Modellazione di sistemi software

Modellare un sistema: rappresentarlo in termini di oggetti matematici che ne riflettono le proprietà

Modellare implica astrarre: semplificare la descrizione del sistema, conservando solo alcuni dei dettagli originari, focalizzandosi sulle proprietà principali e riuscendone a gestire la complessità

Ad esempio, a volte si ignora la durata temporale dei processi

Nelle metodologie di sviluppo del SW per raffinamenti: si definisce il modello, se ne verificano le proprietà, poi si concretizza in codice

Formalismi di modellazione

Molti formalismi sono linguaggi di programmazione (semplificati): si deve programmare la descrizione astratta del sistema.

In alcuni casi ci sono traduttori automatici dalla descrizione originaria del sistema al formalismo di modellazione.

Normalmente però l'astrazione viene eseguita manualmente

Sistemi di transizioni (TS)

Formalismo astratto e generale per la descrizione di sistemi SW.

Il comportamento di un sistema (sequenziale o concorrente) è descritto come se eseguisse una ed un'unica azione elementare (**transizione**) alla volta, che trasformano uno **stato** del sistema in un altro.

Il sistema si trova cioè in ogni istante in uno stato, dal quale può eseguire una transizione tra quelle **abilitate** in quello stato.

Ogni transizione t abilitata in uno stato s trasforma s in uno stato $s' = t(s)$

Il processo continua finché si può eseguire almeno una transizione.

Sistemi di transizioni: definizione

Un sistema di transizioni $\mathcal{T}$ è una tripla $\langle S, I, \rightarrow \rangle$:

- S è l'insieme degli **stati**
- $I \subseteq S$ è l'insieme degli **stati iniziali**
- $\rightarrow \subseteq S \times S$ è l'insieme delle **transizioni**.

  $s \rightarrow s'$ abbrevia $(s, s') \in \rightarrow$
  (esiste una transizione da s a s').

TS con etichette

Un sistema di transizioni può anche avere

- **etichette sugli stati**, che tipicamente rappresentano condizioni che valgono nello stato
- **etichette sugli archi**

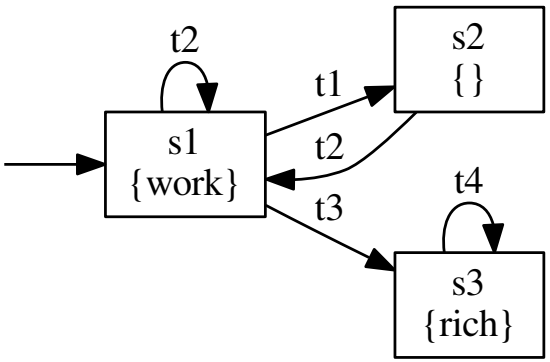
$\mathcal{T}$ è una tupla $\langle S, I, \rightarrow, A, L, \Sigma \rangle$:

- S è l'insieme degli stati
- $I \subseteq S$ è l'insieme degli stati iniziali
- A è l'insieme delle **azioni** (etichette degli archi)
- $\rightarrow \subseteq \boldsymbol{S} \times \boldsymbol{A} \times \boldsymbol{S}$ è l'insieme delle **transizioni**.

$$s \xrightarrow{a} s'$$

- Σ è l'**alfabeto** (un insieme di etichette per gli stati)
- $L : S \rightarrow \Sigma$ è una funzione che associa un'etichetta a ciascuno stato

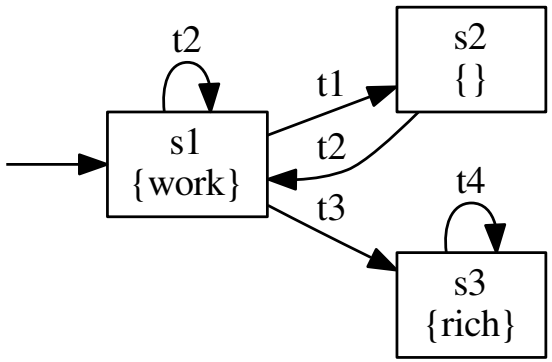
Esempio



- $S = \{s1, s2, s3\}$, $I = \{s1\}$
- $A = \{t1, t2, t3, t4\}$
- $s1 \xrightarrow{t2} s1$, $s1 \xrightarrow{t1} s2$, $s1 \xrightarrow{t3} s3$, $s2 \xrightarrow{t2} s1$, $s3 \xrightarrow{t4} s3$.
- $\Sigma = 2^{\{work, rich\}}$: insieme di tutti i sottoinsiemi di $\{work, rich\}$.
- L è la funzione tale che:

$$L(s1) = \{work\}$$
$$L(s2) = \emptyset$$
$$L(s3) = \{rich\}$$

Esempio



- $S = \{s1, s2, s3\}$, $I = \{s1\}$
- $A = \{t1, t2, t3, t4\}$
- $s1 \xrightarrow{t2} s1$, $s1 \xrightarrow{t1} s2$, $s1 \xrightarrow{t3} s3$, $s2 \xrightarrow{t2} s1$, $s3 \xrightarrow{t4} s3$.
- $\Sigma = 2^{\{work, rich\}}$: insieme di tutti i sottoinsiemi di $\{work, rich\}$.
- L è la funzione tale che: (semantica: $s \models p$ sse $p \in L(s)$)

$$L(s1) = \{work\} \quad (s1 \models work \wedge \neg rich)$$
$$L(s2) = \emptyset \quad (s2 \models \neg work \wedge \neg rich)$$
$$L(s3) = \{rich\} \quad (s3 \models \neg work \wedge rich)$$

Esecuzioni (run)

Un'esecuzione (run) di un TS è una sequenza **massimale** di stati, che inizia in uno stato iniziale e dove ogni stato si ottiene dal precedente mediante una transizione.

Massimale: un'esecuzione termina solo se l'ultimo stato non ha successori (non è **abilitata** alcuna transizione).

Per semplificare, ed evitare di avere sia esecuzioni finite che infinite, si considerano soltanto esecuzioni infinite: le esecuzioni finite sono rappresentate da sequenze infinite in cui l'ultimo stato si ripete all'infinito.

Un'esecuzione di un TS è dunque una sequenza infinita di stati $s_0, s_1, \ldots, s_i, \ldots$ tale che:

- $s_0 \in I$
- per ogni i, vale uno di questi due casi:
  - esiste una transizione $s_i \to s_{i+1}$
  - non esiste alcuna transizione della forma $s_i \to s$ e $s_{i+1} = s_i$

Stati raggiungibili: quelli che compaiono in qualche esecuzione del sistema

Sistemi software e sistemi di transizioni

L'esecuzione di un programma si può vedere come un'esecuzione di un TS dove:

- ogni **stato** del TS rappresenta uno stato in cui si può trovare il sistema SW (valori delle variabili di programma, del program counter, ecc.): uno stato è un'**assegnazione di valori alle variabili**

$x = 0$ $y = 1$	$x = 2$ $y = 1$	$x = 3$ $y = 1$	$x = 4$ $y = 2$	$x = 4$ $y = 4$

- Gli **stati iniziali** possono essere specificati per mezzo una **formula** (senza quantificatori) che deve essere soddisfatta da tutti e solo gli stati iniziali

$$(x > 2 \wedge y \leq 3) \vee x = 0$$

$x = 0$ $y = 1$	$x = 2$ $y = 1$	$x = 3$ $y = 1$	$x = 4$ $y = 2$	$x = 4$ $y = 4$

Sistemi a stati finiti

Sistemi che si possono trovare, in ogni istante, in uno stato appartenente ad un insieme finito.

Per descrivere un sistema a stati finiti è sufficiente un linguaggio proposizionale. Ad esempio, si utilizzerà una lettera proposizionale per rappresentare la verità di una relazione come $x > y$.

Uno **stato** è dunque un'**interpretazione proposizionale**.

$$\text{stato} \quad \Rightarrow \quad \mathcal{M} : P \to \mathit{Bool}$$

Rappresentazione di un'interpretazione proposizionale $\mathcal{M}$: insieme di variabili proposizionali: tutte e solo quelle vere in $\mathcal{M}$

$$\mathcal{M} \;\Rightarrow\; \{p \in P \mid \mathcal{M}(p) = \mathit{TRUE}\}$$

Ad esempio, se $P = \{\mathit{rich}, \mathit{work}\}$, lo stato

$$\boxed{\{\mathit{work}\}}$$

è l'interpretazione $\mathcal{M}$ tale che $\mathcal{M}(\mathit{work}) = \mathit{TRUE}$ e $\mathcal{M}(\mathit{rich}) = \mathit{FALSE}$

Formule e insiemi di interpretazioni

Una formula "denota" l'insieme delle interpretazioni in cui è vera

$$\text{formula } F \quad \Rightarrow \quad \{\mathcal{M} \mid \mathcal{M} \models F\}$$

$$\textit{work} \quad \Rightarrow \quad \left\{ \boxed{\{\textit{work}\}}, \boxed{\{\textit{work}, \textit{rich}\}} \right\}$$

Attenzione: non confondere la **formula** *work* con l'**interpretazione** $\{\textit{work}\}$

Gli **stati iniziali** di un sistema di transizioni su un linguaggio proposizionale sono denotati da una formula proposizionale.

Ad esempio, se $P = \{p, q, r\}$, la formula $(p \wedge \neg q) \vee (q \wedge \neg r)$ denota l'insieme degli stati

Specifica delle transizioni (proposizionali)

L'insieme delle transizioni di un TS (su un linguaggio proposizionale) si può specificare mediante un insieme di regole della forma:

$$t : F \to (p_1, \ldots, p_n) := (G_1, \ldots, G_n)$$

dove t è il nome della transizione, $F, G_1, \ldots, G_n$ sono formule (proposizionali) e $p_1, \ldots, p_n$ sono atomi.

La formula F è la **condizione di abilitazione**, la parte alla destra della freccia descrive come si ottiene lo stato target della transizione.

La transizione t

- è **abilitata** in ogni stato $\mathcal{M}$ che soddisfa F e
- trasforma lo stato $\mathcal{M}$ nello stato $\mathcal{M}' = t(\mathcal{M})$ che si ottiene da $\mathcal{M}$ modificando i valori delle variabili $p_1, \ldots, p_n$ e lasciando immutati gli altri: p_i avrà in $t(\mathcal{M})$ il valore che la formula G_i ha in $\mathcal{M}$.

$$\text{per ogni } p\text{: } \mathcal{M}'(p) = \begin{cases} \mathit{TRUE} & \text{se } p = p_i \in \{p_1, \ldots, p_n\} \text{ e } \mathcal{M} \models G_i \\ \mathit{FALSE} & \text{se } p = p_i \in \{p_1, \ldots, p_n\} \text{ e } \mathcal{M} \not\models G_i \\ \mathcal{M}(p) & \text{altrimenti } (p \notin \{p_1, \ldots, p_n\}) \end{cases}$$

Esempio

Sia $P = \{rich, work\}$, e

$$t : work \rightarrow (work, rich) := (\bot, \neg rich)$$

t è abilitata in $\boxed{\{work\}}$ e in $\boxed{\{rich,work\}}$

$$\boxed{\{work\}} \xrightarrow{t}$$

Esempio

Sia $P = \{rich, work\}$, e

$$t : work \rightarrow (work, rich) := (\bot, \neg rich)$$

t è abilitata in $\boxed{\{work\}}$ e in $\boxed{\{rich,work\}}$

$$\boxed{\{work\}} \xrightarrow{t} \boxed{\{rich\}}$$

$$\boxed{\{rich,work\}} \xrightarrow{t}$$

Esempio

Sia $P = \{rich, work\}$, e

$$t : work \rightarrow (work, rich) := (\bot, \neg rich)$$

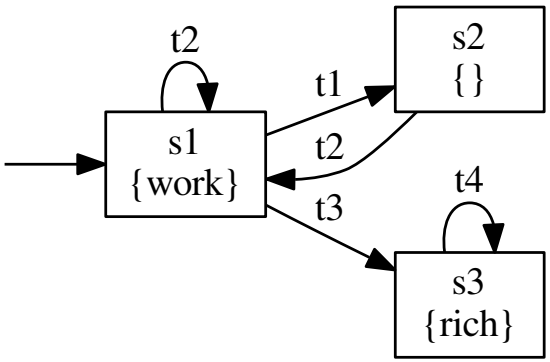
t è abilitata in $\boxed{\{work\}}$ e in $\boxed{\{rich,work\}}$

$$\boxed{\{work\}} \xrightarrow{t} \boxed{\{rich\}}$$

$$\boxed{\{rich,work\}} \xrightarrow{t} \boxed{\emptyset}$$

t non è abilitata né in $\boxed{\{rich\}}$ né in $\boxed{\emptyset}$.

Esempio



Una descrizione implicita di questo TS è data considerando

- Il linguaggio proposizionale $P = \{work, rich\}$
- La formula $\neg rich \wedge work$ per identificare gli stati iniziali
- Le regole di transizione

$$t_1 : work \rightarrow work := \bot$$
$$t_2 : \neg rich \rightarrow work := \top$$
$$t_3 : work \rightarrow (rich, work) := (\top, \bot)$$
$$t_4 : rich \rightarrow rich := rich$$

In generale: il linguaggio per la specifica di sistemi di transizione si può considerare come un **linguaggio di programmazione astratto** per la modellazione di sistemi software.

Assunzioni di fairness

A volte non si vogliono considerare tutte le esecuzioni del modello del sistema, perché alcune di esse, sebbene presenti nel modello, non lo sono nel sistema

Assunzioni di fairness: vincoli semantici imposti sulle esecuzioni, per escludere quelle irragionevoli

Esempio: P_1 e P_2 eseguiti per sempre, senza interazioni.

In ogni stato è sempre abilitata sia una transizione di P_1 che una di P_2.

Nel modello ci sono anche esecuzioni dove, da un certo punto in poi, si eseguono solo transizioni di P_1 (o di P_2).

Una rappresentazione fedele dovrebbe soddisfare il **vincolo di fairness**: ogni esecuzione contiene un numero infinito di transizioni di ciascun processo.

Una modellazione esatta dovrebbe tener conto dello scheduler reale, ma il modello sarebbe troppo specifico e poco pratico.

Si usa un modello più semplice con **fairness constraints** che escludono le esecuzioni irragionevoli.

Vincoli di fairness

Possono avere diverse forme, ma quella più semplice si può esprimere mediante un

insieme di stati $f \subseteq S$

Esecuzione fair: contiene infinite occorrenze di almeno un elemento di f.

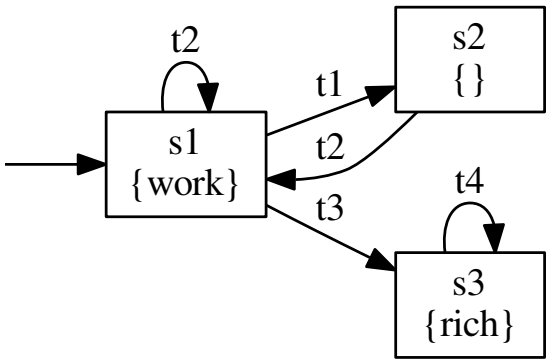
Un **insieme di vincoli di fairness** è un insieme

$$F = \{f_1, ..., f_k\} \text{ con } f_i \subseteq S$$

Esecuzione fair: per ciascun $i = 1, ..., k$, contiene infinite occorrenze di almeno un elemento di f_i.

Un vincolo di fairness (insieme di stati) si può rappresentare mediante una **formula** (che identifica l'insieme degli stati che la soddisfano), e un insieme di vincoli di fairness mediante un **insieme di formule**.

Esempio



Se vogliamo escludere (come irrealistiche!) tutte le esecuzioni in cui non si diventa mai ricchi, si può aggiungere alla descrizione del sistema il vincolo di fairness

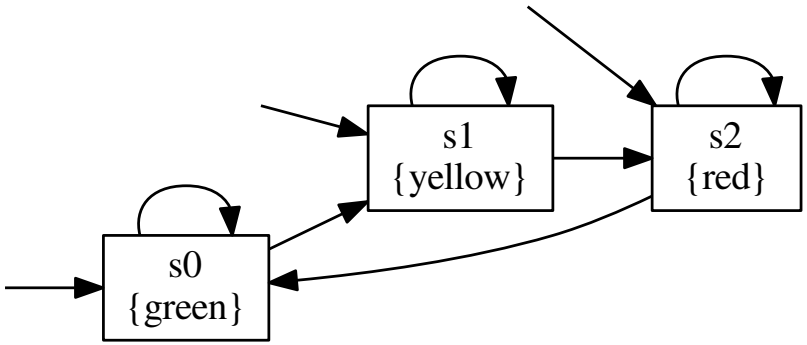
$$f = \{s3\}$$

O, equivalentemente, la formula $F = \mathit{rich} \wedge \neg \mathit{work}$ (o, in questo caso, semplicemente $F = \mathit{rich}$). Infatti F è vera in tutti e solo gli stati di f.

Le esecuzioni fair sono quelle che contengono infinite occorrenze di $s3$ (o che rendono vera F infinite volte):

$$s1, (s2, s1)^*, s3^\omega$$

Esempio 2: il comportamento di un semaforo



Se si vogliono escludere come irrealistiche le esecuzioni in cui da un certo punto in poi il semaforo resta indefinitivamente dello stesso colore, si possono aggiungere i vincoli di fairness

$$F = \{f_1, f_2\} \quad \text{con} \quad f_1 = \{s0\} \text{ e } f_2 = \{s2\}$$

I due vincoli di fairness possono essere espressi mediante l'insieme di formule

$$\Phi = \{green, red\}$$

Le esecuzioni fair sono quelle in cui *green* si verifica infinite volte (passano infinite volte per s0) e anche *red* si verifica infinite volte (passano infinite volte per s2).