Automazione

18 Gennaio 2018

Esercizio 1

Si consideri il problema di accesso concorrente alla risorsa BUS da parte dei seguenti componenti: un numero N di processori (CPU1, CPU2, ..., CPUN) e una MEMORIA per istruzioni e dati.

Il BUS è composto da una sezione DATI, una sezione INDIRIZZO, una sezione SORGENTE e una sezione DESTINAZIONE. Quest'ultime due servono a codificare rispettivamente la sorgente e destinazione dei dati e possono assumere i valori CPU1, CPU2, ..., CPUN o MEMORIA.

Una generica CPUx (con $x = 1, 2, \ldots, N$) ha un registro PROGRAM_COUNTERx che tiene conto dell'indirizzo di memoria della prossima istruzione da eseguire e che viene aggiornato autonomamente dalla CPUx; ha un registro ISTRUZIONEx che memorizza l'istruzione da eseguire; ha un registro STATO_ISTRUZIONEx che tiene conto se l'istruzione è da CARICARE, ESEGUIRE o SALVARE; ha un registro RISULTATOx dove è memorizzato l'ultimo risultato elaborato dalla CPUx; ha un registro INDIRIZZOx che memorizza l'indirizzo di memoria dove salvare il risultato.

La MEMORIA ha un registro BUFFER per memorizzare i dati letti o da scrivere nel BUS, un registro CPU dove memorizzare la CPU a cui inviare i dati letti e un registro POSIZIONE in cui memorizzare l'indirizzo della cella di memoria da leggere o in cui salvare i dati e un flag DISPONIBILE che è TRUE quando la memoria non sta eseguendo operazioni di lettura o scrittura, FALSE altrimenti.

Il ciclo di CPUx è quello di leggere una istruzione, eseguirla e scriverne il risultato nella MEMORIA. Per leggere una istruzione, una CPU pone

SORGENTE=CPUx, DESTINAZIONE=MEMORIA, INDIRIZZO=PROGRAM_COUNTERx

e attende fino a che sul BUS sia presente DESTINAZIONE=CPUx, in tal caso la CPUx pone

ISTRUZIONEx=DATI, ISTRUZIONE_ESEGUITAx=FALSE.

Quando ISTRUZIONE_ESEGUITAx=TRUE, la CPUx pone

DESTINAZIONE=MEMORIA, DATI=RISULTATOx, INDIRIZZO=INDIRIZZOx.

Il ciclo ricomincia da capo e non termina mai (si ipotizzi che CPUx abbia sempre una istruzione da eseguire).

Il ciclo della MEMORIA prevede che quando sul BUS si ha DESTINAZIONE=MEMORIA, essa legga l'indirizzo (POSIZIONE = INDIRIZZO), legga il dato (BUFFER = DATI) e memorizzi la sorgente (CPU=SORGENTE). Se la posizione della locazione di memoria è minore o uguale di un valore costante SLACK, allora l'istruzione è di lettura, altrimenti l'istruzione è di scrittura. Durante le operazioni di lettura o scrittura (che avvengono automaticamente) il flag DISPONIBILE diventa FALSE. Terminata la lettura o scrittura, il flag DISPONIBILE diventa TRUE. In caso di lettura, la memoria pone DATI=BUFFER, DESTINAZIONE=CPU. Il ciclo ricomincia da capo.

All'avvio del sistema, STATO_ISTRUZIONEx = CARICARE per ogni CPU e DISPONIBILE= TRUE. Considerare nulli i transitori di scrittura delle informazioni sul BUS.

Si chiede di tracciare i diagrammi SFC che eseguano correttamente i cicli dei componenti CPUx e MEMORIA, evitando che ci sia accesso al BUS contemporaneo o in ordine scorretto da parte dei componenti. Se possibile, cercare di rendere equamente distribuito l'accesso alle risorse del sistema (BUS e MEMORIA) da parte delle CPU.

Esercizio 2

Si consideri la rete di Petri in Fig. 1.

- Determinare la matrice di incidenza $\boldsymbol{C}$ della rete.
- Disegnare l'albero di raggiungibilità/copertura della rete.
- Classificare la rete in termini di vivezza, reversibilità e limitatezza.
- Calcolare i P-invarianti canonici della rete. La rete è conservativa? Giustificare la risposta.
- Scrivere, se esistono, le equazioni di invarianza.
- Calcolare i T-invarianti della rete e studiarne l'ammissibilità.
- Indicare, se esiste, una sequenza di scatti che porta la rete dalla marcatura $(1\ 1\ 0\ 0\ 2\ 0)^T$ alla marcatura $(1\ 0\ 1\ 1\ 1\ 0)^T$.

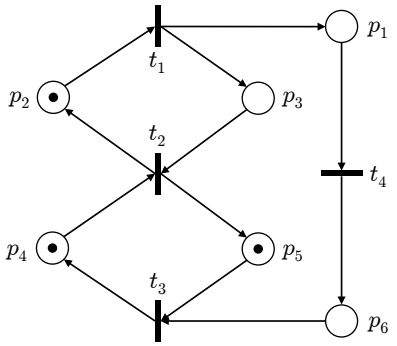

Figura 1: Una rete di Petri con marcatura iniziale $\boldsymbol{x}_0 = (0\ 1\ 0\ 1\ 1\ 0)^T$.

Esercizio 3

In un sistema informatico ci sono 4 agenti software che possono accedere a un disco rigido per la lettura di un dato memorizzato o per la sua scrittura. Nel disco sono state create 4 copie dello stesso dato per permettere letture contemporanee a più agenti. Le operazioni che un agente software esegue per leggere il dato sono le seguenti:

- accesso al disco
- lettura del dato
- copia del dato letto nella propria memoria

Le operazioni che un agente software esegue invece per scrivere il dato sono le seguenti:

- lettura dalla propria memoria del dato da scrivere
- accesso al disco
- scrittura del dato mediante una testina T

Per evitare la corruzione del dato, la scrittura può essere fatta solo da un agente per volta e solo se non c'è nessuna operazione di lettura.

1. Modellare il sistema descritto con una rete di Petri. Utilizzare nomi significativi per i posti.
2. La testina T di scrittura sul disco può guastarsi durante una scrittura, che quindi fallisce e viene abortita. Le copie del dato non sono però corrotte ed è possibile che agenti accedano in lettura e in scrittura con una testina di riserva R. La testina di scrittura T può essere poi riparata (mentre la testina R è a riposo) e gli agenti software potranno nuovamente accedere con essa alla scrittura del dato. Si modifichi la rete di Petri per modellare questi eventi.

[180 minuti; libri aperti]

Soluzioni

18 Gennaio 2018

Esercizio 1

Le risorse mutuamente esclusive sono il BUS e la MEMORIA. Quando un processore necessita di eseguire una istruzione, tutto il ciclo di lettura istruzione e di scrittura del risultato deve essere eseguito in blocco, per evitare situazioni di inconsistenza logica. Durante l'elaborazione di una istruzione il BUS può essere lasciato disponibile ad altre CPU. Pertanto, l'unica risorsa da gestire con un semaforo è il BUS. Per stabilire un ordine di priorità che sia equo tra i processori, si può adottare un registro PRIORITY che assuma un valore numerico da 1 a N in base alla CPU che ha maggiore priorità di accesso alle risorse. Conviene tracciare due diagrammi SFC: uno per la generica CPUx (con $x = 1, 2, \ldots, N$) e uno per la MEMORIA. Nella Fig. 2 si riporta il diagramma SFC della generica CPUx. Nella Fig. 3 si riporta il diagramma SFC della MEMORIA.

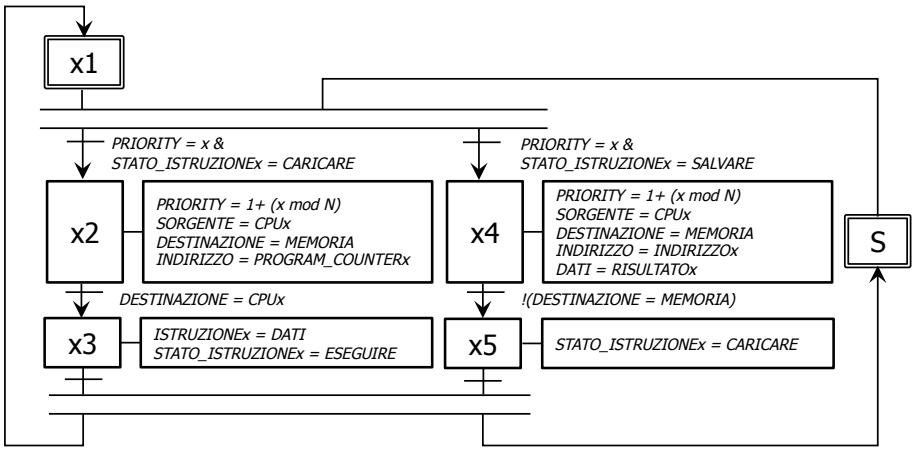

Figura 2: Diagramma SFC della generica CPUx.

- La disponibilità del BUS è registrata in uno stato semaforico S. La generica CPUx partirà da uno stato $x1$ e accederà alla risorsa BUS solo se è ad essa associata la PRIORITY. Inoltre, la CPUx ha due possibilità (mutuamente esclusive) quella di dover CARICARE una istruzione o quella di dover SALVARE il risultato di una istruzione eseguita. Nel primo caso si attiverà lo stato $x2$, nel secondo caso si attiverà lo stato $x4$. In entrambi i casi il BUS viene reso non disponibile.
- Nello stato $x2$ è necessario caricare una istruzione e aggiornare la priorità. Per aggiornare la priorità viene utilizzata la funzione *mod* N sul valore x, incrementata di uno, permettendo così di dare priorità alla successiva CPU e, in caso la CPU corrente sia la N-esima, venga data priorità alla CPU1. Il BUS rimane occupato per tutto il ciclo di lettura della istruzione, la CPUx infatti attende la condizione DESTINAZIONE = CPUx per continuare, legge quindi la istruzione dal BUS e aggiorna lo stato della istruzione (ESEGUIRE) e rilascia il BUS.
- Nello stato $x4$ è necessario salvare una istruzione, quando il risultato è pronto, e aggiornare la priorità. Per procedere allo stato $x5$, la CPUx attende che la MEMORIA liberi il BUS (cosa che avviene nello stato 4 del diagramma SFC della MEMORIA) ponendo una DESTINAZIONE diversa dal valore MEMORIA. Nello stato $x5$ viene quindi aggiornato lo stato della istruzione (CARICARE) e rilasciato il BUS.

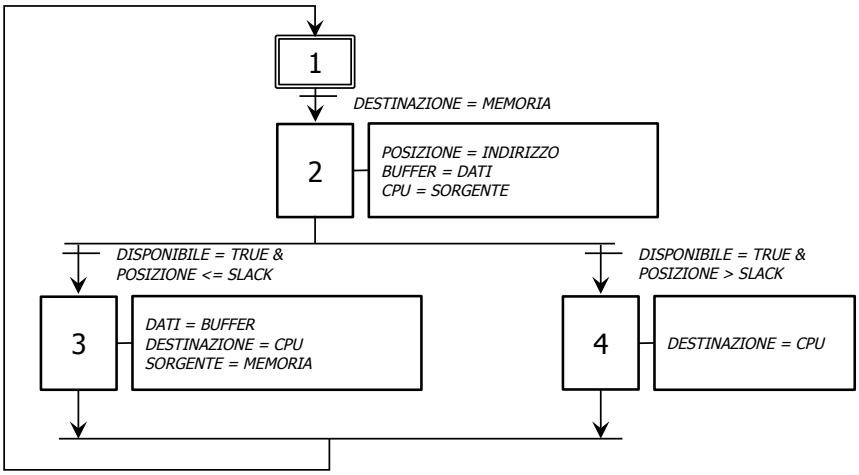

Figura 3: Diagramma SFC della MEMORIA.

- La MEMORIA rimane in attesa che nel BUS venga posto DESTINAZIONE = MEMORIA. Letti i dati dal BUS, la MEMORIA esegue autonomamente la lettura dei dati o la loro scrittura. In entrambi i casi, al termine delle operazioni viene posto DISPONIBILE = TRUE.
- In base al valore del registro posizione è avvenuta una lettura (POSIZIONE $<=$ SLACK) o una scrittura (POSIZIONE $>$ SLACK). Utilizzando una divergenza si differenziano i due casi:
  - Nel caso della lettura vengono aggiornati i dati sul BUS, in particolare viene posto DESTINAZIONE = CPU, per abilitare la lettura dal BUS della CPUx richiedente il dato letto.
  - Nel caso della scrittura, viene cambiata la DESTINAZIONE sul BUS per abilitare lo stato x5 della CPUx che ha richiesto l'operazione di scrittura.
- Una struttura di convergenza chiude quindi il ciclo della memoria.

Esercizio 2

- La matrice di incidenza della rete di Petri in Fig. 1 è

$$\boldsymbol{C} = \begin{pmatrix} 1 & 0 & 0 & -1 \\ -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix},$$

il cui rango è $\rho = 3$, come si può facilmente verificare: la terza riga è proporzionale alla seconda, la quinta alla quarta, la matrice quadrata ottenuta eliminando terza e quinta riga ha determinante nullo ed esiste (almeno) un minore 3×3 non nullo. Lo spazio nullo di $\boldsymbol{C}^T$ ha quindi dimensione $|P| - \rho = 6 - 3 = 3$, mentre lo spazio nullo di $\boldsymbol{C}$ è mono-dimensionale ($|T| - \rho = 4 - 3 = 1$).

- L'albero di raggiungibilità della rete di Petri è riportato in Fig. 4. Si noti che l'albero deve essere costruito con una logica 'depth first'. Inoltre si considera l'ordine lessicografico delle etichette delle transizioni: il ramo che parte dalla transizione t_2 nella marcatura $(1\ 0\ 1\ 1\ 1\ 0)^T$ si espande quindi prima di quello di t_3 e così via.

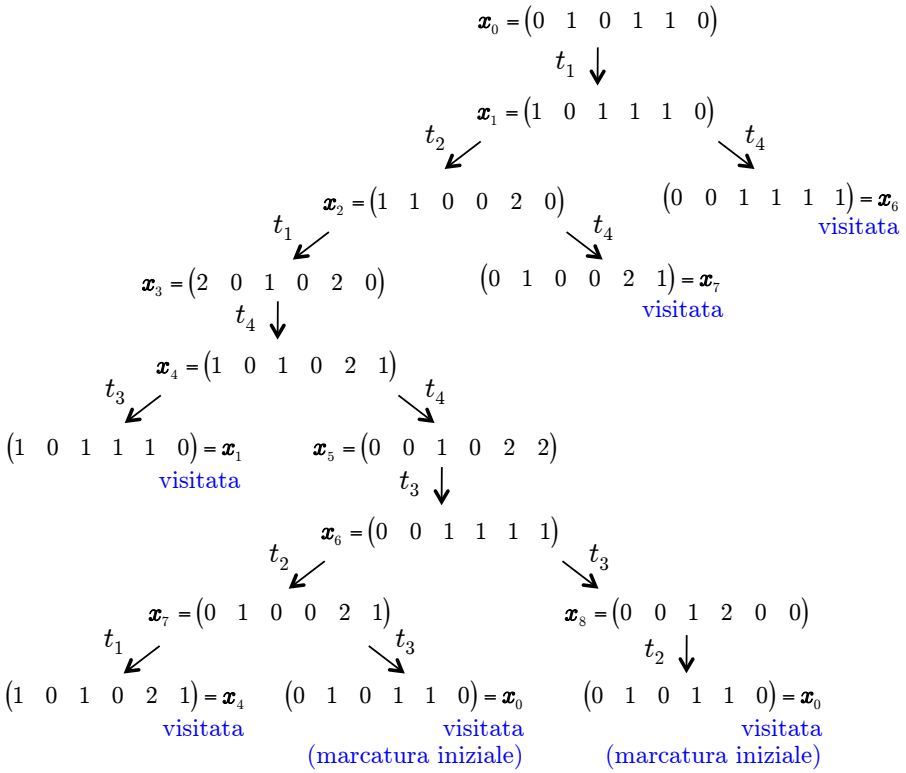

Figura 4: Albero di raggiungibilità della rete di Petri di Fig. 1.

- Analizzando l'albero di raggiungibilità si può verificare che la rete è viva, reversibile e 2-limitata.
- Il calcolo dei P-invarianti

$$\boldsymbol{\gamma}^T \boldsymbol{C} = \boldsymbol{0}^T \quad \Rightarrow \quad \begin{cases} \gamma_1 - \gamma_2 + \gamma_3 = 0 \\ \gamma_2 - \gamma_3 - \gamma_4 + \gamma_5 = 0 \\ \gamma_4 - \gamma_5 - \gamma_6 = 0 \\ -\gamma_1 + \gamma_6 = 0 \end{cases} \quad \Rightarrow \quad \begin{cases} \gamma_1 = \gamma_6 \\ \gamma_4 = \gamma_5 + \gamma_6 \\ \gamma_3 = \gamma_2 - \gamma_6 \end{cases}$$

fornisce (avendo scelto rispettivamente $\{\gamma_6 = \gamma_5 = 0, \gamma_2 = 1\}$, $\{\gamma_6 = 0, \gamma_5 = 1, \gamma_2 = 0\}$ e $\{\gamma_6 = 1, \gamma_5 = 0, \gamma_2 = 1\}$) le tre soluzioni linearmente indipendenti, a supporto minimo e canoniche:

$$\begin{aligned} \boldsymbol{\gamma}^1 &= (\ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ )^T \\ \boldsymbol{\gamma}^2 &= (\ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ )^T \\ \boldsymbol{\gamma}^3 &= (\ 1\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1\ )^T . \end{aligned}$$

Poichè ogni posto compare nel insieme di supporto di almeno un P-invariante, la rete risulta conservativa e pertanto limitata.

- Le equazioni di invarianza della rete (della forma $\boldsymbol{\gamma}^T \boldsymbol{x} = \boldsymbol{\gamma}^T \boldsymbol{x}_0$) sono

$$\begin{aligned} x(p_2) + x(p_3) &= 1 \\ x(p_4) + x(p_5) &= 2 \\ x(p_1) + x(p_2) + x(p_4) + x(p_6) &= 2. \end{aligned}$$

  A queste equazioni sono associati *cicli minimi* della rete nei quali si conserva il numero di token complessivi presenti inizialmente. Tali cicli sono $C^1 = \{p_2, p_3\}$ (con 1 token), $C^2 = \{p_4, p_5\}$ e $C^3 = \{p_1, p_6, p_4, p_2\}$ (entrambi con 2 token). Si noti che a P-invarianti non a supporto minimo corrispondono cicli composti: ad esempio, al P-invariante $\boldsymbol{\gamma} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}^T = \boldsymbol{\gamma}^1 + \boldsymbol{\gamma}^2$ corrisponde il ciclo $C = \{p_2, p_3, p_5, p_4\}$ ottenuto concatenando C^1 e C^2.

- Il calcolo dei T-invarianti fornisce

$$\boldsymbol{C}\,\boldsymbol{\eta} = \boldsymbol{0} \quad \Rightarrow \quad \begin{cases} (\eta_1 - \eta_4 &= 0) \\ -\eta_1 + \eta_2 &= 0 \\ (\eta_1 - \eta_2 &= 0) \\ -\eta_2 + \eta_3 &= 0 \\ (\eta_2 - \eta_3 &= 0) \\ -\eta_3 + \eta_4 &= 0 \end{cases} \quad \Rightarrow \quad \begin{cases} \eta_1 = \eta_2 \\ \eta_2 = \eta_3 \\ \eta_3 = \eta_4. \end{cases}$$

  Pertanto, come previsto, esiste un solo T-invariante:

$$\boldsymbol{\eta} = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}^T.$$

  Come vettore delle occorrenze, $\boldsymbol{\eta}$ è associato a diverse sequenze ammissibili di scatti, ad esempio a $\{t_1, t_2, t_4, t_3\}$ (sequenza che rende la rete reversibile) oppure a $\{t_4, t_3, t_2, t_1\}$.

- A partire dall'albero di raggiungibilità ricavato in precedenza, sequenze ammissibili di scatti che portano la rete dalla marcatura $(1\ 1\ 0\ 0\ 2\ 0)^T$ alla marcatura $(1\ 0\ 1\ 1\ 1\ 0)^T$ sono $\{t_1, t_4, t_3\}$ oppure $\{t_4, t_3, t_1\}$. Vanno privilegiate evidentemente le sequenze ammissibili di lunghezza minima.

Esercizio 3

La gestione richiesta delle operazioni di lettura e scrittura da parte dei 4 agenti software in condizioni nominali è modellata dalla rete di Petri in Fig. 5. Ogni volta che avviene un'accesso al disco per scrittura, vengono sequestrati tutti i token dal posto 'copie del dato' e quindi non è possibile accedere in lettura fintanto che l'operazione di scrittura sul disco non sia stata completata. Viceversa, possono coesistere fino a 4 richieste contemporanee di lettura del dato (ramo sinistro della rete). Si noti che, il posto 'copie del dato' e gli archi ad esso connessi si sarebbero potuti eliminare, pesando invece sia l'arco iniziale in ingresso alla sequenza di lettura sia l'arco finale di ritorno al posto 'agenti software' con un valore 4. Tali due reti di Petri si generalizzano al caso di n agenti e m copie dei dati, con $n \geq m \geq 1$, e n finito. Lo schema in Fig. 5 offre però un leggero vantaggio semantico in termini di interpretazione dei token presenti nei due posti indicati. Infine, anche il posto 'testina T' non è strettamente necessario, ma è funzionale alla seconda parte dell'esercizio dove si considera il guasto di questa testina di scrittura.

La Fig. 6 mostra le modifiche della precedente rete di Petri necessarie a gestire la situazione di guasto della testina T e la sua sostituzione con la testina di riserva R secondo le specifiche

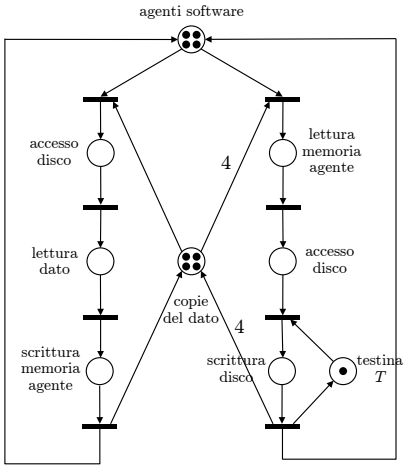

Figura 5: Rete di Petri relativa al primo problema nell'Esercizio 3 (in condizioni nominali).

richieste. La transizione in rosso (con il relativo arco che parte dal normale stato di scrittura con la testina T) rappresenta l'evento di guasto. I due archi in blu indicano la scrittura fallita da parte dell'agente (un token è riportato nel posto 'agenti software' dall'arco di peso unitario) e al contempo la riabilitazione (tramite l'arco di peso 4) del sistema alle operazioni di lettura e scrittura, che verranno però eseguite dalla testina di riserva durante tutto il periodo di riparazione di quella originale. L'arco arancione in uscita dal posto 'accesso disco' serve ad abilitare questo percorso alternativo nel caso in cui la testina di riserva sia entrata in gioco. Gli elementi in verde della rete modellano l'operazione (temporanea) di scrittura sul disco con la testina di riserva R, fintanto che la riparazione di quella originale T non sia terminata —evento che può avvenire solo quando la testina di riserva non stia scrivendo. Gli altri due archi in arancione (uno con peso 4, l'altro unitario) garantiscono il funzionamento ciclico del sistema con la testina di riserva R, in modo analogo a quanto succede con la testina originale T.

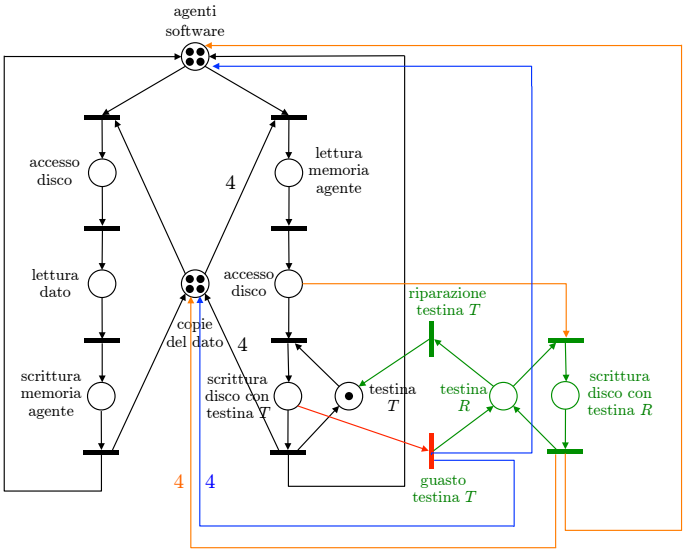

Figura 6: La rete di Petri di Fig. 5 modificata per la gestione del guasto della testina T.

In alternativa, si potrebbe aggiungere un secondo token nel posto 'testina T', eliminando al contempo il posto aggiuntivo 'scrittura dato con testina R', le due transizioni ad esso in ingresso e uscita, e i relativi archi. Il posto 'testina T' prenderebbe in tal caso il nome simbolico 'testine disponibili': quando la testina T è funzionante, sono presenti 2 (testina T a riposo) o 1 token (testina T in scrittura); quando la testina T è guasta, sono presenti 1 (testina R a riposo) o 0 token (testina R in scrittura). Servirebbero anche altri due cambi di nome: 'scrittura disco con testina T' $\rightarrow$ 'scrittura disco' e 'testina R' $\rightarrow$ 'testina T in riparazione'. In questa soluzione si perderebbe però la condizione sul momento di ritorno all'uso della testina T riparata, che potrebbe avvenire anche quanto la testina R è in fase di scrittura.

* * * * *