Diagrammi di stato
State Diagram

a cura di **Angelo Furfaro**
da “UML Distilled”
Martin Fowler

Dipartimento di Ingegneria Informatica Elettronica Modellistica e Sistemistica
Università della Calabria, 87036 Rende(CS) - Italy
Email: a.furfaro@dimes.unical.it
Web: http://dev.dimes.unical.it/~furfaro

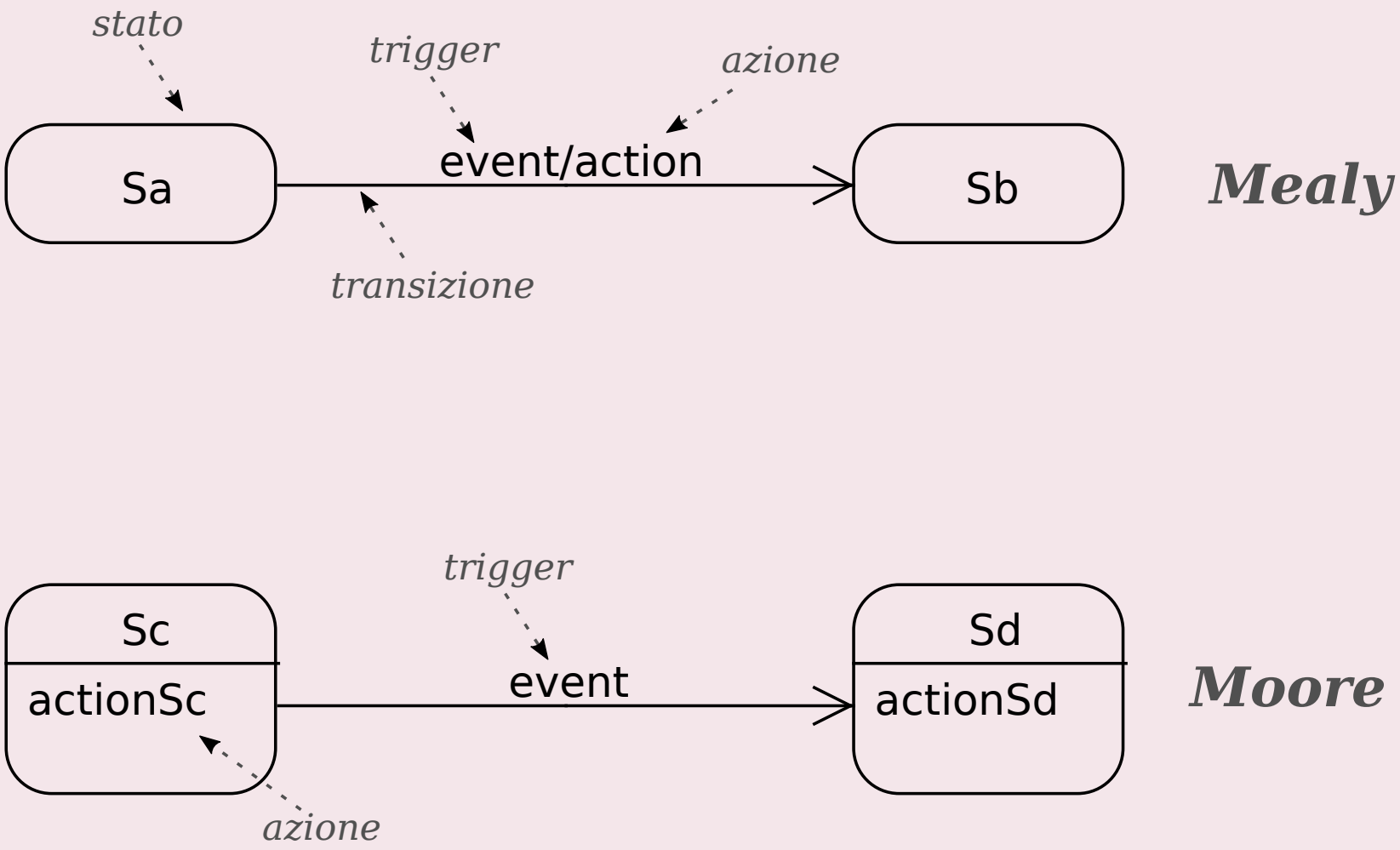
Diagrammi di Stato

- Molto spesso il comportamento di un oggetto, ossia il modo con cui esso risponde ai messaggi che riceve (chiamate ai metodi), cambia dinamicamente.
- La risposta dell'oggetto dipende dal suo *passato*, dalla sua *storia*.
- In questi casi si parla di oggetti provvisti di *stati di controllo*, per esprimere il fatto che a seconda del particolare stato di controllo posseduto sussiste una certa modalità di risposta ai messaggi.
- Lo stato di un oggetto modella la sua *storia*. È la conseguenza delle azioni eseguite in risposta ai messaggi ricevuti in precedenza.
- Si dice anche che un oggetto di possiede al suo interno un automa a stati finiti. La modellazione del comportamento dell'oggetto dunque si accompagna anche alla modellazione del suo automa (o macchina a stati finiti).
- Negli approcci object-oriented un automa a stati finiti è associato ad una classe e ne modella il comportamento degli oggetti che sono istanze di essa.

Automi (o Macchine) a stati finiti

- Un automa a stati finiti (FSM – Finite State Machine) è un grafo orientato in cui i nodi sono gli stati possibili e gli archi corrispondono alle possibili transizioni di stato.
- Uno stato particolare è lo **stato iniziale**, in cui l'oggetto-automa viene a trovarsi ad es. subito dopo la creazione. Lo stato iniziale è specificato mediante una **pseudo-transizione** che emana da un cerchietto nero (**pseudo-stato**).
- Uno o più stati *possono* fungere da **stati finali** o terminatori: l'automa raggiunto uno di questi stati non ammette più evoluzione.
- Una transizione di stato è causata dall'arrivo di un evento (o messaggio) detto **trigger** della transizione.
- Un automa risponde ad un evento mediante una **transizione di stato** e un'**azione**. L'azione è **atomica** o non interrompibile.
- Si parla di **automa di Moore** quando l'azione è agganciata allo stato, di **automa di Mealy** se l'azione è associata all'arco-transizione. Nel primo caso, quale che sia la transizione che porta ad uno stato, si ha sempre l'esecuzione dell'azione corrispondente allo stato. Nel secondo, ogni possibile transizione può specificare una diversa azione.
- I formalismi di Moore e di Mealy sono equivalenti: è sempre possibile trasformare un FSM di Moore in uno di Mealy e viceversa, anche se il numero degli stati non è necessariamente lo stesso nelle due versioni.

Mealy e Moore

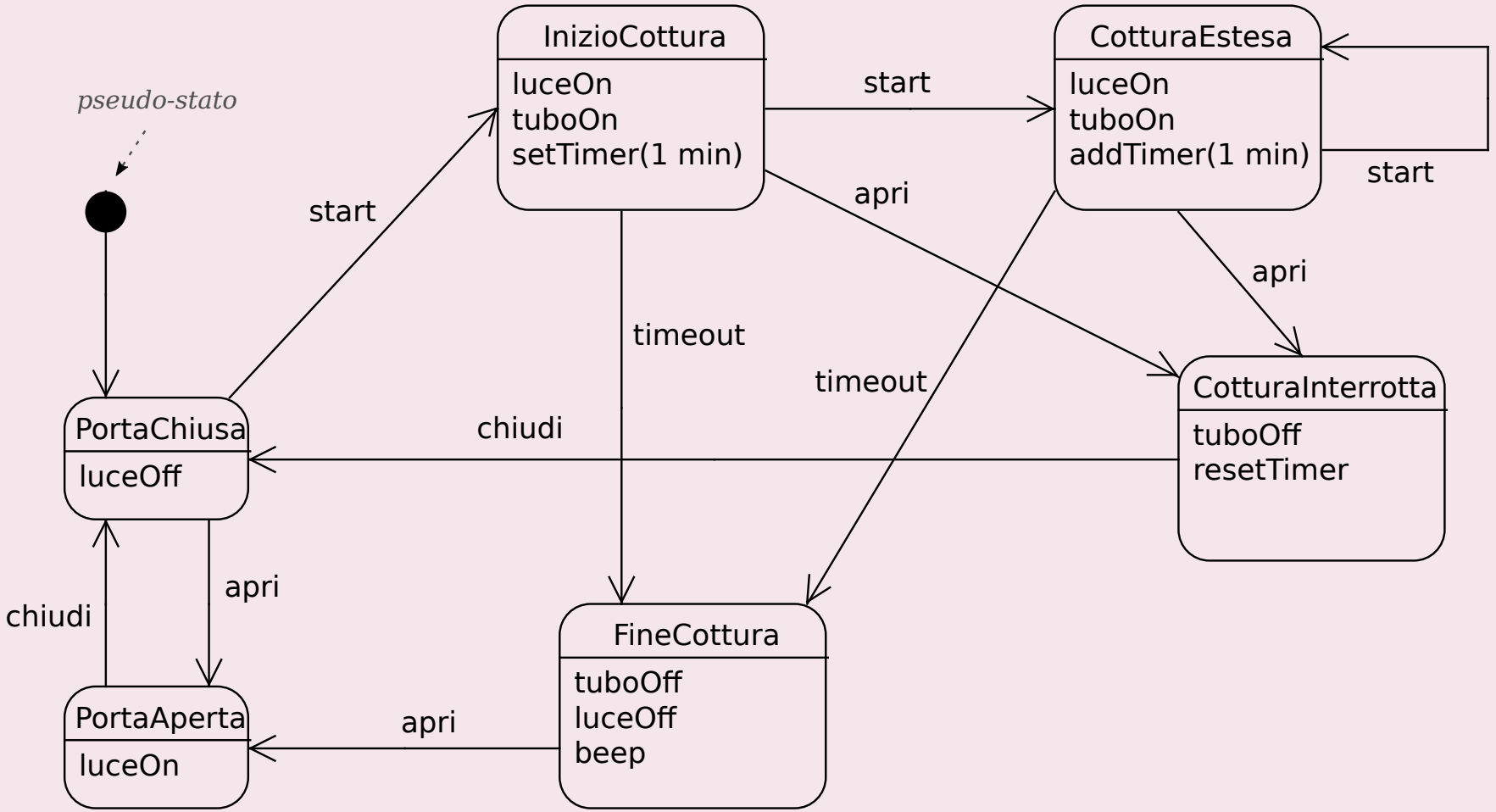


Esempio

Forno a microonde

- Si considera un oggetto-forno che risponde ai seguenti eventi (associati ad es. a pulsanti sul frontalino dell'apparecchiatura):
  - **apri** (apre la porta)
  - **chiudi** (chiude la porta)
  - **start** (attiva il tubo a microonde)
- Il forno è composto, oltre che dal tubo, da una luce e da un timer. L'apertura della porta fa accendere la luce. La chiusura della porta spegne la luce.
- L'attivazione della cottura (evento start) può avvenire solo a porta chiusa, setta il timer ad 1 min e accende anche la luce. Allo scadere del timer, viene emesso un beep ed il tubo si spegne automaticamente. E' possibile aumentare il tempo di cottura re-inviando (anche più volte prima dello spiro del timer) l'evento start che comporta ogni volta l'aggiunta di un ulteriore minuto al tempo di spiro
- È possibile interrompere la cottura aprendo bruscamente la porta, nel qualcaso il timer è resettato ed il tubo disattivato.

Automa Forno



Statechart

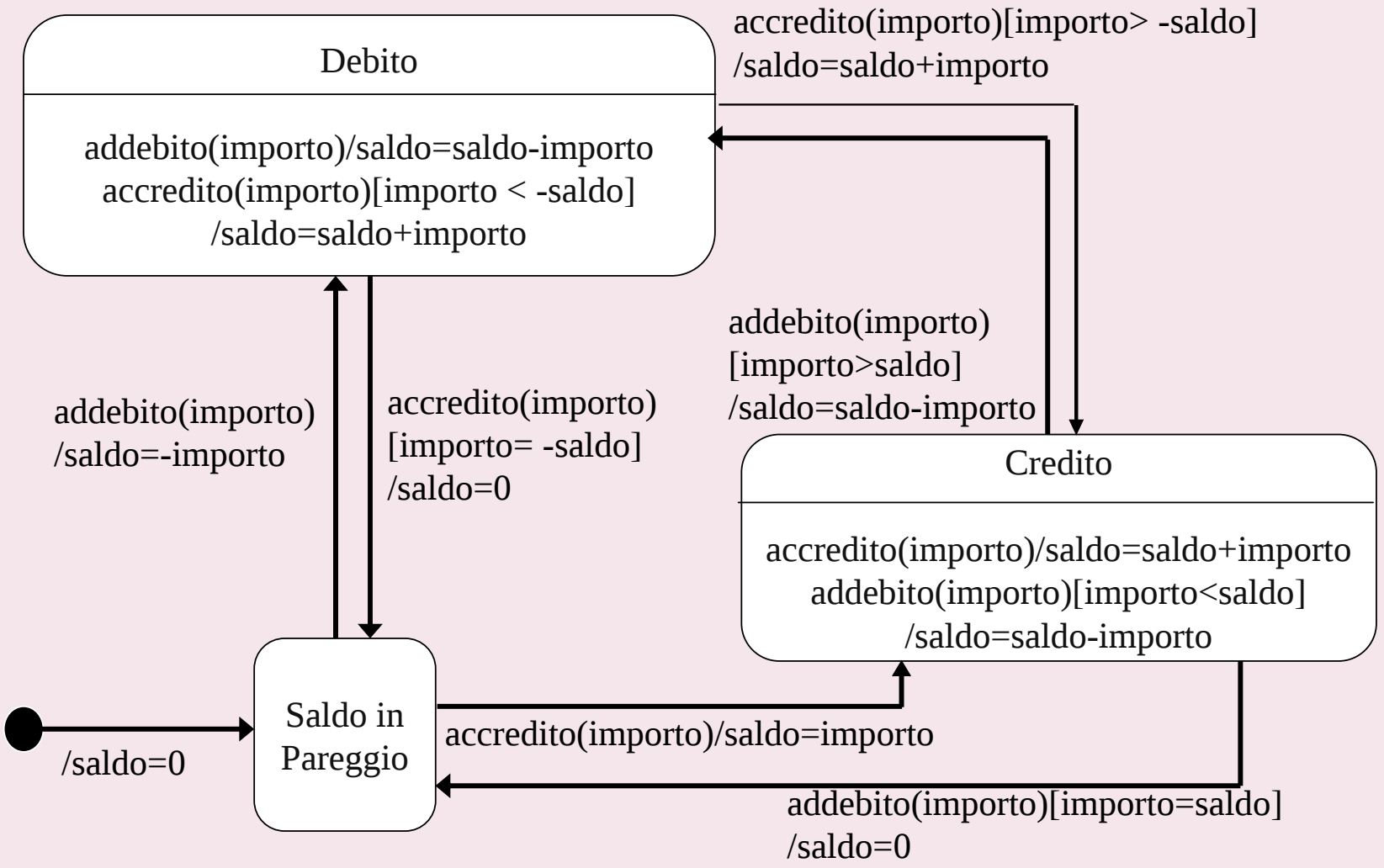
I diagrammi di stato di UML si basano sugli statechart, originariamente inventati da David Harel e disponibili, tra l'altro, in STATEMATE Incorporano una serie di estensioni agli automi "piatti" per modellare situazioni complesse, tra cui:

- Stati gerarchici o annidati (superstati e sottostati)
- Transizioni di gruppo
- Connettori di storia
- Eventi con guardie
- Azioni di entrata (entry), uscita (exit), attività interne (do), e risposta ad eventi, in uno stato
- Transizioni di stato indifferentemente di Mealy o di Moore
- Transizioni spontanee

Stato di uno statechart

Semantica

- Entrando nello stato vengono eseguite (se esistono) le entry action e fatta partire (se esiste) l'attività interna
- Uscendo dallo stato vengono eseguite (se esistono) le exit action e fatta terminare (se esiste ed è attiva) l'attività interna


Debito
addebito(importo)/saldo=saldo-importo
accredito(importo)[importo < -saldo]
/saldo=saldo+importo
accredito(importo)[importo> -saldo]
/saldo=saldo+importo
addebito(importo)
[importo>saldo]
/saldo=saldo-importo
addebito(importo)
/saldo=-importo
accredito(importo)
[importo= -saldo]
/saldo=0
Credito
accredito(importo)/saldo=saldo+importo
addebito(importo)[importo<saldo]
/saldo=saldo-importo
Saldo in
Pareggio
/saldo=0
accredito(importo)/saldo=importo
addebito(importo)[importo=saldo]
/saldo=0

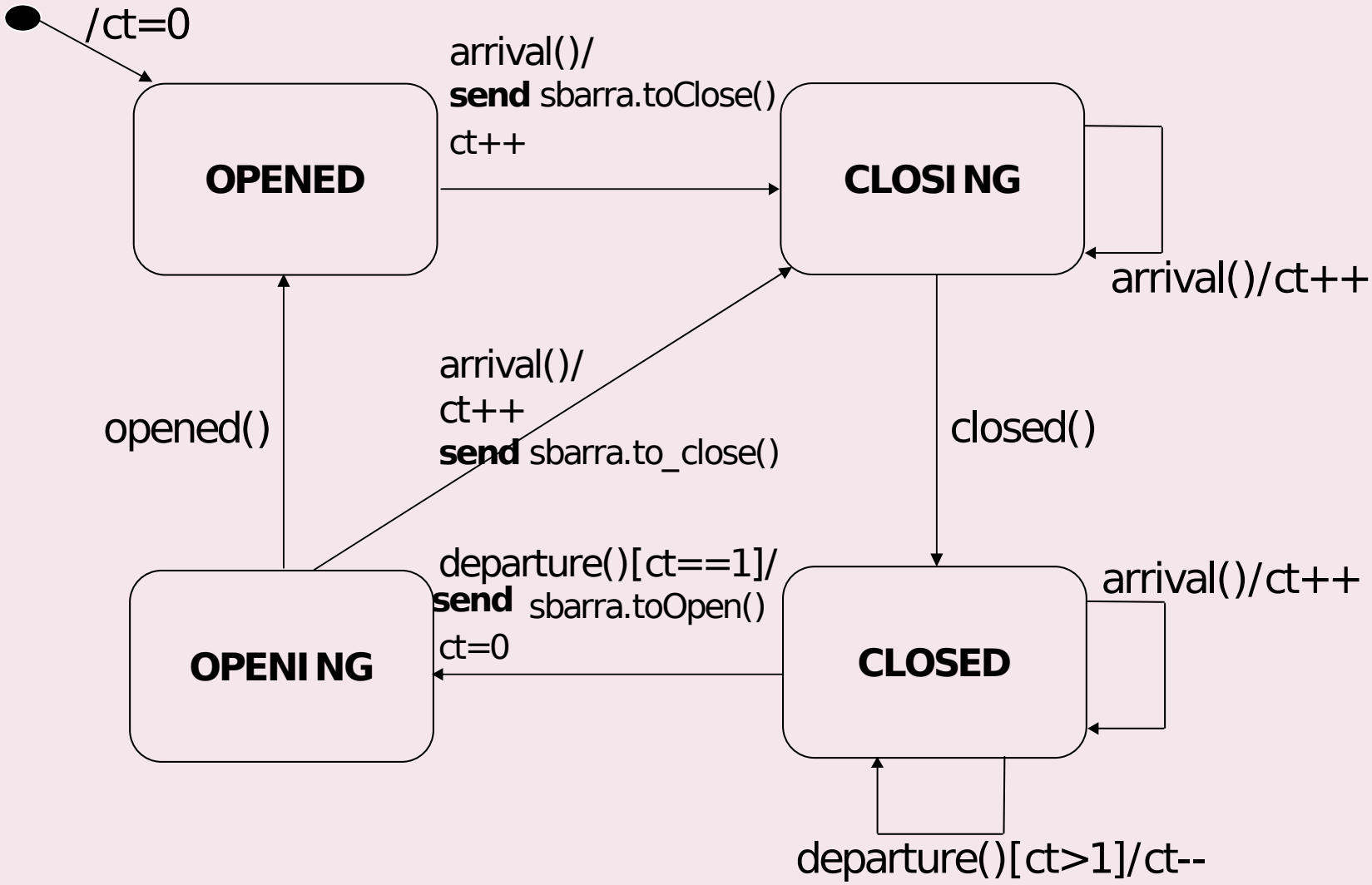
Eventi e guardie

- Trigger di una transizione:
  `evento(parametri) [guardia] / azione`
  la condizione può mancare (`true` default)
- Invio di un evento:
  `send target.evento(parametri)`
- Le guardie sono racchiuse entro `[` e `]` e si basano su dati dell'oggetto. Più in generale possono basarsi anche sui valori dei parametri di un evento ricevuto
- Lo statechart della classe Conto mette in evidenza come alcuni eventi possano essere ricevuti e processati in uno stesso stato, senza cioè far cambiare stato
- Esiste una "sottile" differenza semantica tra eventi ricevuti e processati "dentro" uno stato ed eventi che etichettano archi di auto-anello dello stato:
  - I primi non si accompagnano alle azioni di entry ed exit dello stato.
  - Nel secondo caso, ad ogni uscita dallo stato vengono eseguite, se esistono, le exit action; ad ogni entrata si eseguono, se esistono, le entry action.

Esempio: Controller di un passaggio a livello

- Si considera un incrocio tra alcuni binari ferroviari ed una strada attraversata da automobili. Si suppone che i treni in transito si muovano ad es. solo da sinistra a destra e consistano del solo locomotore. L'incrocio potrebbe rappresentare un sistema reale nei pressi di una stazione ferroviaria
- L'incrocio è gestito da una sbarra e da due sensori (arrivo e partenza). L'arrivo di un treno nella zona dell'incrocio è segnalato dal primo sensore che invia al controller un messaggio `arrival()`. Quando un treno si allontana dall'incrocio, il secondo sensore invia un messaggio `departure()`. Tali messaggi consentono al sistema di comandare opportunamente alla sbarra di alzarsi o abbassarsi.
- Poiché l'apertura/chiusura della sbarra non sono attività istantanee, la sbarra provvede altresì a spedire al controller il messaggio `closed()`/`opened()` quando rispettivamente si è abbassata/alzata completamente.
- Il tempo richiesto per alzarsi/abbassarsi sono conosciuti solo dalla sbarra e possono essere diversi rispettivamente nei due movimenti. Si ammette inoltre che il tempo impiegato da un treno per raggiungere l'incrocio dal momento della segnalazione del suo arrivo, sia molto più grande di quello richiesto per l'azionamento completo della sbarra
- L'obiettivo è modellare il comportamento del controller.

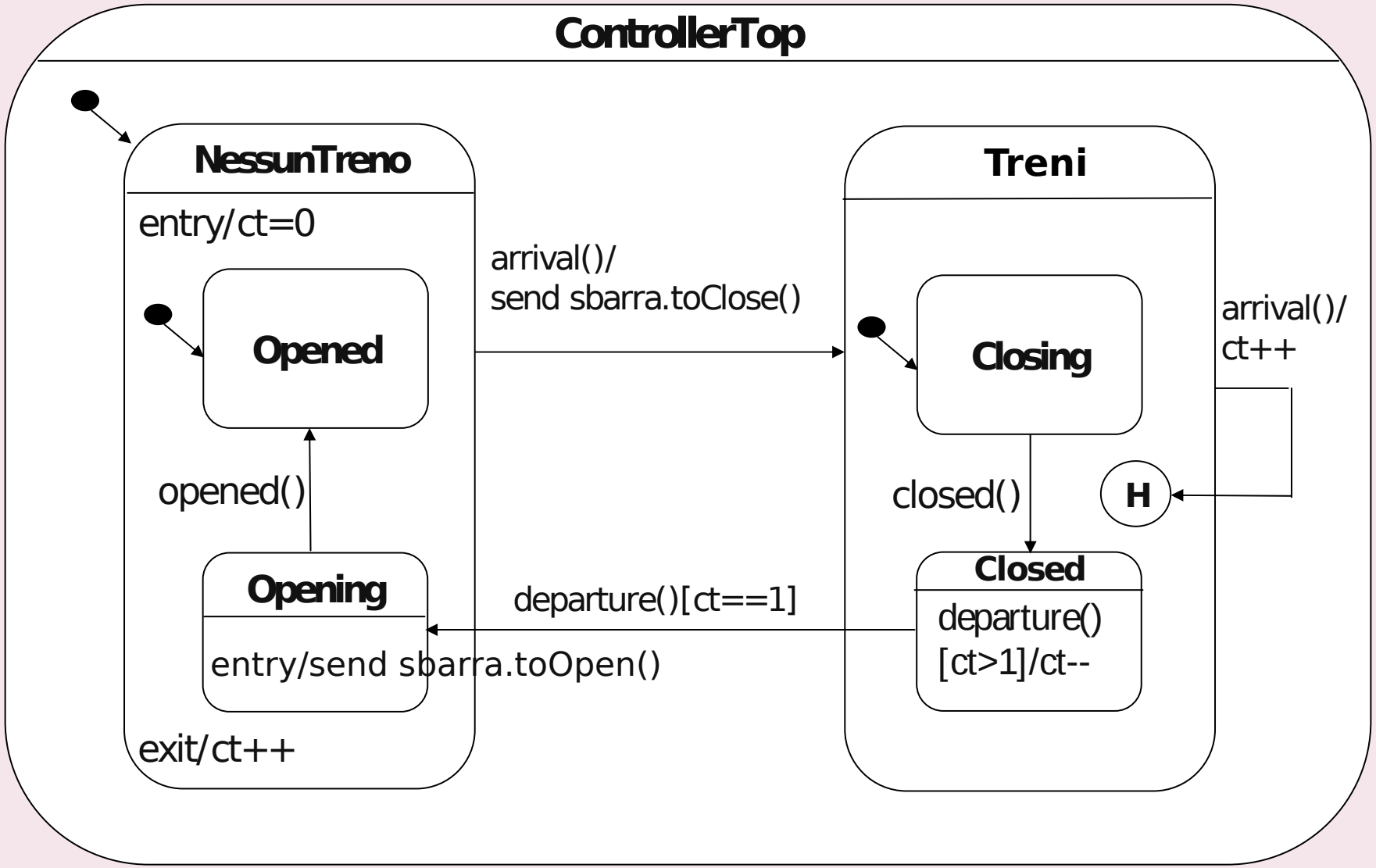
Uno statechart *piatto* per il Controller



Osservazioni

- L'attributo `ct` dell'oggetto Controller, conta il numero dei treni in transito
- Solo quando `ct` va a zero dopo un `departure()` è possibile comandare l'apertura della sbarra
- Il controller invia alla sbarra i messaggi `toClose()` e `toOpen()` il cui significato è ovvio
- Nella diapositiva che segue si riporta l'automa del controller come statechart UML gerarchico, col solo obiettivo di mostrare un primo uso dei meccanismi disponibili

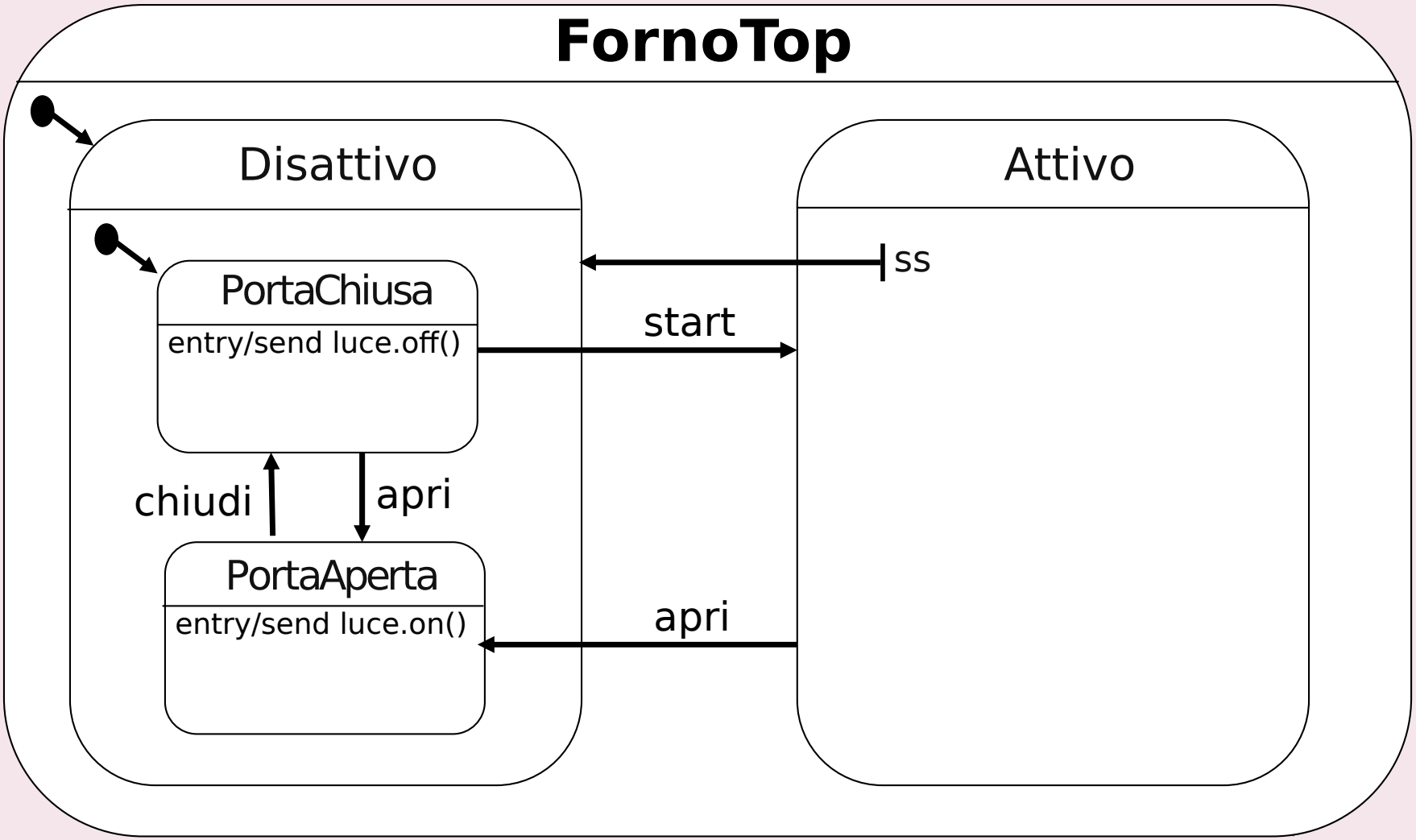
Il Controller come statechart gerarchico



Commenti

- Il macro stato **ControllerTop** ammette due sotto stati **NessunTreno** e **Treni** ciascuno dei quali a sua volta è decomposto in due sotto stati
- Ogni qualvolta si entra in **NessunTreno** (stato di default o iniziale di **ControllerTop**), viene eseguita la entry action: ct=0, che azzera il contatore. Similmente, quando **NessunTreno** è abbandonato, si esegue la exit action: ct++
- La transizione con trigger arrival() che fuoriesce dal bordo di **NessunTreno** è una transizione di gruppo: quale che sia lo stato interno di **NessunTreno**, ci si porta in **Treni** e si comanda alla sbarra di chiudersi
- Il sotto stato **Closed** di **Treni** ha un evento interno departure() con la guardia [ct$>$1] che fa rimanere in **Closed** e decrementa ct
- Anche lo stato **Treni** ammette una transizione di gruppo con trigger arrival() che comporta l'incremento di ct e, dato l'uso del connettore di storia **H** (si veda nel seguito per altre informazioni al riguardo), fa riassumere il sotto stato di **Treni** che era corrente al tempo di arrival() (dunque **Closing** o **Closed**)
- Lo statechart può essere semplificato aggiungendo alla entry action di **NessunTreno** l'azione send sbarra.toOpen() e alla sua exit action l'azione send sbarra.toClose(). Inoltre è possibile evitare il connettore di storia rendendo arrival() un evento interno di **Treni**, con annessa azione ct++

Uno statechart per il Forno



Superstati e sottostati

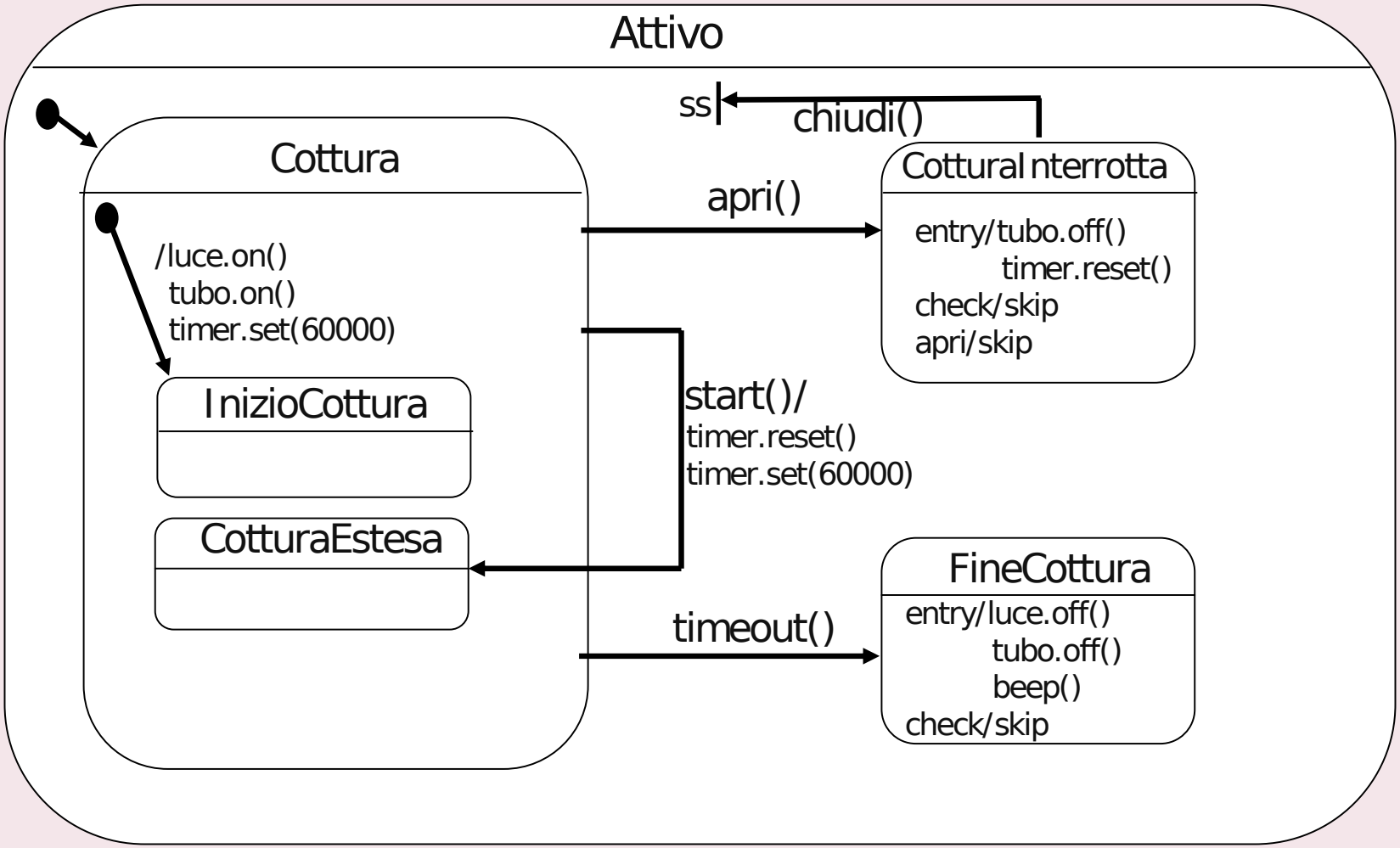
- **FornoTop** è un super (o macro) stato in quanto ammette decomposizione
- **Disattivo** è un substato di **FornoTop** ed a sua volta è macro stato
- **PortaChiusa** e **PortaAperta** sono stati foglia di **Disattivo**
- **Attivo** è un altro macro stato la cui struttura interna è mostrata nella slide che segue
- La transizione apri che parte dal bordo di **Attivo** e rientra in **Disattivo** è una transizione di gruppo: quale che sia lo stato interno di **Attivo**, l'arrivo di apri comporta sempre il raggiungimento dello stato **PortaAperta** di **Disattivo** con le azioni conseguenti
- La transizione start che parte da **PortaChiusa** di **Disattivo** e finisce sul bordo di **Attivo** indica che viene installato lo stato di default (o iniziale) interno ad **Attivo**
- Lo stub state ss indica l'esistenza di uno stato interno in **Attivo**, non ancora specificato, da cui fuorisce una transizione che conduce sul bordo di **Disattivo** (e dunque in **PortaChiusa**). Il trigger di questa transizione sarà specificato nel raffinamento di **Attivo**. Gli stub state (barrette) denotano equivalentemente *spezzoni di transizioni* la cui esatta composizione è materia di sviluppo incrementale dello statechart

Concetto di configurazione

- L'esistenza di stati annidati fa sì che un automa possa trovarsi contemporaneamente in più stati
- Esempio: lo stato corrente di **FornoTop** potrebbe essere espresso dalla configurazione:

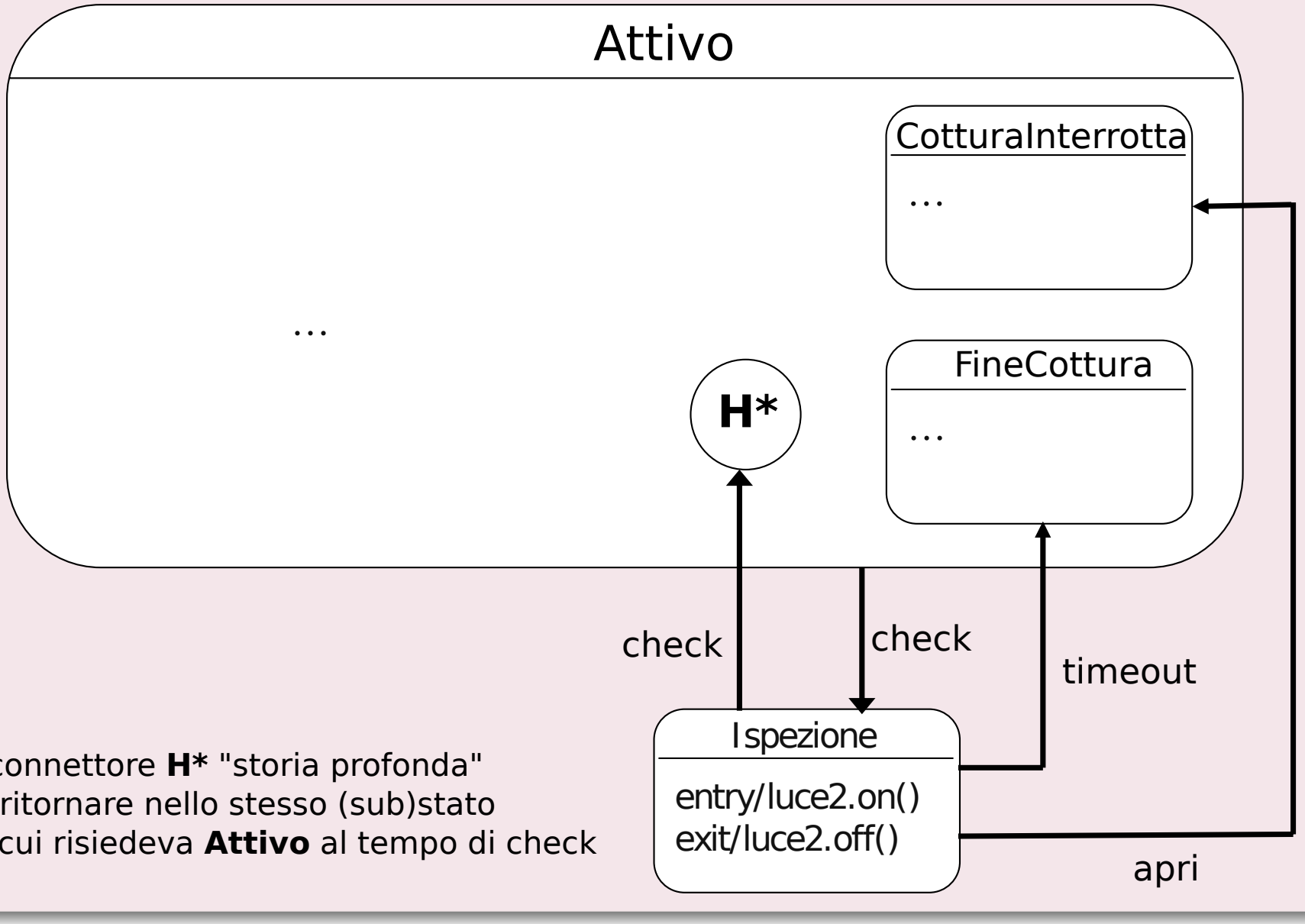
  FornoTop.Disattivo.PortaChiusa
- In altre parole, se il forno si trova in **PortaChiusa**, si trova anche in **Disattivo** e in **FornoTop** (top state)
- Il macro stato **FornoTop** segue la semantica della *or*–decomposition. Di momento in momento, lo stato corrente del forno o è **Disattivo** (unitamente ad un suo stato interno) o è **Attivo** (unitamente ad un suo stato interno).
- Per gli stati il principio dell'*or*-esclusivo (xor): pur valendo il concetto di configurazione, in ogni istante di in *uno* ed uno solo dei suoi sottostati possibili

Il macro stato Attivo



apri, start e timeout sono esempi di transizioni di gruppo per il sottostato **Cottura**. Quando il controllo è in **Cottura**, apri sposta sempre in **CotturaInterrotta** altrimenti conduce in **PortaAperta** di **Disattivo**

Aggiunta ispezione nello stato Attivo



Il connettore **H*** "storia profonda"
fa ritornare nello stesso (sub)stato
in cui risiedeva **Attivo** al tempo di check

Stato Ispezione

- È uno stato foglia di FornoTop e si raggiunge quando FornoTop è nello stato Cottura di Attivo e l'utente invia l'evento check. Si nota che check è ridefinito in FineCottura e CotturaInterrotta in modo da essere trascurato (skip)
- Ripremendo check a partire dallo stato Ispezione, si riassume in Attivo lo stesso (sub)stato ed al suo interno il suo (sub)stato etc ricorsivamente sino ad uno stato foglia, in cui Attivo si trovava al tempo di check
- Poiché l'ispezione potrebbe avvenire quando il timeout è imminente e la persona, proprio a seguito dell'ispezione, potrebbe decidere di aprire bruscamente il forno, dallo stato Ispezione sono state considerate esplicitamente le transizioni conseguenti a timeout e ad apri
- In previsione della transizione legata al connettore di storia, non sono state introdotte entry action negli stati InizioCottura e CotturaEstesa. Piuttosto, le azioni di gestione della luce, del tubo e del timer sono state "agganciate", à la Mealy, all'arco della transizione di default per InizioCottura e all'evento start per quanto riguarda CotturaEstesa. Tutto ciò per impedire che rientrando per storia in uno stato foglia di Cottura venga rieseguita (erroneamente) la corrispondente entry action

Stato Ispezione

Questioni semantiche:

- si è ammesso che abbandonando temporaneamente lo stato Attivo, il firing del timer (se esistente) non si interrompa
- la ridefinizione interna ad Attivo della transizione apri da Cottura a CotturaInterrotta, fa sì che quest'ultima (la più recente) sia più prioritaria della omonima transizione apri che da Attivo porta in PortaAperta di Disattivo. Dunque: se ci si trova in Attivo ed in Cottura, l'effetto di apri resta interno ad Attivo.

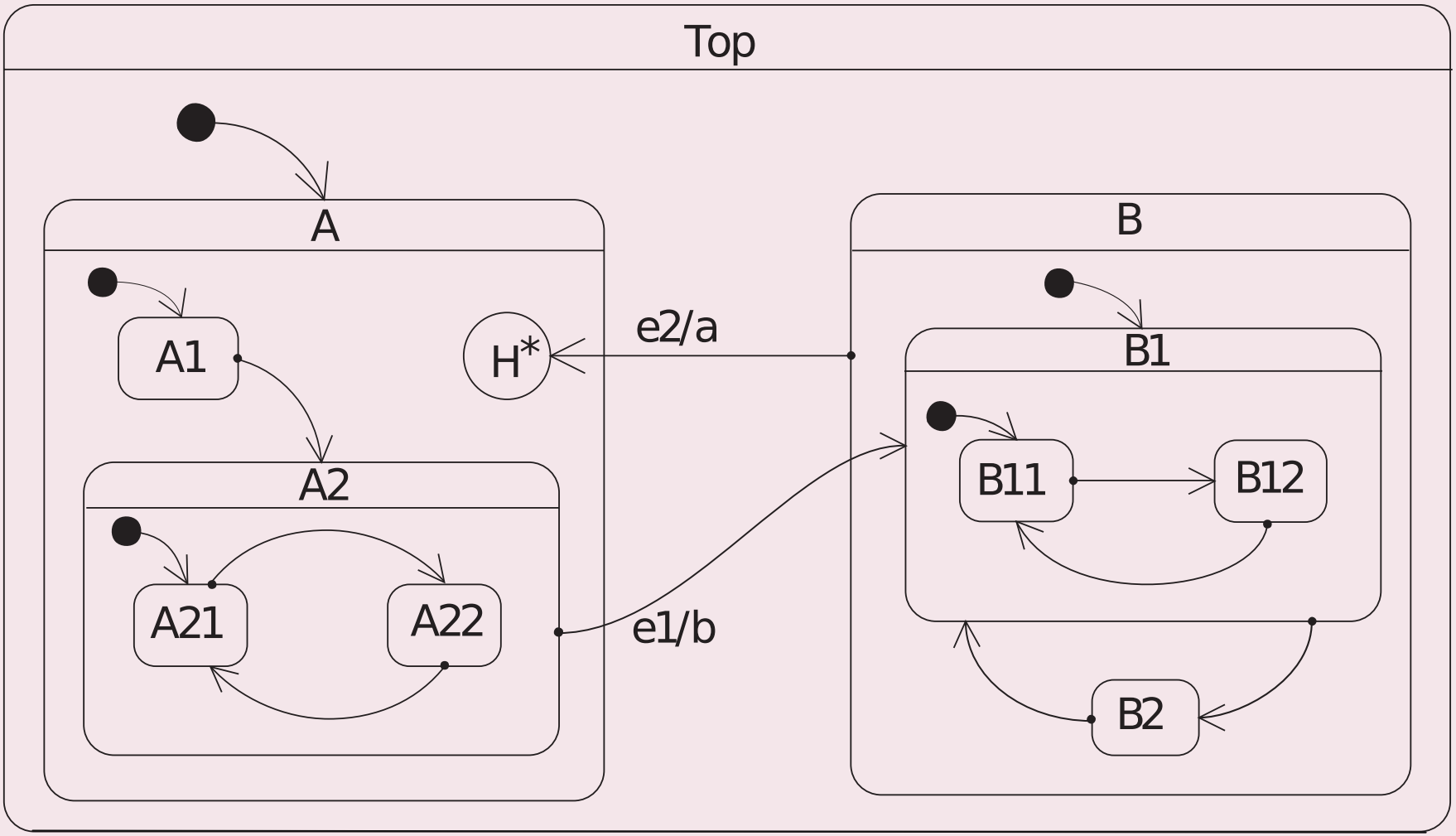
Connettori di Storia H* e H

- Il connettore **H*** indica storia profonda, **H** storia superficiale
- **H***, come si è detto, fa rientrare nello stato foglia corrente esistente al tempo di ricezione di check
- L'uso di **H**, se lo stato corrente era Cottura, avrebbe comportato il ripristino solo del livello di Cottura (e implicitamente del suo stato di default)
- In generale, il connettore **H*** o **H** dovrebbe avere anche una transizione uscente verso un sottostato del macrostato in cui il connettore compare, che verrà scelta nel caso l'ingresso con storia non trovi alcuna memoria (storia) di stati presente nel macrostato. In sua assenza ed in mancanza di memoria (es. in quanto non si è precedentemente entrati nel macrostato) si raggiunge lo stato di default

Altre osservazioni

- L'evento start fa passare dallo stato Disattivo (se è corrente in esso il substato PortaChiusa) allo stato Attivo, ed immediatamente nel substato Cottura e quindi nello stato foglia InizioCottura
- Poiché start innesca l'inizio cottura, in cui la luce si deve necessariamente accendere, si potrebbe equivalentemente porre l'azione luce.on() come exit action dello stato Disattivo, togliendola dalle entry action di InizioCottura

Esempio



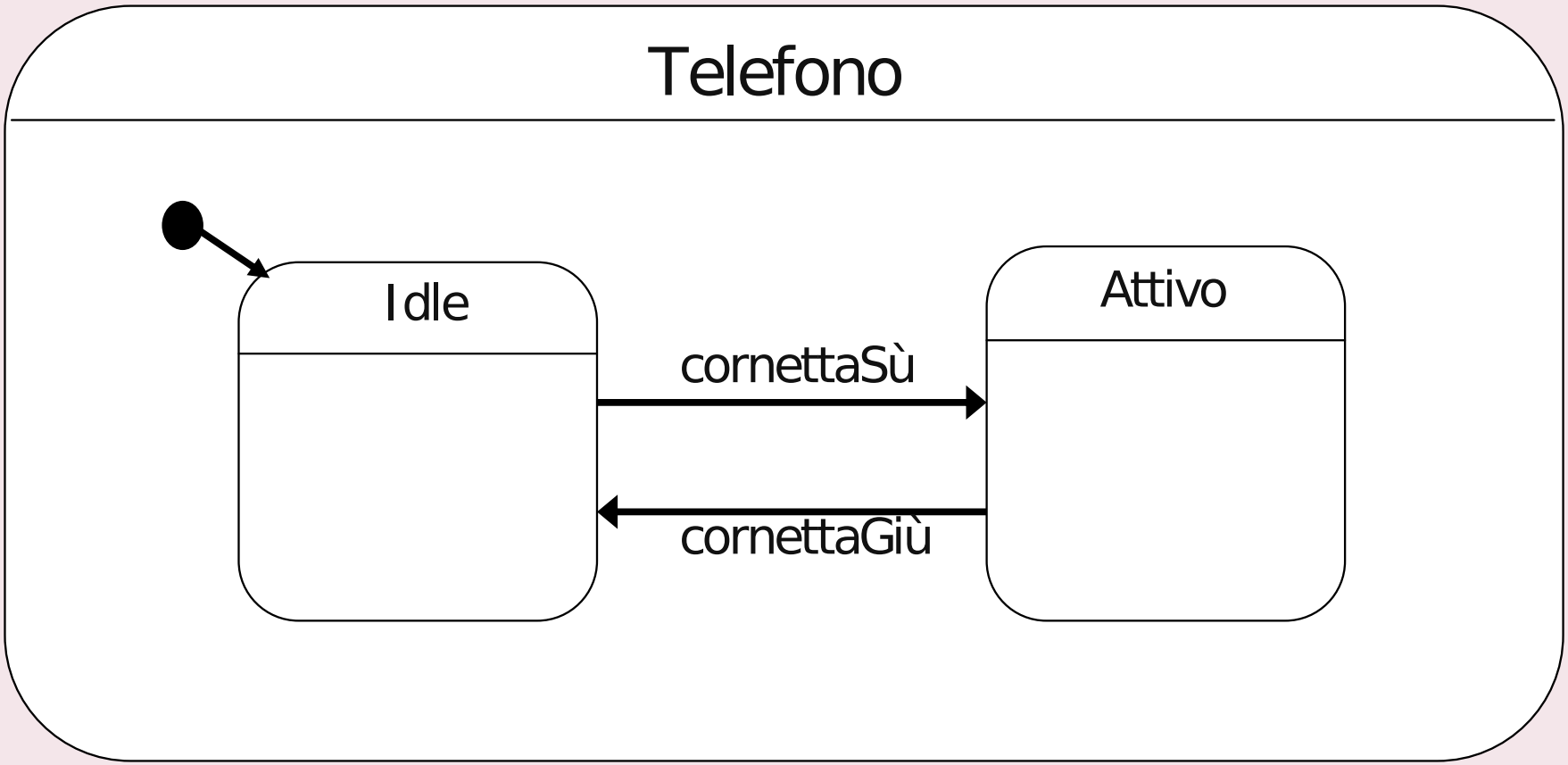
Semantica di una transizione di stato

- Nell'ipotesi che sia corrente il sotto stato (foglia) A22, la configurazione corrente è Top.A.A2.A22
- Se viene presa la transizione annotata con e1/b (trigger e1, azione b), allora si passa nel sotto stato B11 (foglia) e la nuova configurazione è Top.B.B1.B11
- A seguito della transizione e1/b, si ha dunque un cambio di configurazione. Alcuni stati vengono abbandonati (stati uscenti), in altri si entra (stati entranti). Sugli stati uscenti si eseguono nell'ordine, se esistono, le exit action. Sugli stati entranti vengono eseguite nell'ordine, se esistono, le entry action (si ignorano per semplicità le attività)
- Per stabilire gli stati uscenti e gli stati entranti vale la regola del minimo common ancestor (mca). Partendo dallo stato corrente (A22 nell'esempio) occorre risalire nella configurazione origine sino (ed escluso) al primo macro stato in comune con lo stato destinazione. Nell'esempio il mca è Top. Dunque gli stati uscenti sono $\{A22, A2, A\}$ mentre gli stati entranti sono $\{B, B1, B11\}$
- Seguendo la transizione da A21 ad A22, mca è A2 per cui si esce solo da A21 e si entra solo in A22.

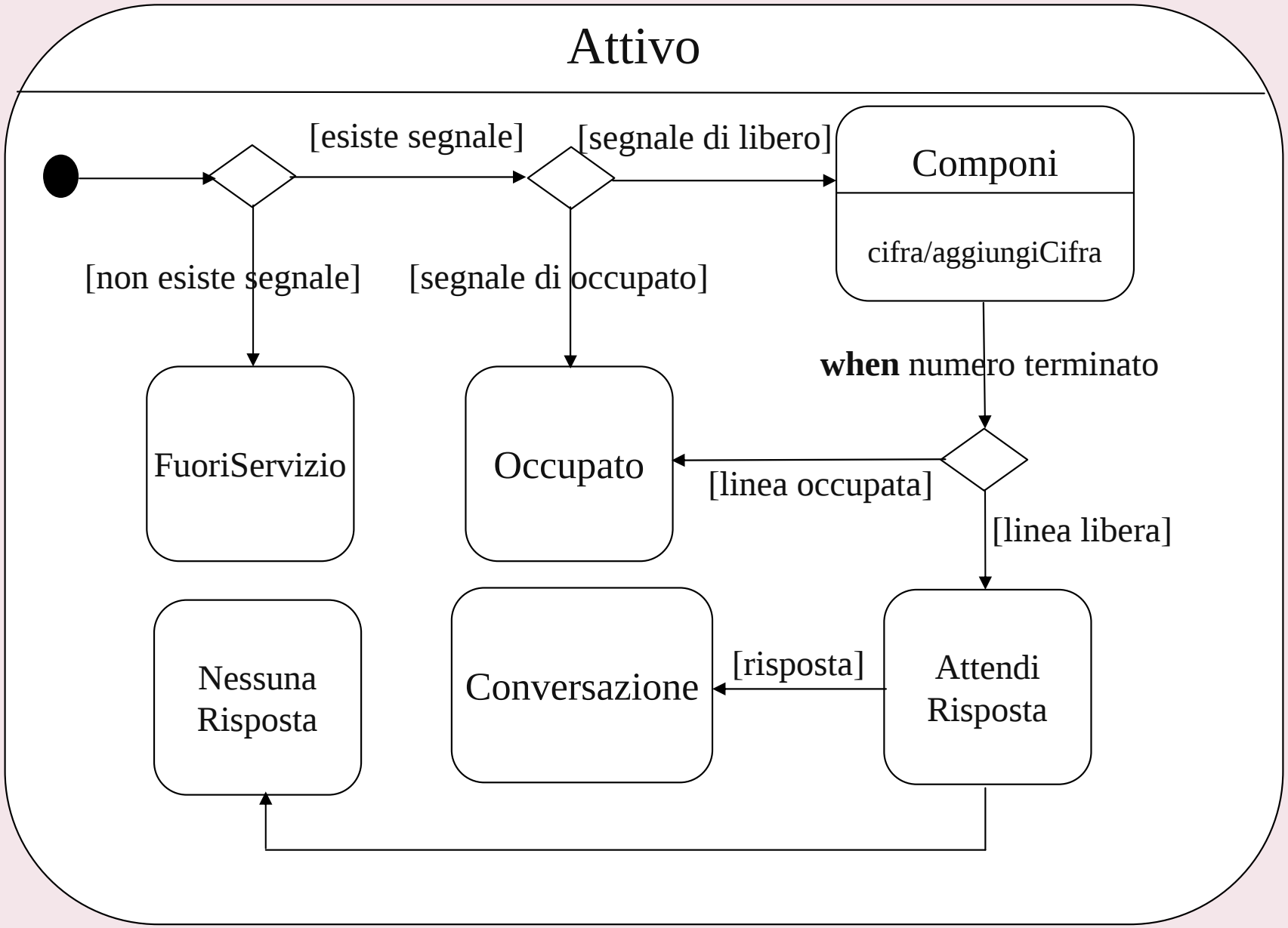
Esercizio

Nell'ipotesi che al tempo della transizione e1/b era corrente A22, e che al momento in cui viene presa la transizione e2/a è corrente B12, specificare l'effetto del rientro con storia **H*** nel macro stato A, e in particolare la sequenza di stati uscenti e la sequenza di stati entranti. Cosa cambia se il connettore di storia è semplicemente **H** ?

Statechart (semplificato) di una telefonata



Il superstato Attivo



Osservazioni

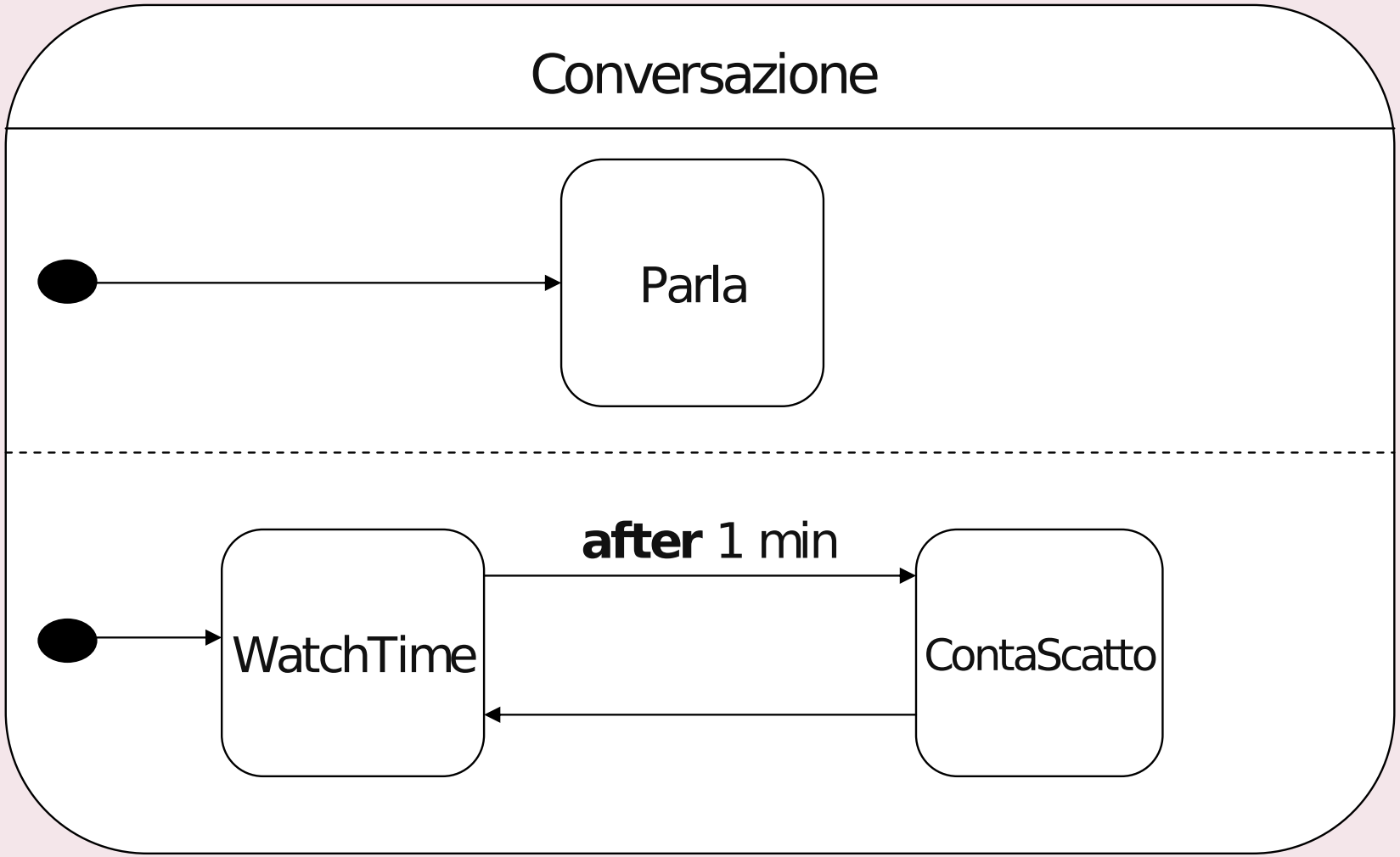
- La decomposizione dello stato Attivo di Telefono mostra in che modo si possono esprimere processi decisionali in uno statechart di UML
- Il rombo (o cerchietto) è il punto decisionale o di scelta. Le condizioni testate, che selezionano un cammino di transizione di stato, si basano su dati e/o parametri di evento, e sono racchiuse entro [e]. Un'alternativa utile nei casi pratici è [else] che consente la scelta un ramo di transizione quando nessuno degli altri ha la condizione true
- L'esempio lascia intendere che esistono eventi (non mostrati) la cui ricezione modifica il valore di alcuni attributi dell'oggetto telefono su cui si fondano le condizioni
- L'espressione "**when** situazione" esprime un evento di cambiamento, legato al raggiungimento di una situazione particolare di dati nell'oggetto. Nell'esempio, per semplicità, si considera un numero terminato o quando è stata digitata l'ultima cifra di un numero corretto o quando si è digitata una cifra di un numero scorretto nel qualcaso si suppone che la linea diventi occupata

Attività

- Si è ammesso tacitamente che lo stato AttendiRisposta disponga di una attività interna relativa all'attesa di un tempo massimo affinchè l'interlocutore risponda
- Un'attività si introduce, in uno stato, con la sintassi:

  do: attività
- Un'attività, diversamente da un'azione, non è atomica. L'arrivo di un evento capace di innescare una transizione ad un differente stato, interrompe definitivamente l'esecuzione dell'attività. Nell'esempio, non appena la persona chiamata solleva la cornetta, si ha la "risposta" e dunque il chiamante esce dallo stato AttendiRisposta e si porta in Conversazione. Similmente, dopo un lasso di tempo massimo, si ha comunque la transizione da AttendiRisposta a NessunaRisposta
- L'esistenza di stati con un'attività interna giustifica le transizioni spontanee, ossia non etichettate con trigger, che vengono prese automaticamente al termine naturale dell'attività
- In generale, uno stato con un'attività interna o con eventi processati localmente allo stato, è sintomatico di un macrostato che ammette un'articolazione in sottostati.

Stati paralleli in Conversazione



Stati paralleli

- La *and*–decomposition di Conversazione indica che quando si entra in questo macro-stato, in realtà si entra contemporaneamente nei due stati Parla e WatchTime
- Si tratta di stati paralleli o concorrenti. I due sotto-automi paralleli sono separati dalla linea tratteggiata. In generale possono sussistere due o più sotto automi concorrenti. L'esempio si riferisce al fatto che si vogliono contare gli scatti della telefonata dopo ogni minuto
- Il modo "classico" degli statechart di UML per denotare il passaggio del tempo è attraverso un evento di tempo: **after** `espressione_di_tempo`. Naturalmente è possibile essere più espliciti con timer e timeout
- ContaScatto è uno stato transitorio: in esso si incrementa un contatore di scatti e quindi si ritorna immediatamente in WatchTime. Esso si potrebbe eliminare agganciando l'azione, *à la Mealy*, all'arco etichettato con: **after** 1 min
- Sotto automi paralleli implicano l'esistenza di thread multipli in uno stesso oggetto e prefigurano, per la necessaria protezione di dati condivisi e modificabili, l'adozione di meccanismi di mutua esclusione e sincronizzazione

Barre di sincronizzazione

- Spesso è necessario sincronizzare le evoluzioni di stato di due o più sotto automi paralleli, in modo che l'avanzamento al prossimo stato in un sotto automa possa avvenire solo se un altro sotto automa ha raggiunto un certo stato
- Le barre di sincronizzazione possono essere utilizzate come fork (per innescare simultaneamente più transizioni) o come join (per sincronizzare più transizioni entranti es. in una uscente)
- Pseudo stati di sincronismo (synch state) indicati con un cerchietto ed un * all'interno possono essere introdotti e condivisi tra i sotto automi da sincronizzare