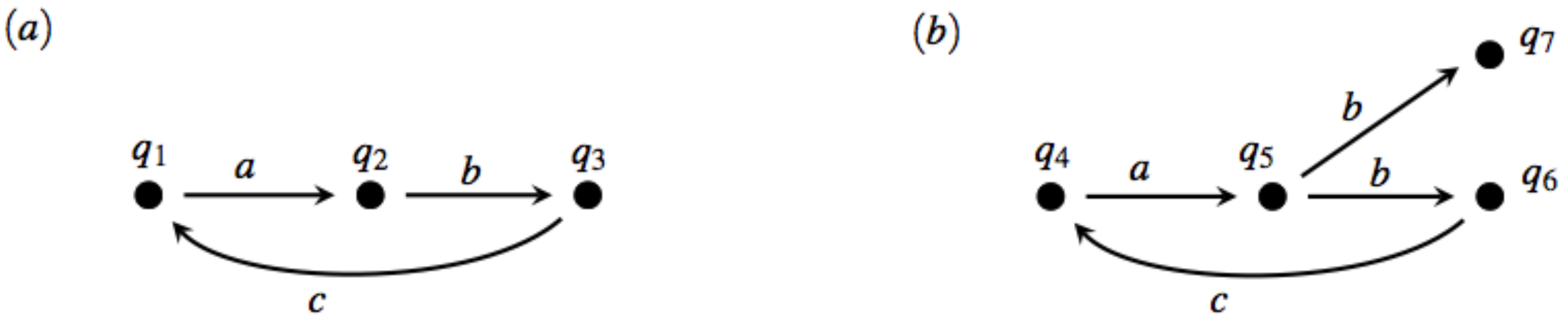
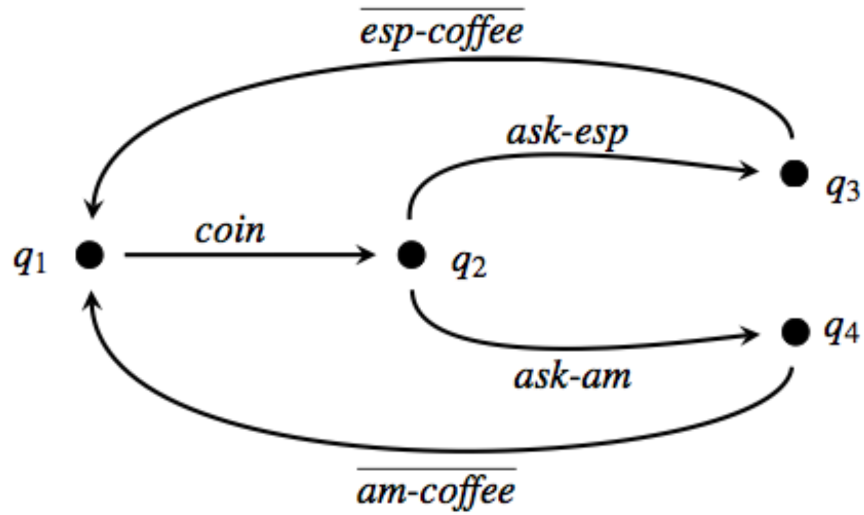


Fig. 2.14 Two not completed trace equivalent systems

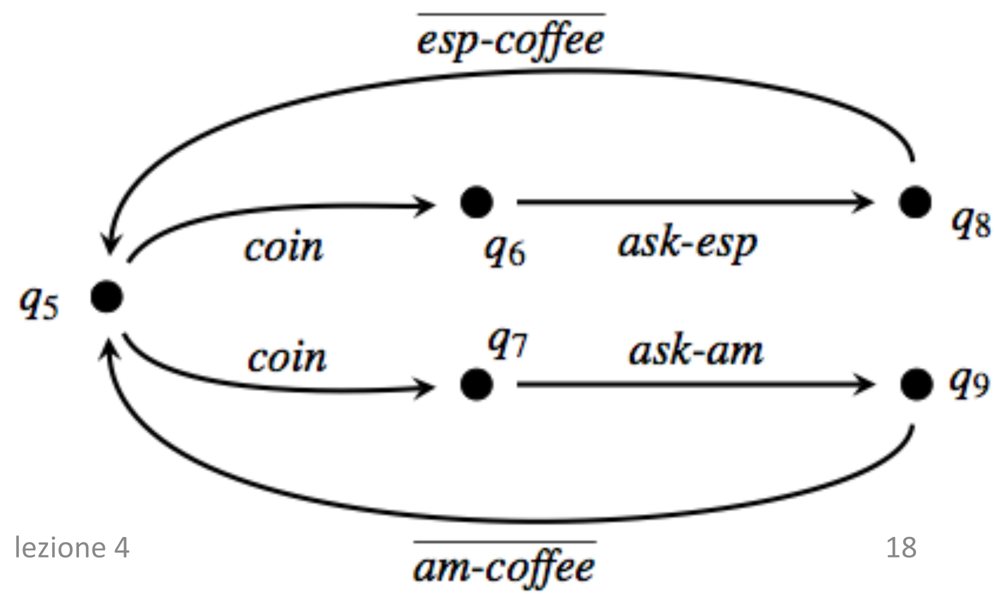
- Quali sono le tracce complete di questi due?
- Esistono due rooted lts tali che abbiano le stesse tracce complete, ma non le stesse tracce?

È completed Trace eq. sufficiente?



$Tr(q_1) = Tr(q_5)$ & $CTr(q_1) = CTr(q_5) = \emptyset$, quindi le due sono completed trace equivalent. Allora serve un'equivalenza più fine.

Complessità: PSPACE complete, esattamente come language equivalence su automi. Allora serve un'equivalenza più facilmente verificabile.



Serve a qualcosa la trace equiv?

- trace equivalence/preorder è utile per la verifica di safety properties.
- Una safety property è una proprietà che afferma che "something bad never happens". Se TS1 (specifica) soddisfa un safety property e se TS2 (implementazione) è tale che Tr(TS2) è incluso (preorder) in Tr(TS1), allora anche l'implementazione TS2 soddisfa quella safety property. (Ad es: "mai un coffe se non prima coin" per vending machine.)
- Al contrario, trace equivalence non può essere usata per verificare liveness properties, che richiedono qualche progresso: "something good will happen eventually". In esempio di lucido 15, la liveness property è "action b will happen eventually" che è soddisfatta da q_5 ma non da q_1.

Serve a qualcosa la completed trace equivalence?

- Completed trace equivalence è la semantica più astratta che osserva il deadlock (totale).
- Ma completed trace equivalence non è una congruenza per la restrizione del CCS
- P = a.(b.0 + c.0) e Q= a.b.0 +a.c.0 sono completed trace equivalent, ma (vc)P e (vc)Q non sono completed trace equivalent
- Trace equivalence è invece una congruenza per gli operatori del CCS (lo vedremo)