Capitolo 8

Diagrammi UML degli stati e delle transizioni

In questo capitolo volgiamo la nostra attenzione verso un altro importante documento prodotto nella fase di specifica del software: il diagramma UML degli stati e delle transizioni. Questo diagramma rappresenta uno dei principali documenti UML per la descrizione degli aspetti *dinamici* o *comportamentali* dei programmi, ovvero quelli che si manifestano durante l'esecuzione. Dal punto di vista cronologico, anche questo, come il diagramma delle classi, è un diagramma prodotto durante la fase di analisi. Va comunque notato che un diagramma degli stati e delle transizioni assomiglia molto ad un algoritmo o a un programma, in quanto fornisce importanti indicazioni sul *come* un servizio deve evolvere. Impareremo ad equipaggiare questi diagrammi con opportune formule di LTL, che forniranno la *specifica* più propriamente detta, e a verificare la coerenza del diagramma con la specifica.

8.1 Sintassi dei diagrammi UML degli stati e delle transizioni

In questo capitolo prenderemo in considerazione sia singoli diagrammi degli stati e delle transizioni, sia sistemi costituiti da insiemi di essi. Un diagramma singolo rappresenta bene un sistema *sincrono*, che reagisce ad *eventi* esterni mediante un *cambiamento di stato* e l'esecuzione di un'*azione*.

In particolare, la specifica di un singolo diagramma UML degli stati e delle transizioni prevede la scelta di vari elementi, appresso indicati.

Stati: È necessaria la scelta di un insieme finito S di *stati*, che concettualmente rappresentano momenti in cui il sistema è stabile.

Eventi: È anche necessaria la scelta di un insieme finito E di *eventi*, che concettualmente rappresentano dei segnali che il sistema sincrono riceve dall'esterno, ovvero da una persona o, più tipicamente, da un altro sistema o componente.

Azioni: È infine necessaria la scelta di un insieme finito A di *azioni*, che concettualmente rappresentano delle reazioni che il sistema comunica all'esterno, ovvero a persone o a altri sistemi.

Transizioni: Una transizione rappresenta concettualmente un cambiamento di stato generato da un evento ed accompagnato da un'azione. Più precisamente, è una quadrupla (s, e, s', a), dove $s, s' \in I$, $e \in E$, $a \in A$. s è lo stato di partenza, s' lo stato di arrivo, e l'evento scatenante, a l'azione compiuuta.

Un vincolo da rispettare è che il diagramma sia *deterministico*, ovvero non possono esistere due transizioni (s, e, s', a) e $(s, e, , s'', a')$ con lo stesso stato di partenza (s) e lo stesso evento scatenante (e) ma con diverso stato di arrivo (s', s'') o azione compiuta (a, a').

Definiamo ora formalmente il diagramma.

Definizione 8.1.1 (Diagramma UML degli stati e delle transizioni) *Un diagramma UML degli stati e delle transizioni $\mathcal{T}$ è una quadrupla (I, E, A, T), dove I è un insieme di stati, E è un insieme di eventi, A è un insieme di azioni, T è un insieme di transizioni. Gli insiemi I, E, A devono essere mutuamente disgiunti e T è un insieme di quadruple del tipo (s, e, s', a), dove $s, s' \in S$, $e \in E$, $a \in A$.*

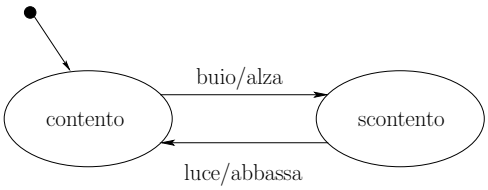

Figura 8.1 Diagramma UML degli stati e delle transizioni per un bambino

Denoteremo l'appartenenza ai vari insiemi mediante simboli di predicato di adeguata arità: $I/1$, $E/1$, $A/1$, *per gli stati, gli eventi e le azioni, rispettivamente,* $T/4$ *per le transizioni.*
Il diagramma $\mathcal{T}$ *deve essere deterministico, ovvero deve valere che:*

$$\forall ss's''eaa' \qquad T(s,e,s',a) \ \wedge \ T(s,e,s'',a') \ \rightarrow s' = s'' \ \wedge \ a = a'.$$

Esempio 8.1.2 La figura 8.1 riporta un diagramma in cui:

- $I = \{contento, scontento\}$,
- $E = \{buio, luce\}$,
- $A = \{alza, abbassa\}$,
- $T = \{(contento, buio, scontento, alza), (scontento, luce, contento, abbassa)\}$.

Il diagramma si riferisce ad un bambino che gioca con un interruttore (non modellato, per ora). Dal punto di vista del bambino, la luce e il buio sono eventi esogeni, a cui egli reagisce mediante transizioni, che si concretizzano in opportune azioni. ◦

Non è obbligatorio che tutti gli insiemi della quadrupla (I, E, A, T) siano non vuoti, come riportato dall'esempio successivo.

Esempio 8.1.3 La figura 8.2 riporta un diagramma in cui:

- $I = \{nuovo, affidabile, moroso, sospetto\}$,
- $E = \{paga, in\_ritardo\}$,
- $A = \emptyset$.

L'elenco delle transizioni viene lasciato come esercizio.
Il diagramma si riferisce ad un cliente di un istituto bancario, modellato dal punto di vista di quest'ultimo. In questo senso gli eventi $(paga, in\_ritardo)$, sono azioni esogene alle quali l'istituto reagisce mediante cambiamenti della maniera in cui percepisce il cliente (ovvero del suo stato) e senza effettuare alcuna azione. ◦

Rispetto alla trattazione tradizionale dei diagrammi UML degli stati e delle transizioni, per semplicità in questo capitolo non prenderemo in considerazione le cosiddette *condizioni*, ovvero quegli aspetti del sistema globale che possono istante per istante essere veri o falsi. Tipicamente (si veda ad es., [23, 18]) una transizione viene modellata mediante una quintupla (s, e, c, s', a), in cui il terzo argomento c è appunto la condizione, che deve essere vera affinché la transizione si attivi.

8.2 Semantica dei diagrammi UML degli stati e delle transizioni

In questo paragrafo specificheremo la semantica dei diagrammi UML degli stati e delle transizioni, fornendo la traduzione in un sistema di transizioni di LTL (cfr. paragrafo 6.2) di ognuno degli elementi sintattici specificati nel paragrafo precedente.
Le differenze maggiori fra un diagramma UML degli stati e delle transizioni $\mathcal{T} = (I, E, A, T)$ e un sistema di transizioni $M = (S, R, L)$ sono le seguenti:

- uno stato di S specifica la verità o falsità di un *insieme* di variabili, non di una sola come avviene in I per $\mathcal{T}$;

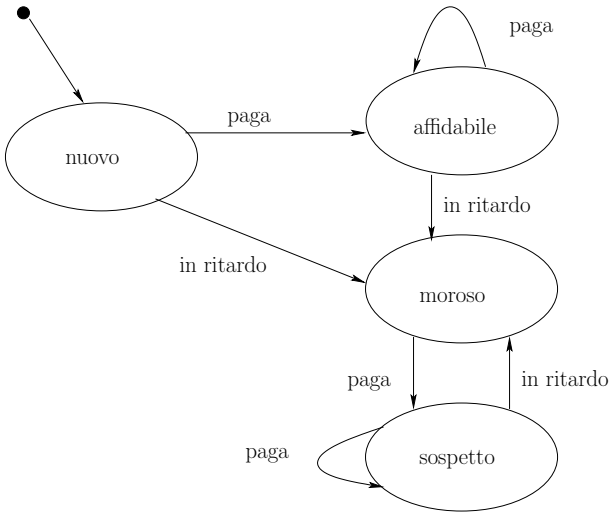

Figura 8.2 Diagramma UML degli stati e delle transizioni per un cliente di istituto bancario

- la relazione di transizione R di M non è etichettata sugli archi, mentre le transizioni T di $\mathcal{T}$ lo sono;
- la relazione di transizione R di M è non deterministica, mentre le transizioni T di $\mathcal{T}$ sono deterministiche.

Il nostro scopo è rappresentare un diagramma UML $\mathcal{T}$ mediante un sistema di transizioni M di LTL, in maniera da preservarne il significato. Per ovviare alle differenze appena menzionate, procederemo con la seguente strategia:

- costuiremo l'alfabeto di M utilizzando tutti i simboli di I, E, A, più altri per azioni ed eventi nulli;
- l'intuizione della traduzione consiste nell'attribuire ad ogni stato S ($\in M$) esattamente tre variabili vere:
  - una per lo stato ($\in I$) di $\mathcal{T}$ *all'istante corrente*,
  - una per l'evento ($\in E$) di $\mathcal{T}$ *all'istante corrente*,
  - una per l'azione ($\in A$) di $\mathcal{T}$ *all'istante passato*;
- costruiremo la relazione di transizione R di M considerando tutti gli elementi delle transizioni in T (stati, evento e azione).

Definiamo ora formalmente la traduzione, introducendo innanzitutto il concetto di evento e azione nulli per un diagramma UML, che rappresentano intuitivamente l'assenza (o l'inefficacia) di eventi e di azioni.

Definizione 8.2.1 (Evento e azione nulli) *Dato un diagramma UML degli stati e delle transizioni $\mathcal{T} = (I, E, A, T)$, definiamo gli insiemi:*

$$\begin{aligned} E^+ &= E \cup \{evento\_nullo\}, \\ A^+ &= A \cup \{azione\_nulla\}, \end{aligned}$$

dove evento_nullo e azione_nulla, sono simboli che non occorrono in $I \cup E \cup A$. E^+ e A^+ saranno usati anche come simboli di predicato unari, come in definizione 8.1.1.

Definizione 8.2.2 (Traduzione in LTL di un diagramma UML degli s. e t.: I (alfabeto)) *Dato un diagramma UML degli stati e delle transizioni $\mathcal{T} = (I, E, A, T)$, sia $\mathcal{T}^+ = (I, E^+, A^+, T)$ con E^+ e A^+ definiti come in Definizione 8.2.1. Definiamo un insieme* **LP** *di lettere proposizionali nel seguente modo:*

$$\mathbf{LP} = I \cup E^+ \cup A^+.$$

$$
\begin{array}{rcl}
\text{Alfabeto}(\mathbf{LP}) & = & I \cup E^+ \cup A^+ \\
I & : & \{\text{contento, scontento}\} \\
E^+ & : & \{\text{buio, luce, evento\_nullo}\} \\
A^+ & : & \{\text{alza, abbassa, azione\_nulla}\}
\end{array}
$$

c:	contento
s:	scontento
l:	luce
b:	buio
ne:	evento_nullo
z:	alza
a:	abbassa
na:	azione_nulla

Tabella 8.1 Alfabeto esteso per il diagramma di figura 8.1

Esempio 8.2.3 [continuazione dell'esempio 8.1.2] La tabella 8.1 riporta l'alfabeto (con relative abbreviazioni) costruito secondo la definizione 8.2.2. ◦

Definizione 8.2.4 (Traduzione in LTL di un diagramma UML degli s. e t.: II (stati)) *Siano $\mathcal{T}^+$ e* **LP** *come in definizione 8.2.2. Definiamo un insieme di stati S ($\subseteq \mathcal{P}(\mathbf{LP})$, ovvero contenuto nell'insieme delle parti di* **LP***) nel seguente modo: per ogni lista di stati, eventi e azioni $ss'ee'aa'$ tale che*

$$T(s, e, s', a) \ \wedge \ A^+(a') \ \wedge \ E^+(e'),$$

sono presenti in S i seguenti stati:

1) (s, e, a'),

2) (s', e', a).

Gli stati di cui al punto 1) rappresentano in LTL gli antecedenti alle transizioni UML: la transizione può avvenire qualsiasi sia stata l'azione che ha precedentemente condotto allo stato s. Gli stati di cui al punto 2) rappresentano in LTL i conseguenti alle transizioni UML: una volta che la transizione è avvenuta ottenendo gli effetti desiderati, l'evento successivo e' può essere qualunque.

Esempio 8.2.5 [continuazione dell'esempio 8.2.3] La figura 8.3 riporta gli stati LTL ottenuti secondo la definizione 8.2.4, prendendo in considerazione una sola delle transizioni di figura 8.1. ◦

Definizione 8.2.6 (Traduzione in LTL di un diagramma UML degli s. e t.: III (transizioni)) *Siano $\mathcal{T}^+$,* **LP** *ed S come in definizione 8.2.4. Definiamo la relazione di transizione R (contenuta in $S \times S$, ovvero una relazione binaria su S) nel seguente modo:*

a) per ogni lista di stati, eventi e azioni $ss'ee'aa'$ come in definizione 8.2.4 tali che $\{(s, e, a'), (s', e', a)\} \subseteq S$, di cui il primo corrispondente al punto 1) e il secondo al punto 2) della definizione 8.2.4, è presente in R la coppia $\langle (s, e, a'), (s', e', a)\rangle$;

b) per ogni lista di stati, eventi e azioni $ss'ee'aa'$ tale che

$$I(s) \ \wedge \ I(s') \ \wedge \ \neg T(s, e, s', a) \ \wedge \ E^+(e') \ \wedge \ A^+(a')$$

(ovvero non è contemplata al punto a)), è presente in R la coppia $\langle (s, e, a'), (s, e', azione\_nulla)\rangle$.

Ituitivamente, le coppie di cui al punto a) sono giustificate dalla presenza di transizioni in T, mentre le coppie di cui al punto b) rappresentano il fatto che il sistema riceve un evento che non gli fa cambiare stato, e quindi reagisce mantenendo lo stato precedente e generando l'*azione_nulla*.

Esempio 8.2.7 [continuazione dell'esempio 8.2.5] La figura 8.4 riporta alcune delle coppie ottenute secondo la definizione 8.2.6. In particolare, in a) viene presa in considerazione la sola transizione di figura 8.3, mentre in b) vengono prese in considerazione due fra le "transizioni mancanti" di figura 8.1: quelle con stato iniziale *contento* ed evento *luce* o *evento_nullo*. ◦

Esercizio 8.1 Completare il grafo di figura 8.4, riportando tutti gli stati e tutte le coppie al fine di ottenere il sistema di transizioni LTL $M = (S, R, L)$ corrispondente al diagramma UML di figura 8.1. ◦

Esercizio 8.2 Applicare il metodo visto per ottenere il sistema di transizioni corrispondente al diagramma UML di figura 8.2. ◦

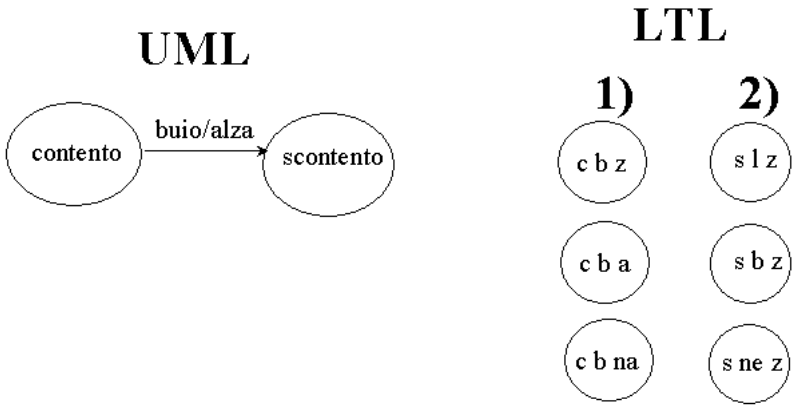

Figura 8.3 Sinistra: una delle transizioni del diagramma di figura 8.1; destra: stati LTL corrispondenti

8.3 Proprietà dei diagrammi UML degli stati e delle transizioni

In questo paragrafo vogliamo imparare a esprimere mediante la notazione di LTL alcune proprietà che possono utilmente fornire specifiche dei sistemi dinamici.

Faremo riferimento ai diagrammi delle figure 8.1 e 8.2 che, per ragioni didattiche, abbiamo già mostrato. È importante tuttavia ricordare che tali diagrammi sono il prodotto di un'attività progettuale che normalmente *segue* la scrittura delle specifiche, che noi diamo solamente ora.

8.3.1 Sistemi sincroni

Esempio 8.3.1 [continuazione dell'esempio 8.1.2] Con riferimento alla figura 8.1, una possibile proprietà del sistema (sincrono) è:

- In presenza di un generatore di eventi che ad ogni istante alterna fra buio e luce, il bambino alterna fra contento e scontento ad ogni istante.

La seguente formula LTL

$$\phi: \; G(\mathit{contento} \;\equiv\; X \; \mathit{scontento}) \tag{8.1}$$

esprime il cambiamento di stato del bambino, istante per istante, in ogni momento del futuro.

Un generatore di eventi che alterna buio e luce ad ogni istante (con la sequenza che inizia con buio), può essere caratterizzato invece dalla seguente formula:

$$\psi: \; G(\mathit{luce} \;\equiv\; \neg \mathit{buio}) \;\wedge\; \mathit{buio} \;\wedge\; G\;(\mathit{luce} \;\equiv X\; \mathit{buio}). \tag{8.2}$$

La proprietà può essere verificata dimostrando che

$$M, \mathit{contento} \models \psi \rightarrow \phi,$$

dove M è la struttura temporale ottenuta dal diagramma degli stati e transizioni usando la metodologia vista in Sezione 8.2 ("*contento*" è lo stato iniziale del diagramma). [1] ∘

[1] Se rilassiamo il vincolo che la sequenza di eventi inizi con "*buio*", diventa possibile che il bambino non cambi stato dopo il primo evento (ricevendo "*luce*"). In questo caso vale che M, contento $\models \psi \rightarrow X\phi$, ovvero la proprietà vale sicuramente solo dopo un transitorio lungo un istante di tempo. La stessa cosa accade se lasciamo libero lo stato iniziale del bambino.

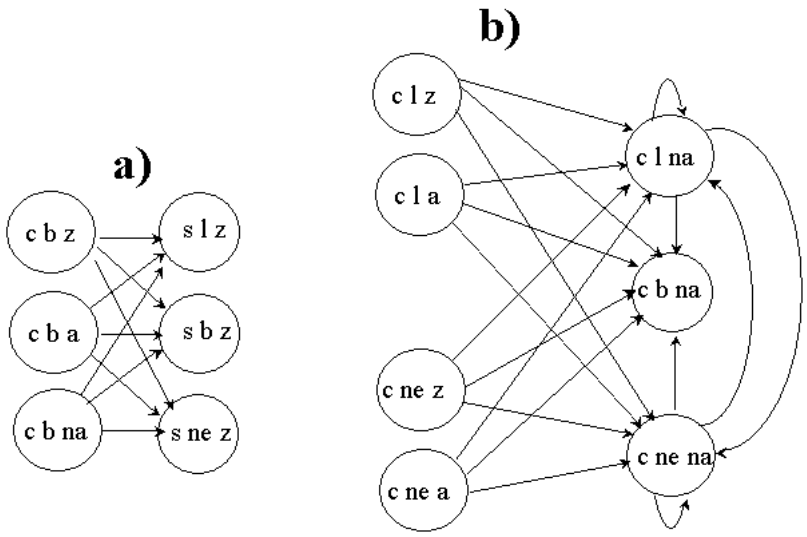

Figura 8.4 Alcune fra le coppie corrispondenti al diagramma di figura 8.1

Esempio 8.3.2 [continuazione dell'esempio 8.1.3] Con riferimento alla figura 8.2, alcune possibili proprietà del sistema (sincrono) sono:

1. inizialmente il cliente è nuovo;
2. dopo il primo evento il cliente è affidabile o moroso;
3. dopo il primo evento il cliente non sarà mai più nuovo;
4. un cliente che continua a pagare rimane affidabile o nuovo;
5. un cliente in ritardo anche solo una volta non sarà mai più affidabile.

Le seguenti formule LTL rappresentano le specifiche di cui sopra:

1. $stato = nuovo$;
2. $stato = nuovo \wedge evento \neq evento\_nullo \rightarrow X(stato = affidabile \vee stato = moroso)$;
3. $evento = evento\_nullo\ W\ XG\ stato \neq nuovo$;
   usando $(\phi\ U\ \psi)\ \vee\ G\phi$ al posto di $\phi\ W\ \psi$:
   $(evento = evento\_nullo\ U\ XG\ stato \neq nuovo)\ \vee\ G\ evento = evento\_nullo$;
4. $(G\ evento\ \neq\ ritardo) \rightarrow G(stato = nuovo\ \vee\ stato = affidabile)$;
5. $G(F\ evento\ =\ ritardo \rightarrow FG\ stato\ \neq\ affidabile)$;
   specifica migliore, usando W (cfr. paragrafo 6.5.1.1):
   $evento\ \neq\ ritardo\ W\ XG\ stato\ \neq\ affidabile$.

○

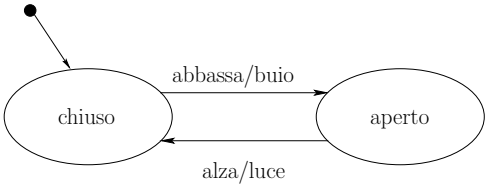

Figura 8.5 Diagramma UML degli stati e delle transizioni per un interruttore

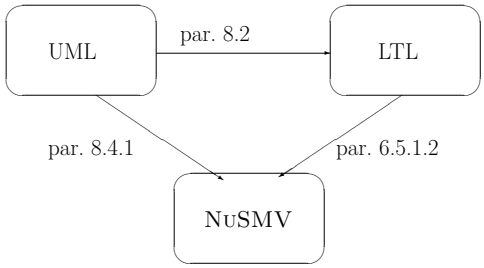

Figura 8.6 Rappresentazioni di un diagramma UML degli stati e delle transizioni

8.3.2 Sistemi asincroni

È possibile combinare uno o più sottosistemi sincroni per ottenere un sistema asincrono. I sottosistemi vengono messi in comunicazione tra loro attraverso opportuni *canali* e le modalità di connessione faranno evolvere il sistema complessivo.

Esempio 8.3.3 Con riferimento alla figura 8.1, introduciamo nella figura 8.5 un ulteriore sottosistema sincrono che rappresenta un interruttore. L'interruttore ha due stati differenti (*chiuso* e *aperto*) e commuta fra questi due a seguito di eventi, producendo azioni.

Si suppone che il bambino (cfr. figura 8.1) interagisca con l'interruttore, scambiandosi eventi ed azioni. In particolare, notiamo che l'insieme delle azioni del primo (*luce* e *buio*) coincide con l'insieme degli eventi dell'altro, e viceversa.

Abbiamo quindi la possibilita di fare intergire i due sottosistemi, ponendo semplicemente fra loro un canale di comunicazione che faccia sì che l'azione generata dal bambino diventi un evento per l'interruttore, e viceversa, istante per istante. In tale maniera, ci attendiamo che valga per il sistema complessivo una proprietà simile alla (8.1). ○

8.4 Aspetti computazionali

In questo paragrafo vogliamo mostrare l'uso di NuSMV per la verifica automatica di proprietà come quelle presentate nel paragrafo 8.3.

8.4.1 Uso di NuSMV *per la verifica di proprietà dei diagrammi UML degli stati e delle transizioni*

In questo paragrafo iniziamo col trattare i singoli sottosistemi (sincroni), fornendo delle regole generali e alcuni esempi, che prosegono quelli del paragrafo 8.3.1.

La figura 8.6 mostra schematicamente le varie rappresentazioni dei sottosistemi, nella logica temporale e nei diagrammi UML degli stati e delle transizioni. Potremmo usare i metodi generali, esposti nei paragrafi 6.5.1.2 e 8.2, per trasformare un diagramma UML degli stati e delle transizioni in un programma NuSMV, passando per LTL. In questo paragrafo preferiamo mostrare un metodo più diretto, che trasforma un diagramma UML U in un file NuSMV `F`.

Il file NuSMV `F` ha il seguente formato:

- vengono dichiarati due moduli:
  1. uno per la descrizione dell'oggetto (`U`) del tipo di U;
  2. quello obbligatorio (`main`), che funge da cliente del primo e gli fornisce gli eventi;

- nel primo modulo vengono dichiarate le variabili necessarie a descrivere gli stati, gli eventi e le azioni di U, che appartengono tutte a tipi enumerati opportunamente dichiarati; tali tipi comprendono `null`, che sta sia per l'evento sia per l'azione nulla (cfr. definizione 8.2.1);
- nel secondo modulo viene dichiarato un oggetto di tipo `U`, di cui si diventa clienti;
- la funzione di transizione del primo modulo specifica le transizioni e le "non-transizioni" del diagramma UML U:
  - per ogni transizione di U va specificato un caso;
    - nell'antecedente del caso (a sinistra di "`:`") vanno specificati lo stato di partenza e l'evento scatenante, lasciando l'azione indeterminata (ricordiamo infatti, cfr. paragrafo 8.2, che tale azione si riferisce al *passato*, e quindi è irrilevane ai fini della transizione);
    - nel conseguente del caso (a destra di "`:`") vanno specificati lo stato di arrivo e l'azione compiuta, lasciando l'evento successivo indeterminato (ricordiamo infatti, cfr. paragrafo 8.2, che tale evento si riferisce al *futuro*, e quindi è irrilevante ai fini della transizione);
  - le "non-transizioni" di U vengono catturate tutte insieme con un "caso di default" (in NuSMV viene eseguito solamente il primo caso il cui antecedente è vero); tale caso ("`1`", cioè "vero") specifica che, all'istante successivo;
    - lo stato rimane lo stesso, e
    - non viene compiuta alcuna azione (`null`);
- la funzione di transizione del secondo modulo specifica la legge con cui il cliente genera gli eventi per U;
- la specifica LTL (unica) contiene la formula che si desidera verificare o confutare.

Esempio 8.4.1 [continuazione dell'esempio 8.3.1] In questo esempio immaginiamo che il cliente sia un generatore di eventi che:

- pone il bambino (tramite `ASSIGN`) nel suo stato iniziale di *contento* (cfr. figura 8.1) e
- ad ogni istante (tramite `TRANS`) genera un evento alternando fra buio e luce.

Ci aspettiamo che il bambino alterni fra contento e scontento ad ogni istante, cfr. proprietà (8.1).

Il file NuSMV per l'esempio è il seguente:

```
-- Time-stamp: "2006-06-09 10:41:35 cadoli"
-- File: bambino.smv
-- Descrizione: un bambino irrequieto guidato da eventi

MODULE bambino
VAR
-- descrive gli stati, gli eventi e le azioni
  stato : {contento, scontento};
  evento: {luce, buio, null};
  azione: {alza_levetta, abbassa_levetta, null};
TRANS
-- descrive le transizioni
  case
    stato = contento &  evento = buio: next(stato)  = scontento &
                                       next(azione) = alza_levetta;
    stato = scontento & evento = luce: next(stato)  = contento &
                                       next(azione) = abbassa_levetta;
    1: next(stato) = stato & next(azione) = null;
  esac
-- fine MODULE bambino

MODULE main
VAR
-- un cliente dell'oggetto bambino
  b: bambino;
ASSIGN
-- assegna lo stato e l'evento iniziali al bambino
```

```
  init(b.stato) := contento;
  init(b.evento) := buio;
TRANS
-- descrizione di un "generatore di eventi"
  case
    b.evento = buio : next(b.evento) = luce;
    b.evento = luce : next(b.evento) = buio;
  esac
-- fine MODULE main

LTLSPEC
-- il bambino alterna fra contento e scontento ad ogni istante
G(b.stato = contento <-> X b.stato = scontento)
```

La specifica è vera e viene confermata da NuSMV.

È interessante notare che NuSMV può essere usato anche per determinare sequenze di eventi "pericolose", che pongono il bambino in stallo, ovvero in una situazione in cui il suo stato non cambia più.

Allo scopo, è sufficiente modificare il secondo modulo (`main`) omettendo la funzione di transizione, ottenendo il file appresso riportato.

```
-- Time-stamp: "2006-06-09 10:06:21 cadoli"
-- File: bambinoStallo.smv
-- Descrizione: un bambino irrequieto può andare in stallo?

MODULE bambino
VAR
-- descrive gli stati, gli eventi e le azioni
  stato : {contento, scontento};
  evento: {luce, buio, null};
  azione: {alza_levetta, abbassa_levetta, null};
TRANS
-- descrive le transizioni
  case
    stato = contento &  evento = buio: next(stato)  = scontento &
                                       next(azione) = alza_levetta;
    stato = scontento & evento = luce: next(stato)  = contento &
                                       next(azione) = abbassa_levetta;
    1: next(stato) = stato & next(azione) = null;
  esac
-- fine MODULE bambino

MODULE main
VAR
-- un cliente dell'oggetto bambino
  b: bambino;

LTLSPEC
-- il bambino non è in stallo
G(b.stato = contento <-> X b.stato = scontento)
```

L'esecuzione di NuSMV determina un controesempio alla specifica LTL, che consiste in uno stato iniziale e una serie di eventi che la falsifica, ovvero pone il bambino in stallo. L'output è riportato appresso: si noti "`Loop starts here`", che rappresenta un ciclo, in cui la proprietà non è soddisfatta, che viene ripetuto all'infinito.

```
-- specification   G (b.stato = contento <->  X b.stato = scontento)  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  b.stato = contento
  b.evento = buio
  b.azione = null
-> State: 1.2 <-
  b.stato = scontento
```

```
  b.azione = alza_levetta
-> State: 1.3 <-
  b.evento = luce
  b.azione = null
-- Loop starts here
-> State: 1.4 <-
  b.stato = contento
  b.evento = buio
  b.azione = abbassa_levetta
-> State: 1.5 <-
  b.stato = scontento
  b.evento = luce
  b.azione = alza_levetta
-> State: 1.6 <-
  b.stato = contento
  b.evento = buio
  b.azione = abbassa_levetta
```

○

Esercizio 8.3 Si determinino, anche con l'aiuto di NuSMV, altre due sequenze di eventi che mandano il bambino in stallo. ○

Esempio 8.4.2 [continuazione dell'esempio 8.3.2] In questo esempio rinunciamo a descivere nel modulo cliente la legge con cui gli eventi vengono generati. Il nostro obiettivo è infatti verificare che le proprietà menzionate nell'esempio 8.3.2 siano tutte vere. Per fare ciò poniamo l'oggetto nel suo stato iniziale di *nuovo* (cfr. figura 8.2).

Il file NuSMV per l'esempio è il seguente:

```
-- Time-stamp: "2006-06-08 11:54:40 cadoli"
-- File: cliente_banca.smv
-- Descrizione: diagramma UML S&T compito PrSWI 20-APR-06

MODULE cliente_banca
-- rappresenta un diagramma degli stati e delle transizioni
VAR
-- descrive gli stati, gli eventi e le azioni
  stato : {nuovo, affidabile, moroso, sospetto};
  evento: {paga, ritardo, null};
TRANS
-- descrive le transizioni
  case
    stato = nuovo & evento = paga: next(stato) = affidabile;
    stato = nuovo & evento = ritardo: next(stato) = moroso;
    stato = affidabile & evento = paga: next(stato) = affidabile;
    stato = affidabile & evento = ritardo: next(stato) = moroso;
    stato = moroso & evento = paga: next(stato) = sospetto;
    stato = sospetto & evento = paga: next(stato) = sospetto;
    stato = sospetto & evento = ritardo: next(stato) = moroso;
-- per tutti i casi non contemplati dal diagramma S&T
    1: next(stato) = stato;
  esac
-- fine MODULE cliente_banca

MODULE main
VAR
-- un cliente dell'oggetto "cliente_banca"
  c: cliente_banca;
ASSIGN
-- assegna lo stato iniziale a c
  init(c.stato) := nuovo;
-- TRANS
-- descrizione di un "generatore di eventi" (in questo caso è superflua)
--  case
--    1: next(c.evento) = paga | next(c.evento) = ritardo |
--       next(c.evento) = null;
```

```
--  esac
-- fine MODULE main

LTLSPEC
-- Alcune specifiche soddisfatte
-- INIZIALMENTE IL CLIENTE È NUOVO
-- c.stato = nuovo

-- DOPO IL PRIMO EVENTO IL CLIENTE È AFFIDABILE O MOROSO
-- c.stato = nuovo & c.evento != null -> X (c.stato = affidabile | c.stato = moroso)

-- DOPO IL PRIMO EVENTO IL CLIENTE NON SARÀ MAI PIÙ NUOVO
-- (usiamo "phi U psi | G phi" al posto di "phi W psi")
-- (c.evento = null U X G c.stato != nuovo) | G c.evento = null

-- UN CLIENTE CHE CONTINUA A PAGARE RIMANE AFFIDABILE O NUOVO
-- (G c.evento != ritardo) -> G (c.stato = nuovo | c.stato = affidabile)

-- UN CLIENTE IN RITARDO ANCHE SOLO UNA VOLTA NON SARÀ MAI PIÙ AFFIDABILE
-- G(F c.evento = ritardo -> F G c.stato != affidabile)
-- specifica migliore, usando W
-- (c.evento != ritardo U X G c.stato != affidabile) | G c.evento != ritardo
```

Le specifiche sono tutte vere e vengono confermate da NuSMV. ○

Esercizio 8.4 Si determinino, anche con l'aiuto di NuSMV, sequenze di eventi che mandano il cliente in stallo (cfr. esempio 8.4.1). ○

Esercizio 8.5 [Soluzione a pag. 129] Si consideri la seguente specifica:

> Un docente universitario può essere in servizio oppure assente, in particolare per ferie, malattia o anno sabbatico. Dall'anno sabbatico si può tornare solamente in servizio, mentre se se si è in ferie si può anche andare in malattia e se si è in malattia si può anche andare in anno sabbatico.

1. Scrivere la specifica di cui sopra in formato LTL.
2. Progettare un diagramma UML degli stati e delle transizioni che corrisponda alla specifica.
3. Tradurre, secondo la metodologia indicata in questo paragrafo, la specifica LTL e il diagramma in formato NuSMV.
4. Verificare, con l'aiuto di NuSMV, che il diagramma del punto 2 corrisponda alle specifiche di cui al punto 1

○

8.4.2 Uso di NuSMV *per la verifica di proprietà di sistemi asincroni*

In questo paragrafo proseguiamo la nostra trattazione, considerando sistemi asincroni. Forniremo regole generali e proseguiremo gli esempi del paragrafo 8.3.2.

Come menzionato in precedenza, è importante che i moduli che rappresentano i singoli sottosistemi sincroni vengano collegati da un opportuno canale, che permetta di fare transitare le azioni di un modulo facendole diventare azioni per l'altro, e viceversa. Per semplicità, ci riferiremo a sistemi costituiti da due soli sottosistemi sincroni, corrispondenti a due diagrammi UML degli stati e delle transizioni U_1 e U_2.

Il file NuSMV `F` ha il seguente formato:

- vengono dichiarati tre moduli:
  a) due per la descrizione degli oggetti (`U1` e `U2`) del tipo di U_1 e U_2, rispettivamente; per le regole di costruzione di questi moduli rimandiamo al paragrafo 8.4.1;
  b) quello obbligatorio (`main`), che funge da cliente dei primi due e fornisce loro gli eventi;
- nel modulo `main` vengono dichiarati due oggetti di tipo `U1` e `U2`, di cui `main` diventa cliente;
- il modulo `main` pone inoltre tali oggetti nel loro stato iniziale, tramite `init`;

- la funzione di transizione del modulo `main` specifica la legge con cui il cliente genera gli eventi per U_1 e U_2; come abbiamo visto, una maniera interessante consiste nel fare scambiare eventi ed azioni fra i due sottosistemi; in particolare, ciò può essere fatto utilizzando assegnazioni, che valgono per tutti gli istanti tranne l'iniziale (gestito con il punto precedente) e `next`;
- la specifica LTL (unica) contiene la formula che si desidera verificare o confutare.

Esempio 8.4.3 [continuazione dell'esempio 8.3.3] In questo esempio usiamo le regole generali appena elencate per verificare proprietà interessanti del sistema asincrono così costruito. Ci aspettiamo una proprietà simile alla (8.1), che caratterizzi il fatto che il sistema non va mai in stallo. Infatti, siamo in gado di verificare la seguente proprietà:

$$G(stato = contento \ \equiv \ X \ X \ stato = scontento). \tag{8.3}$$

La presenza di un ulteriore connettivo temporale "X" è giustificata dalla presenza del canale di comunicazione, che di fatto "ritarda" l'invio degli eventi.

Il file NuSMV per l'esempio è il seguente (la specifica è vera e viene confermata dal sistema):

```
-- Time-stamp: "2006-06-09 11:22:18 cadoli"
-- File: bambinoInterruttore.smv
-- Descrizione: un semplice sistema asincrono

MODULE bambino
-- rappresenta un diagramma UML degli stati e delle transizioni
  stato : {contento, scontento};
  evento: {luce, buio, null};
  azione: {alza_levetta, abbassa_levetta, null};
TRANS
  case
    stato = contento &  evento = buio: next(stato)  = scontento &
                                       next(azione) = alza_levetta;
    stato = scontento & evento = luce: next(stato)  = contento &
                                       next(azione) = abbassa_levetta;
    1: next(stato) = stato & next(azione) = null;
  esac
-- fine MODULE bambino

MODULE interruttore
-- rappresenta un diagramma UML degli stati e delle transizioni
VAR
  stato : {chiuso, aperto};
  evento: {alza_levetta, abbassa_levetta, null};
  azione: {luce, buio, null};
TRANS
  case
    stato = chiuso & evento = abbassa_levetta: next(stato)  = aperto &
                                               next(azione) = buio;
    stato = aperto & evento = alza_levetta:    next(stato)  = chiuso &
                                               next(azione) = luce;
    1: next(stato) = stato & next(azione) = null;
  esac
-- fine MODULE interruttore

MODULE main
-- rappresenta il canale di comunicazione fra i due moduli
VAR
-- due clienti del modulo due_stati
  b: bambino;
  i: interruttore;
ASSIGN
-- assegna lo stato e l'evento iniziali al bambino e all'interruttore
  init(b.stato) := contento;
  init(b.evento) := buio;
  init(i.stato) := aperto;
  init(i.evento) := null;
-- le azioni iniziali di b e i sono indifferenti
```

```
-- sincronizzazione dei processi
  next(i.evento) := next(b.azione);
  next(b.evento) := next(i.azione);
-- fine MODULE main

LTLSPEC
-- il bambino alterna fra contento e scontento ad ogni coppia di istanti
G (b.stato = contento <->  X X(b.stato = scontento))
```

○

Esercizio 8.6 Si determinino, anche con l'aiuto di NuSMV, situazioni che non sincronizzano il sistema complessivo, mandando uno o più dei sottosistemi in stallo (cfr. esempio 8.4.1). ○

8.5 Soluzione di alcuni esercizi

Soluzione esercizio 8.5. Per comodità, diamo un nome ai requisiti:

a) Un docente universitario può essere in servizio oppure assente, in particolare per ferie, malattia o anno sabbatico.

b) Dall'anno sabbatico si può tornare solamente in servizio.

c) Se se si è in ferie si può anche andare in malattia.

d) Se si è in malattia si può anche andare in anno sabbatico.

A titolo di esempio, aggiungiamo alcune proprietà che sono *implicite* nei requisiti, e che desideriamo siano verificate.

(e) Dalle ferie non si può andare direttamente in anno sabbatico.

(f) Dall'anno sabbatico non si può andare direttamente in ferie.

1. Per quanto riguarda il punto 1, i requisiti possono essere tradotti così:
   (a) $G(stato = in\_servizio \ \vee \ stato = ferie \ \vee \ stato = malattia \ \vee \ stato = anno\_sabbatico)$.
   (b) $G((stato = anno\_sabbatico \ \wedge \ evento = rientro) \ \rightarrow \ X stato = in\_servizio)$,
   oppure
   $G(stato = anno\_sabbatico \ \rightarrow \ X(stato = in\_servizio \ \vee \ stato = anno\_sabbatico))$
   (c)
   $$G(stato = ferie \ \rightarrow \quad (evento = rientro \ \rightarrow \ X stato = in\_servizio) \ \wedge \ (evento = in\_malattia \ \rightarrow \ X stato = malattia))$$
   (d)
   $$G(stato = malattia \ \rightarrow \quad (evento = rientro \ \rightarrow \ X stato = in\_servizio) \ \wedge \ (evento = in\_anno\_sabbatico \ \rightarrow \ X stato = anno\_sabbatico))$$
   (e) $G(stato = ferie -> X stato \neq anno\_sabbatico)$
   (f) $G(stato = anno\_sabbatico -> X stato \neq ferie)$
2. Per quanto riguarda il punto 2, il diagramma è riportato in figura 8.7.
3. Per quanto riguarda i punti 3 e 4, riportiamo il file NuSMV, che ci consente di verificare le proprietà menzionate.

```
-- Time-stamp: "2006-06-19 16:36:16 marco"
-- File: docente.smv

MODULE docente
VAR
-- descrive gli stati, gli eventi e le azioni
  stato : {in_servizio, ferie, malattia, anno_sabbatico};
  evento: {in_ferie, rientro, in_malattia, in_anno_sabbatico, null};
TRANS
-- descrive le transizioni
  case
    stato = in_servizio & evento = in_ferie: next(stato) = ferie;
    stato = in_servizio & evento = in_malattia: next(stato) = malattia;
    stato = in_servizio & evento = in_anno_sabbatico: next(stato) =
    in_anno_sabbatico;
    stato = ferie & evento = in_malattia: next(stato) = malattia;
    stato = ferie & evento = rientro: next(stato) = in_servizio;
    stato = malattia & evento = in_anno_sabbatico: next(stato) =
    anno_sabbatico;
    stato = malattia & evento = rientro: next(stato) = in_servizio;
    stato = anno_sabbatico & evento = rientro: next(stato) = in_servizio;
-- per tutti i casi non contemplati dal diagramma S&T
    1: next(stato) = stato;
  esac
-- fine MODULE cliente_banca

MODULE main
VAR
-- un cliente dell'oggetto "docente"
  d: docente;
ASSIGN
-- assegna lo stato iniziale a d
  init(d.stato) := in_servizio;
-- fine MODULE main

LTLSPEC
-- (a) UN  DOCENTE UNIVERSITARIO PUÒ ESSERE IN SERVIZIO OPPURE ASSENTE, IN
-- PARTICOLARE PER FERIE, MALATTIA O ANNO SABBATICO
-- G (d.stato = in_servizio | d.stato = ferie | d.stato = malattia |
--    d.stato = anno_sabbatico)

-- (b) DALL'ANNO SABBATICO SI PUÒ TORNARE SOLAMENTE IN SERVIZIO
-- G ((d.stato = anno_sabbatico & d.evento = rientro) ->
--    X (d.stato = in_servizio))
-- OPPURE
-- G (d.stato = anno_sabbatico ->
--       X (d.stato = in_servizio | d.stato = anno_sabbatico))

-- (c) SE SE SI È IN FERIE SI PUÒ ANCHE ANDARE IN MALATTIA
-- G (d.stato = ferie ->
--    ( (d.evento = rientro -> X d.stato = in_servizio) &
--      (d.evento = in_malattia -> X d.stato = malattia) )
--   )

-- (d) SE SI È IN MALATTIA SI PUÒ ANCHE ANDARE IN ANNO SABBATICO
-- G (d.stato = malattia ->
--    ( (d.evento = rientro -> X d.stato = in_servizio) &
--      (d.evento = in_anno_sabbatico -> X d.stato = anno_sabbatico) )
--   )

-- (e) DALLE FERIE NON SI PUÒ ANDARE DIRETTAMENTE IN ANNO SABBATICO
-- G (d.stato = ferie -> X d.stato != anno_sabbatico)

-- (f) DALL'ANNO SABBATICO NON SI PUÒ ANDARE DIRETTAMENTE IN FERIE.
-- G (d.stato = anno_sabbatico -> X d.stato != ferie)
```

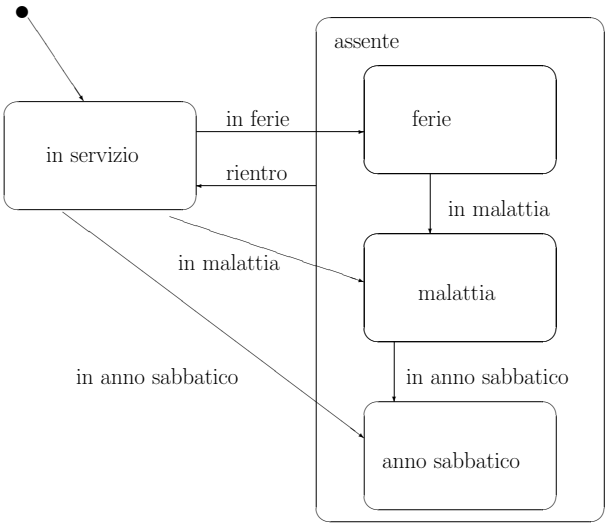

Figura 8.7 Diagramma UML degli stati e delle transizioni per l'esercizio 8.5