Dipartimento di Elettronica e Informazione
POLITECNICO DI MILANO


# Automazione industriale
# dispense del corso
# 17. GRAFCET – Sequential Functional Chart

Luigi Piroddi
piroddi@elet.polimi.it

# Introduzione

Fino agli anni ’70 le tecniche in uso per progettare sistemi di automazione erano:

- equazioni combinatorie ricavate da automi a stati finiti
- rappresentazioni circuitali di equazioni logiche
- grafi di evoluzione dello stato (v. automi)
- descrizioni testuali
- sistemi proprietari di varia natura

Tali tecniche producevano descrizioni che soffrivano di numerosi inconvenienti:

- dimensione elevata
- imprecisione
- incompletezza
- ambiguità
- *difficoltà a rappresentare parallelismi tra sequenze*

Nel 1975 venne istituita una commissione con lo scopo di trovare un formalismo maggiormente adatto alla descrizione di sistemi complessi di automazione.

Il risultato dei lavori della commissione fu la definizione del *GRAFCET* (GRAPHe de Coordination Etapes Transitions), formalismo derivato dalle reti di Petri (modelli astratti e generali per rappresentare sistemi ad eventi discreti, introdotti nei primi anni ’60) e finalizzato alla soluzione di problemi di controllo logico in ambito industriale.

Il GRAFCET nasce dalle seguenti esigenze:

- usare modelli formali per rappresentare funzioni di controllo logico
- standardizzare formalismi rappresentativi
- semplificare le reti di Petri e adattarle ad un’implementazione industriale
- usare un linguaggio facilmente interpretabile dal punto di vista del processo

Esso costituisce una semplificazione strutturale delle reti di Petri, con l’aggiunta di una specifica formale del comportamento ingresso/uscita, in modo che da esso si possa ottenere senza ambiguità del codice eseguibile da un macchina sincrona, come il PLC.

Il successo del GRAFCET è testimoniato dal fatto che è diventato uno dei linguaggi standard di modellizzazione di processi industriali discreti ed è stato incluso nella normativa IEC 61131 sui linguaggi di programmazione di PLC, con il nome di Sequential Functional Chart (SFC).

Il GRAFCET è un formalismo molto ricco:

- contiene strutture modellistiche che consentono di rappresentare quanto serve nella maggioranza dei problemi di automazione industriale (per quello che riguarda il controllo logico/sequenziale)
- consente inoltre di programmare in modo strutturato (*modularità* e *gerarchia*)
- spesso sono disponibili più costrutti per rappresentare lo stesso tipo di comportamento

Implicazioni della ricchezza modellistica:

- l’evoluzione di un GRAFCET dipende dalla sua topologia, ma anche da diversi tipi di variabili (di ingresso, di stato, temporali e condivise)
- non basta guardare lo schema per capirne l’evoluzione
- in generale, il GRAFCET non si presta all’applicazione di metodi rigorosi di analisi e verifica (si usa prevalentemente la simulazione)

# SFC e reti di Petri sincronizzate

Punti in comune:

- tecniche basate su posti e transizioni
- rappresentazioni grafiche molto simili
- evoluzione dello stato sincronizzata al verificarsi di eventi esterni

Differenze principali:

- lo stato di una fase nell’SFC è booleano
- nell’SFC tutte le transizioni simultaneamente superabili sono effettivamente tutte contemporaneamente superate
- in un SFC le condizioni di scatto di una transizione possono dipendere dallo stato delle singole fasi
- la marcatura di un posto in una rete di Petri è un numero intero
- nelle reti di Petri le transizioni abilitate vengono superate una alla volta (e in questo modo può darsi che non scattino tutte!)
- ciò non è previsto nelle reti di Petri

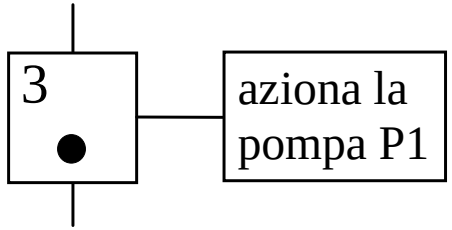
# Elementi di base

- la *fase* (o *passo* o *tappa*, dal termine francese *étape*),
  con le *azioni* ad essa associate
- la *transizione*,
  con la *condizione logica* ad essa associata
- l’*arco orientato*

La fase (v. posto nelle reti di Petri) rappresenta una condizione invariante del sistema che può essere modificata solo all’occorrenza di un determinato evento, che genera una transizione che porta il sistema in un’altra fase.

Essa:

- è simboleggiata da un quadrato (con doppia cornice se iniziale)
- è etichettata con un numero (identificatore di tappa)
- può essere attiva o inattiva
- se è attiva viene marcata (con un gettone)
- può essere associata ad una o più “azioni”




Generalmente, ogni SFC ha una sola fase iniziale, e vi possono essere diversi diagrammi SFC non connessi tra loro (ed eseguiti in parallelo).

Oltre che graficamente, una fase può essere descritta con un costrutto software specifico:

- `STEP nome_fase: ...` *azioni* `... END_STEP`
- `INITIAL_STEP nome_fase: ...` *azioni* `... END_STEP`

In uno schema SFC si dicono *variabili di stato* (o *marker di fase*) le variabili logiche associate allo stato di attivazione delle fasi.

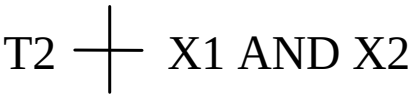
- l’esistenza della fase *i* implica quella di una variabile Xi che vale 1 se la fase *i* è attivata e 0 altrimenti
- se alla fase è stato assegnato un nome, il marker di fase è denotato anche `nome_fase.X`
- tale variabile (definita direttamente dal sistema operativo) è utilizzabile, ma non modificabile dall’utente

Le variabili di stato hanno questo nome perché l’insieme delle fasi attive rappresenta lo stato del sistema ad eventi discreti descritto dallo schema SFC (*condizione* dell’SFC).

Nello schema SFC si può fare direttamente riferimento alle variabili di stato: tipicamente esse compaiono nelle condizioni associate alle transizioni.

## La transizione

- è simboleggiata da una barretta orizzontale
- è dotata di un identificatore (Tn)
- può essere abilitata o no
- se abilitata, può essere superabile o no
- se superabile, può essere superata
- ad essa è associata una *condizione logica* (detta anche *ricettività*)




## Una condizione può essere specificata in vari modi:

- testo strutturato
- diagramma LD
- diagramma FBD
- assegnando un nome alla transizione e definendola a parte con un costrutto software specifico (`TRANSITION nome_transizione :=...; END_TRANSITION`)

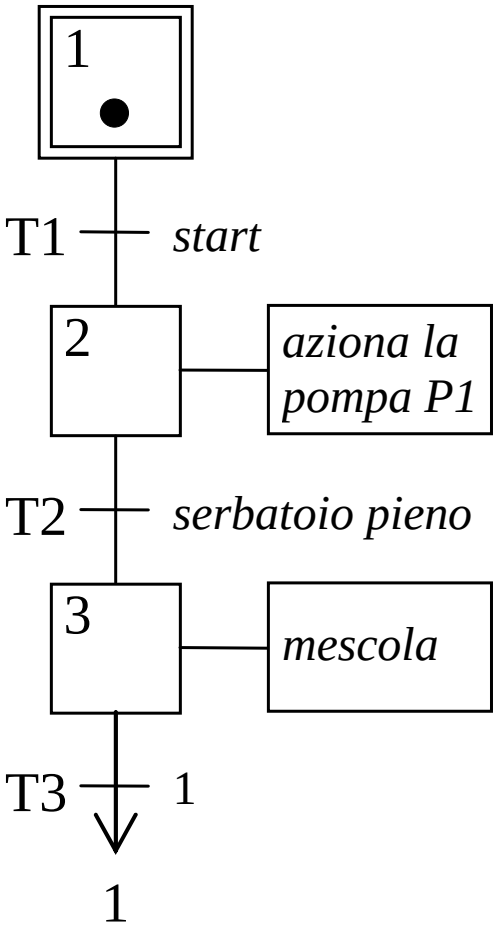
Fasi e transizioni sono collegati da *archi orientati*, che definiscono la sequenza tra le fasi:

- le fasi possono essere collegate solo a transizioni e le transizioni solo a fasi
- per convenzione gli archi vanno rappresentati con segmenti collegati con angoli retti
- si omette la freccia direzionale se sono diretti dall’alto verso il basso
- un arco orientato uscente da una transizione può essere seguito anche da un identificativo dello stato, configurando un collegamento indiretto a tale stato (jump)

Nel caso di definizione testuale del diagramma SFC, la topologia del diagramma si specifica nel costrutto `TRANSITION` che diventa:

```
TRANSITION FROM fasi_precedenti
           TO fasi_successive
       := ... ; (* condizione associata *)
END_TRANSITION
```

# Esempio




Schema funzionale costituito da 3 fasi, numerate da 1 a 3, e 2 transizioni, T1 e T2.

Fasi:

- la fase 1 è quella iniziale, di pura attesa (nessuna azione associata)
- quando è attiva la fase 2, viene eseguita l’azione associata chiamata, informalmente, *aziona la pompa P1*
- quando è attiva la fase 3, viene eseguita l’azione associata chiamata, informalmente, *mescola*.

Transizioni:

- T1 è a valle della fase 1 ed è associata ad una variabile di ingresso chiamata *start*
- a T2, a valle della fase 2, è associata una condizione logica chiamata, informalmente, *serbatoio pieno*
- a T3 è associata una condizione sempre vera

Il sistema è inizialmente in fase di attesa.

Quando riceve il comando *start*, attiva la sequenza, azionando la pompa P1.

La pompa viene disattivata al raggiungimento della condizione *serbatoio pieno*, dove viene attivata l’azione *mescola*.

Successivamente il sistema torna nello stato di attesa.

L’SFC rappresenta il comportamento della parte di comando del sistema automatizzato: esso realizza le funzioni del controllore.

I suoi ingressi sono le misure dei sensori del campo e le sue uscite sono i valori da inviare agli attuatori.

L’SFC rappresenta il *comportamento desiderato* del processo.

Possibile definizione testuale dell’SFC dell’esempio:

```
INITIAL_STEP quiete
:
END_STEP
STEP riempimento
: aziona la pompa
END_STEP
STEP pieno
: mescola
END_STEP
TRANSITION FROM quiete TO riempimento
:= start;
END_TRANSITION
TRANSITION FROM riempimento TO pieno
:= serbatoio pieno;
END_TRANSITION
TRANSITION FROM pieno TO quiete
:= ;
END_TRANSITION
```

# Regole di evoluzione

Occorre definire senza ambiguità il comportamento dinamico, con opportune *regole di evoluzione*.

Le regole di evoluzione di un SFC si basano sulle condizioni logiche in cui si trovano le transizioni dello schema.

- una transizione si dice *abilitata* se tutti le fasi a monte sono attive
- una transizione si dice *superabile* se è abilitata e la sua ricettività assume il valore logico vero (si dice anche che la transizione può scattare)

Regole:

❶ quando una transizione è superabile, essa viene effettivamente superata (tutte le fasi a monte vengono disattivate e tutte le fasi a valle vengono attivate)

❷ se più transizioni sono superabili in un certo istante, vengono superate tutte contemporaneamente (modello *sincrono*)

❸ se la condizione associata alla transizione di uscita di una fase è già vera quando la fase viene attivata (fase *instabile*), le azioni ad essa associate vanno comunque eseguite prima della disattivazione della fase

# Strutture di programmazione

In GRAFCET è possibile rappresentare due strutture tipiche dei sistemi ad eventi:

- la *scelta*
  (operatori di tipo OR, OR-divergenza e OR-convergenza)
- il *parallelismo*
  (operatori di tipo AND, AND-divergenza e AND-convergenza)

Altre strutture comunemente utilizzate:

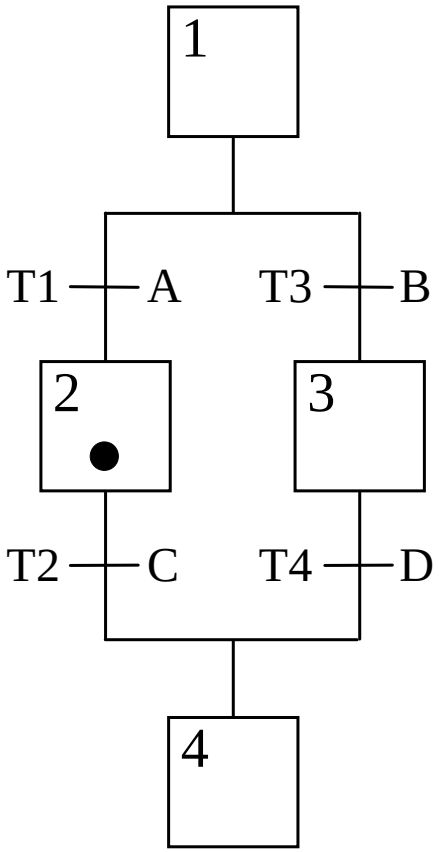
- salto di sequenza
- ripetizione di sequenza
- semaforo di mutua esclusione
- sincronizzazione locale
- rendez-vous

# Scelta

Costrutto di tipo IF-THEN-ELSE.

Si usano due nodi diramatori:

- OR-divergenza
- OR-convergenza



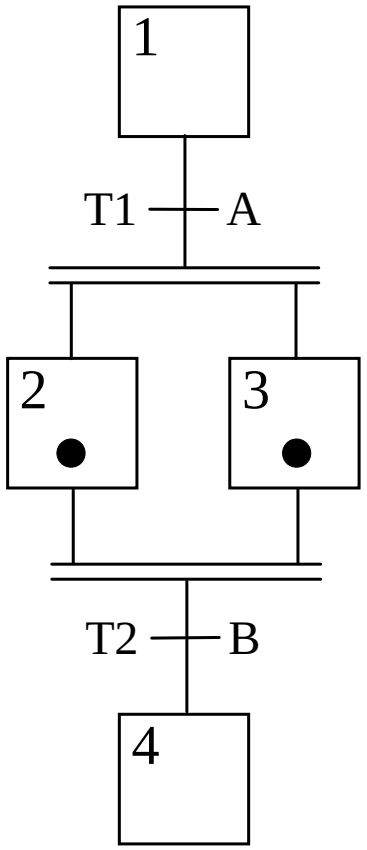

OR-divergenza
fase seguita da più transizioni
con condizioni *mutuamente esclusive*

OR-convergenza
più sequenze terminano nella stessa
fase attraverso transizioni diverse

NB. Collegamenti tra fasi di tipo fork o join, con una transizione per ramo.

# Parallelo

Attivazione e disattivazione simultanea di più fasi.




Si usano due nodi diramatori (con barre doppie):

- AND-divergenza (concorrenza)
- AND-convergenza (sincronizzazione)

AND-divergenza
transizione seguita da più fasi

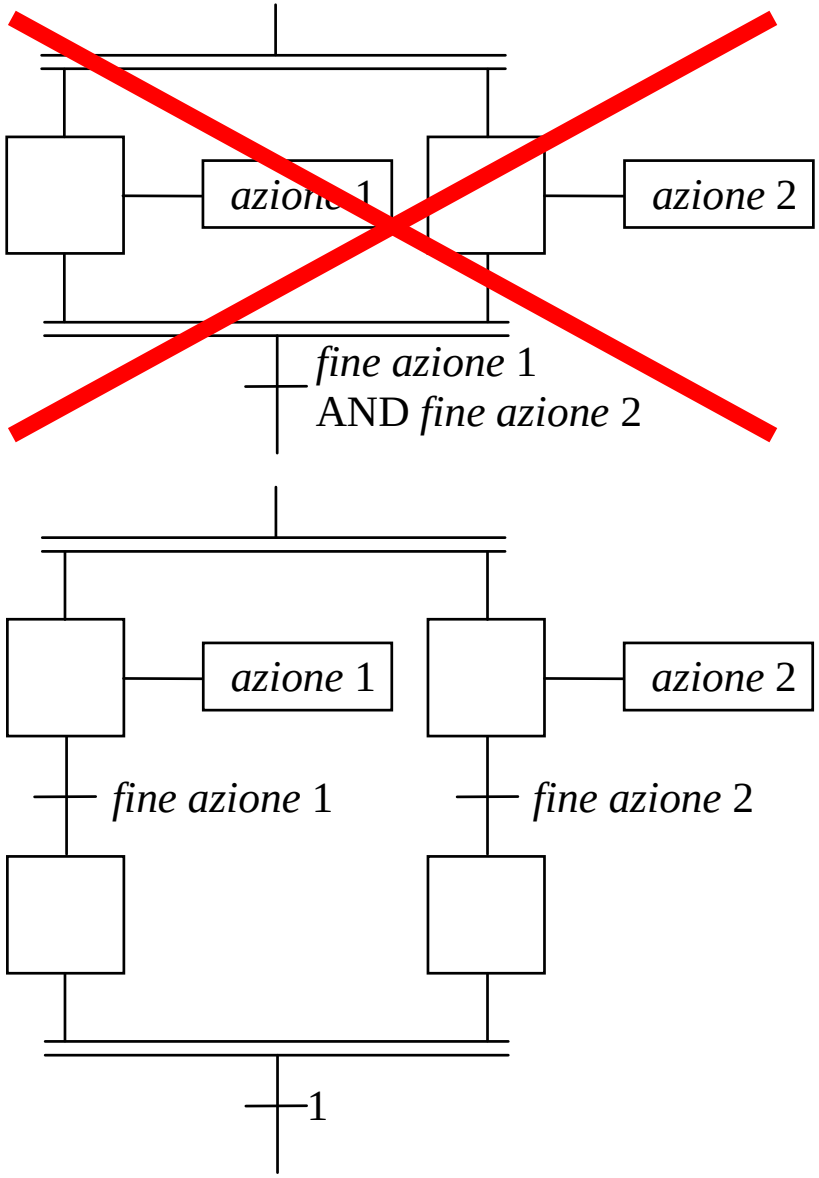
AND-convergenza
più fasi precedono la stessa transizione
(superabile se tutte le fasi sono attive)

NB. Collegamenti tra fasi di tipo fork o join, con *una* transizione per collegamento.

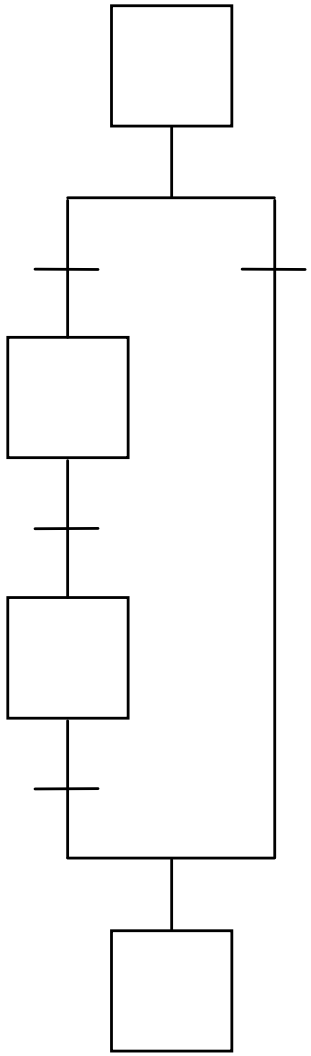
Tipicamente, quando si svolgono più operazioni in parallelo, accade che non tutte terminino simultaneamente.

In generale, occorre evitare di mettere in AND le condizioni di fine delle operazioni coinvolte, perché ciò causa il protrarsi di un’operazione oltre la durata prevista.

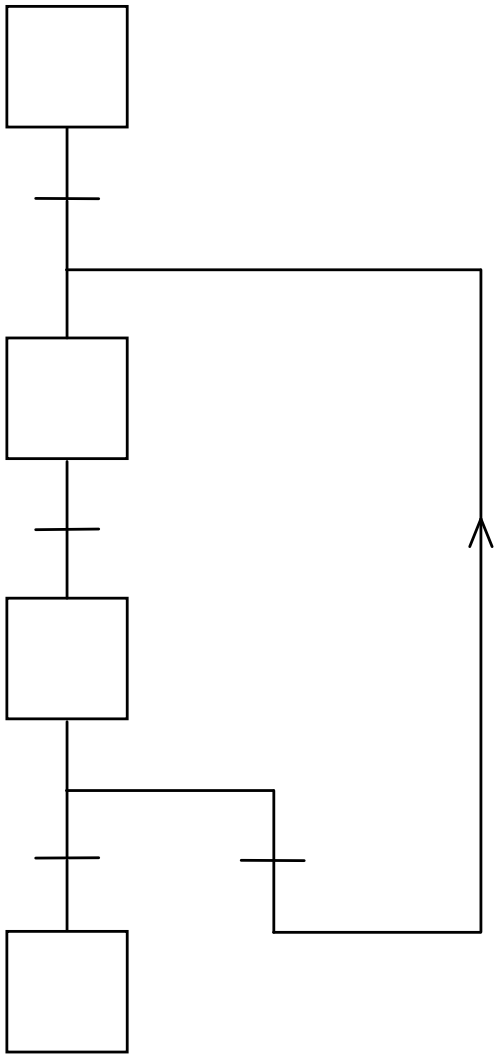
Un modo semplice per risolvere il problema consiste nell’inserire degli stati *dummy*, di pura attesa, a valle delle singole fasi in parallelo, seguiti da una transizione di sincronizzazione immediatamente superabile (condizione associata sempre vera).



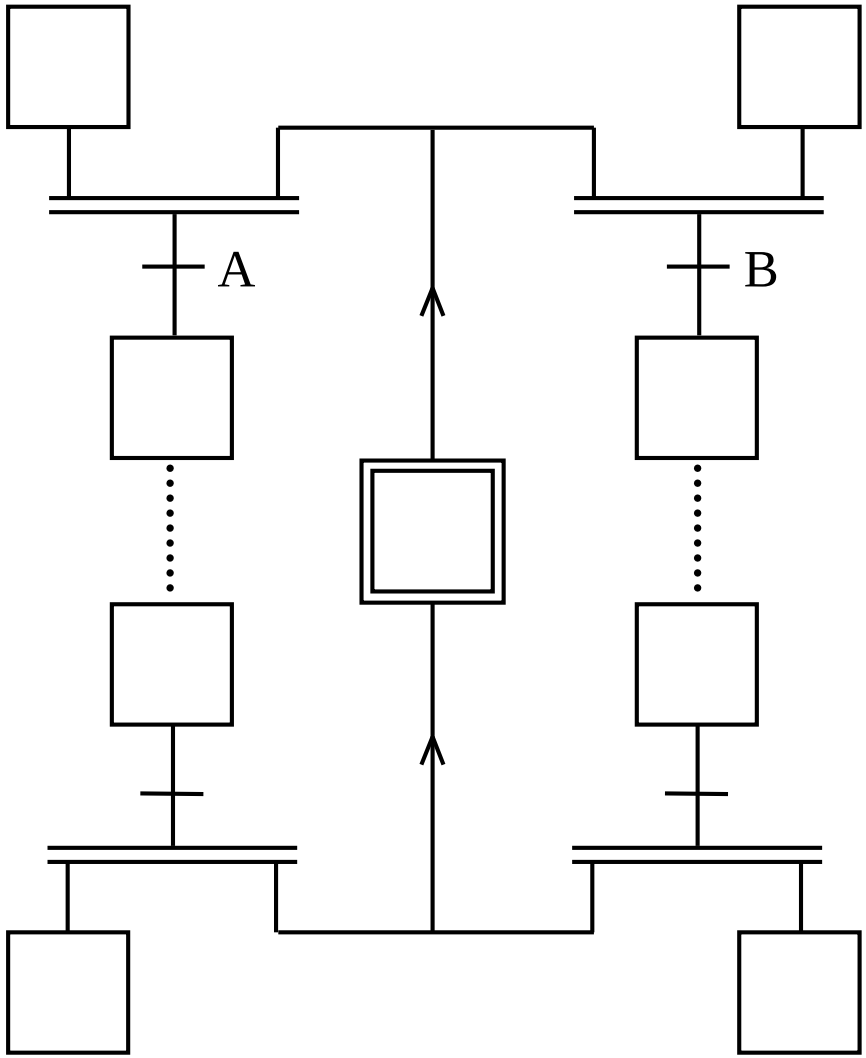
# Salto di sequenza



# Ripetizione di sequenza



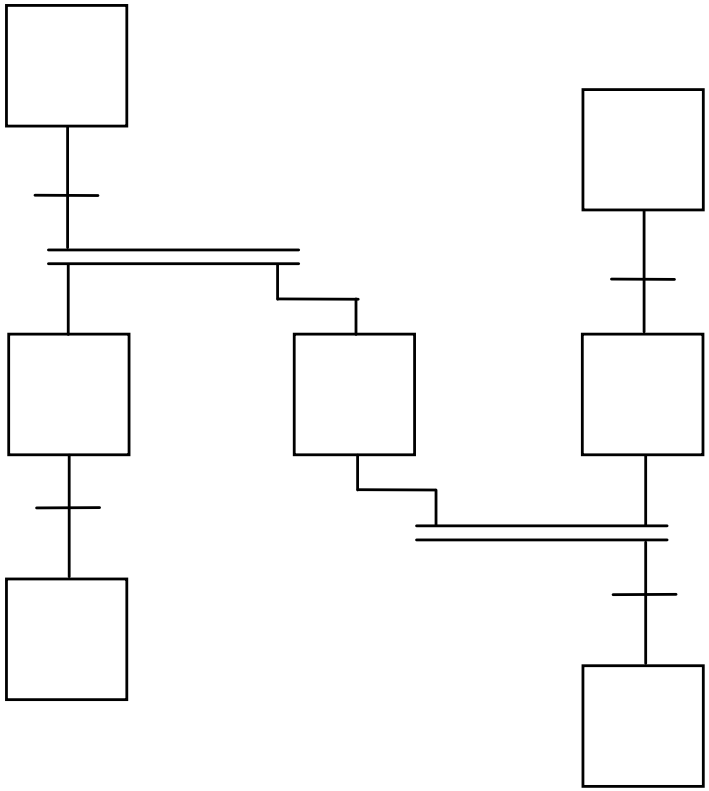
# Semaforo di mutua esclusione




Le due condizioni, A e B, devono essere mutuamente esclusive.

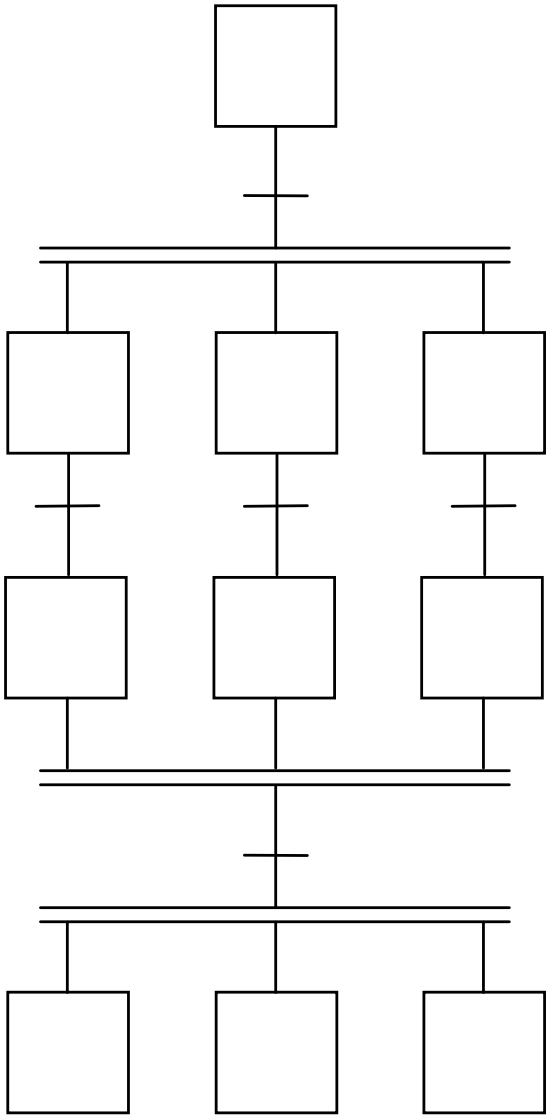
La fase del semaforo deve essere inclusa tra le fasi iniziali (risorsa condivisa disponibile).

## Sincronizzazione locale



La sequenza di destra deve attendere la sequenza di sinistra.

## Rendez-vous



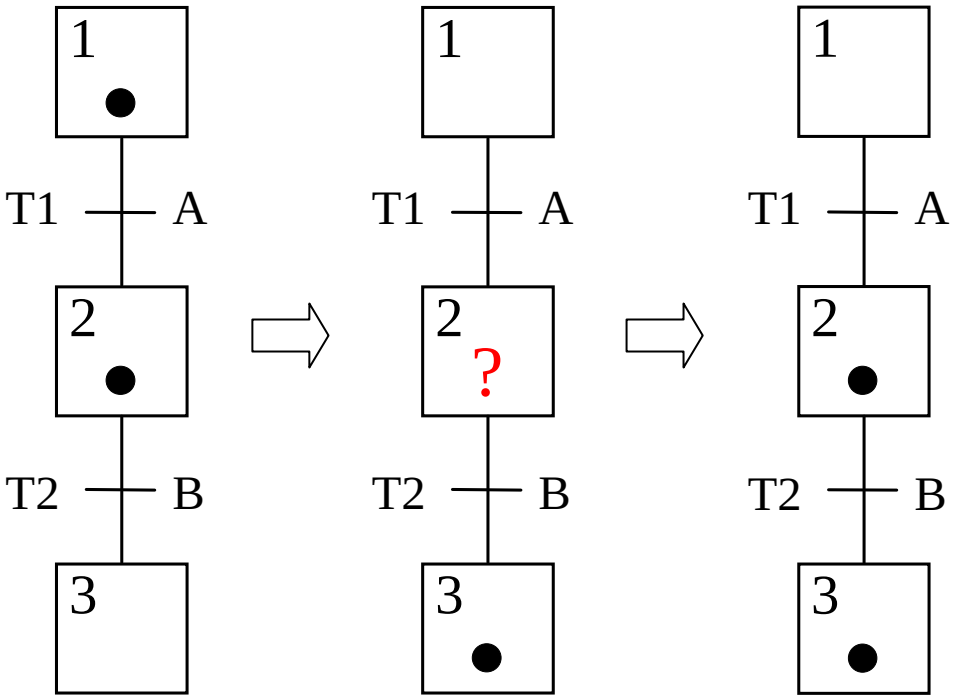
# Situazioni e costrutti da evitare

- disattivazione e attivazione contemporanea di una fase
- scatto contemporaneo di transizioni a valle di un nodo di OR-divergenza
- divergenza chiusa da una convergenza di tipo diverso

# Disattivazione e attivazione contemporanea di una fase

Può accadere che una fase debba essere contemporaneamente disattivata e attivata.

Una situazione del genere può essere indice di confusione nell’evoluzione dello schema e andrebbe generalmente evitata.




Le fasi 1 e 2 sono attive.

Se le condizioni logiche associate alle due transizioni sono entrambe vere, queste ultime sono entrambe superabili.

Per superare la transizione T1 bisogna disattivare la fase 1 e attivare la fase 2.

Per superare la transizione T2 bisogna disattivare la fase 2 e attivare la fase 3.

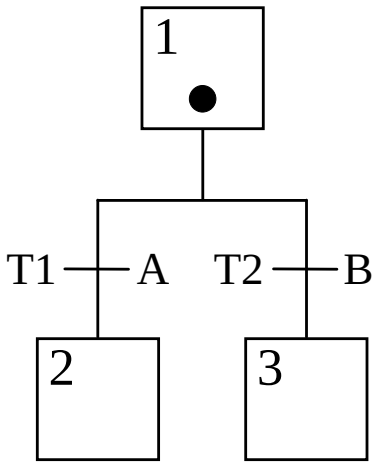
Potenziale confusioni:

- la fase 2 è attiva o inattiva?

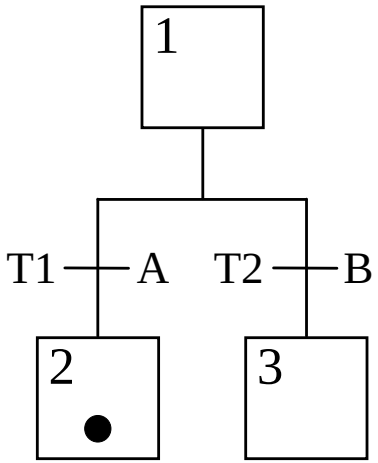
  per convenzione la si assume attiva
- se A = TRUE ma B = FALSE, si passa da uno stato con 2 gettoni (in 1 e 2) ad uno con un solo gettone (in 2)

Nonostante la possibile ambiguità del costrutto, esso è talvolta utilizzato di proposito per modellizzare p.es. le varie porzioni successive di un nastro trasportatore.

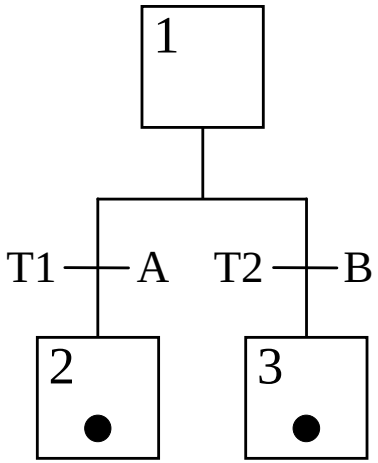
# Scatto contemporaneo di transizioni a valle di un nodo di OR-divergenza




A = TRUE
B = FALSE




evoluzione corretta (viene attivato uno solo dei due rami)



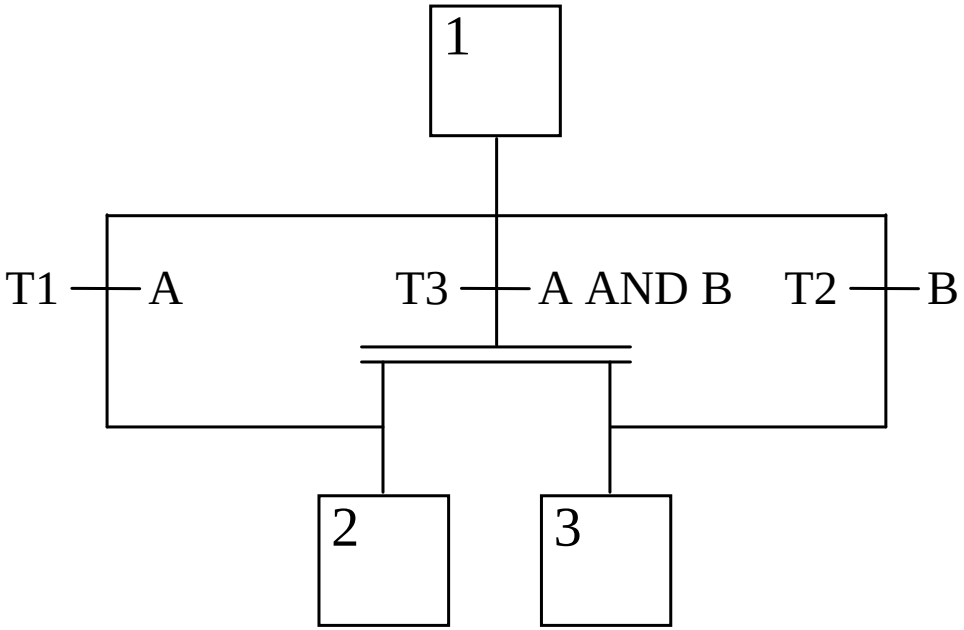

evoluzione scorretta (entrambi i rami vengono attivati)

A = TRUE
B = TRUE

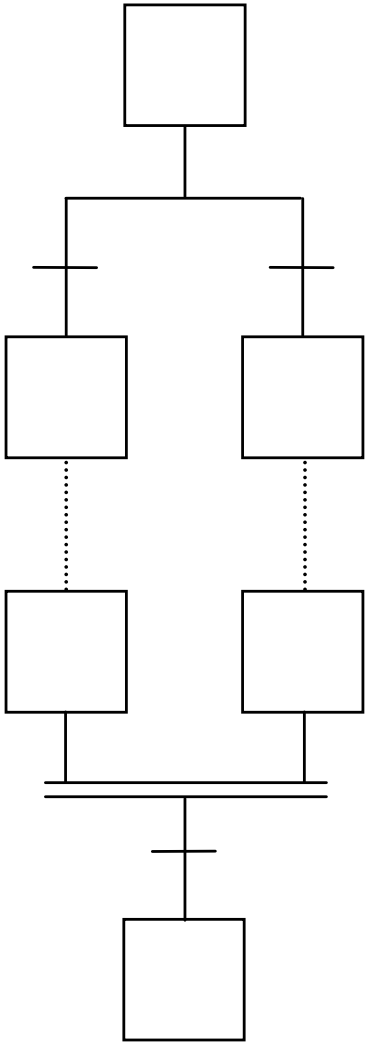
Per evitare questo genere di problemi occorre imporre esplicitamente che le condizioni associate alle transizioni di una scelta siano *mutuamente esclusive*:

- $\tilde{B}$ = NOT(A)
- $\tilde{B}$ = B AND NOT(A) (priorità alla transizione di sinistra → se A e B sono entrambe vere, solo la transizione di sinistra risulta superabile)

In realtà, il secondo tipo di comportamento non è vietato, ma per evitare ambiguità di sintassi conviene utilizzare una struttura diversa che esprima esplicitamente tale possibilità:

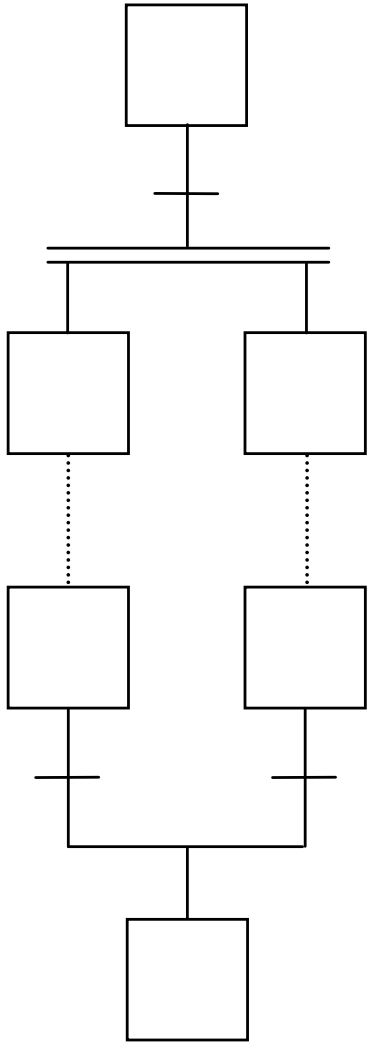


# Divergenza chiusa da una convergenza di tipo diverso



OR-divergenza seguito da AND-convergenza

In tal caso, poiché solo una sequenza sarà attiva, la transizione a valle del nodo di sincronizzazione non sarà mai superabile



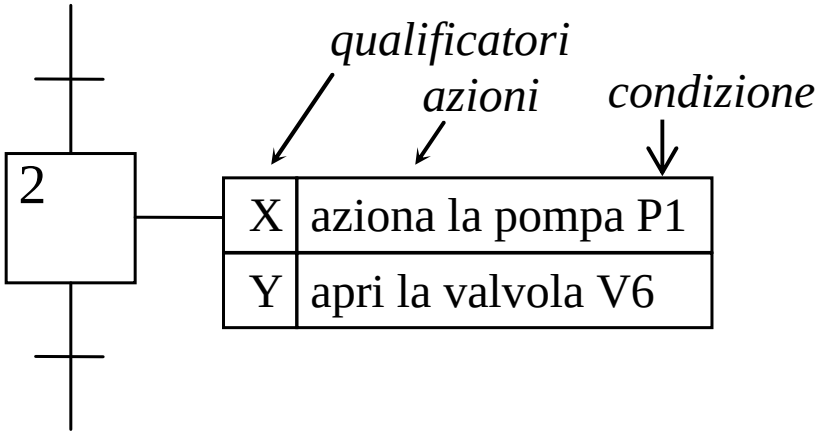
AND-divergenza seguito da OR-convergenza

In tal caso, la fase a valle del nodo di convergenza può essere attivata più di una volta, generando ambiguità di interpretazione

# Azioni

Le *azioni* servono per esplicitare i comandi da emettere e sono associate alle fasi.

Una fase può avere più azioni, eseguite in *parallelo* (attenzione però a eseguire in parallelo azioni di durata differente!).




Oltre che graficamente, un’azione può essere descritta con un costrutto software specifico:

- `ACTION nome_azione :`
  `... `*corpo dell’azione*` ...`
  `END_ACTION`

Il corpo dell’azione può essere specificato in uno qualunque dei linguaggi previsti dalla normativa 61131-3.

Le azioni possono essere condizionate (un’azione condizionata viene eseguita se – e fintanto che – la fase corrispondente è attiva e la condizione associata è vera).

# Modalità di esecuzione di un’azione

Di default, un’azione è attiva per tutto il tempo in cui la fase corrispondente è attiva.

Tuttavia, non è detto che questo sia il modo corretto di eseguire un’operazione o comandare un attuatore.

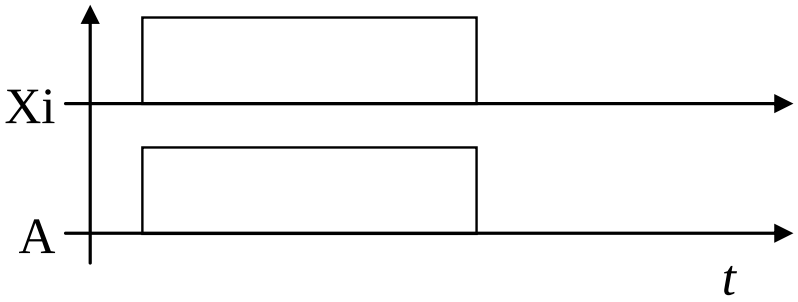
Per questo motivo, le azioni sono dotate di attributi (*qualificatori*) per descrivere in modo non ambiguo l’andamento temporale delle corrispondenti variabili d’uscita:

- N normal o non-stored, azione continua
- L time-limited, azione limitata nel tempo
- D delayed, azione ritardata nel tempo
- P pulse, azione di tipo impulsivo
- S stored o set, azione memorizzata
- R reset, azione cancellata
- SD stored and time-delayed, azione memorizzata e ritardata
- DS delayed and stored, azione ritardata e memorizzata
- SL stored and time-limited, azione memorizzata e limitata nel tempo

*Qualificatore N* (*normal o non-stored, azione continua*)

L’azione (A) termina quando la fase (Xi) diventa inattiva.

Essa viene emessa in modo continuo nel tempo fintantoché la fase associata rimane attiva.



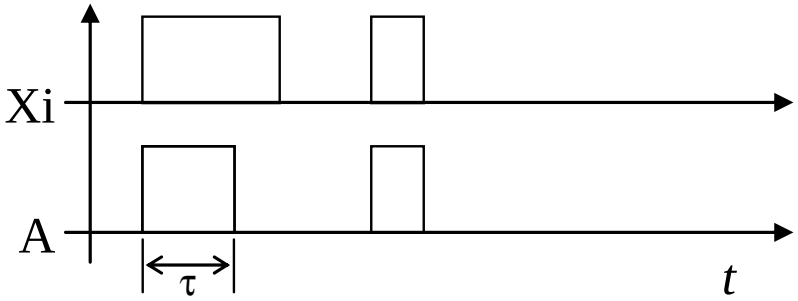

Se due o più fasi successive sono associate alla stessa azione, questa non viene interrotta durante le transizioni.

Come detto, è la modalità operativa di default.

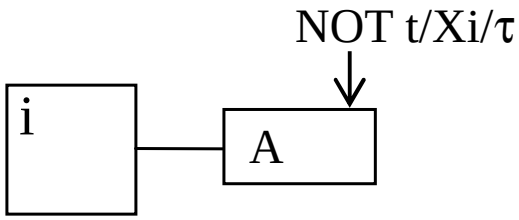
Se il qualificatore di azione è omesso, si assume che l’azione sia continua.

## *Qualificatore L* (*time-limited*, *azione limitata nel tempo*)

L’azione comincia quando la fase diventa attiva e termina quando la fase diventa inattiva o quando è trascorso un certo intervallo di tempo.



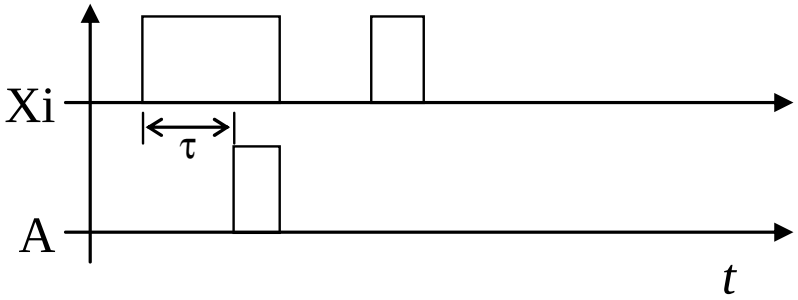

L’azione limitata nel tempo è un caso particolare di azione condizionata, in cui la condizione è una variabile temporale negata associata alla stessa fase a cui è associata l’azione.



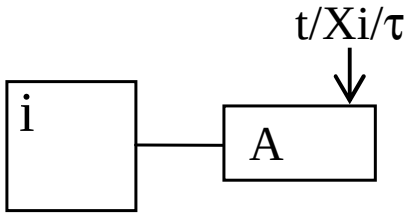

Ad esempio, certe elettrovalvole devono essere comandate con impulsi di durata fissata.

## *Qualificatore D* (*delayed*, *azione ritardata nel tempo*)

L’azione comincia τ istanti dopo che la fase è diventata attiva, se esso lo è ancora in quel momento, e termina quando la fase diventa inattiva.




L’azione ritardata nel tempo è un caso particolare di azione condizionata, in cui la condizione è una variabile temporale associata alla stessa fase a cui è associata l’azione.



*Qualificatore P* (*pulse*, *azione di tipo impulsivo*)

L’azione comincia quando la fase diventa attiva (o disattiva, secondo le opzioni), ovvero sul fronte di salita (o di discesa) della variabile associata allo stato della fase, e viene eseguita una volta sola, indipendentemente dalla durata della fase.

Serve ad esempio per:

- un calcolo aritmetico
- l’attivazione di un temporizzatore
- l’aggiornamento di un contatore

*Qualificatore S* (*stored* o *set*, *azione memorizzata*)

L’azione comincia quando la fase diventa attiva e rimane attiva finché non viene resettata con un’azione uguale di tipo R, indipendentemente dalla durata della fase. Si noti che più fasi potrebbero avere la stessa azione con il qualificatore S.

*Qualificatore R* (*reset*, *azione cancellata*)

Fa terminare un’azione memorizzata. Le azioni associate a qualificatori S ed R vengono eseguite anche se la durata di attivazione della fase è di un solo ciclo.

*Qualificatore SD* (*stored and time-delayed*, *azione memorizzata e ritardata*)

L’azione comincia $\tau$ istanti dopo che la fase è diventata attiva, anche se essa nel frattempo è diventata inattiva.

L’azione rimane attiva finché non viene resettata con un’azione uguale di tipo R.

*Qualificatore DS* (*delayed and stored*, *azione ritardata e memorizzata*)

L’azione comincia $\tau$ istanti dopo che la fase è diventata attiva, se essa lo è ancora in quel momento.

Se attivata, l’azione rimane attiva finché non viene resettata con un’azione uguale di tipo R.

*Qualificatore SL* (*stored and time-limited*, *azione memorizzata e limitata nel tempo*)

L’azione comincia quando la fase diventa attiva e termina quando viene resettata o comunque quando è trascorso un certo intervallo di tempo.

# Le condizioni logiche associate alle transizioni

Le condizioni associate alle transizioni sono espressioni logiche generali, che possono includere variabili di ingresso, di stato, temporali e “condivise”.

Variabili di ingresso:

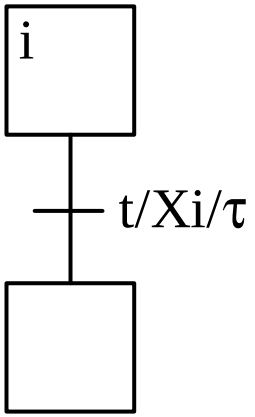
- possono essere *booleane* (p.es. presenza_pezzo) o *reali* (p.es. pressione o temperatura)
- le variabili booleane possono essere *semplici* (X, stesso valore binario del segnale del sensore) o *a fronte* ($\uparrow$X, valgono 1 quando il segnale del sensore commuta da 0 a 1)
- esempio di condizione logica:
  (TEMPERATURA $\leq$ 25°) AND NOT PRESENZA_PEZZO
- attenzione a mettere più clausole logiche in AND sulla medesima condizione!

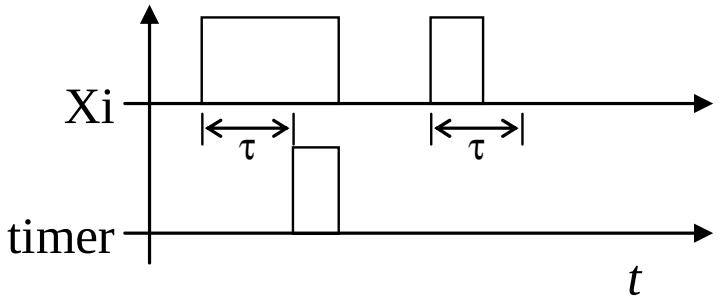
Variabili di stato:

- sono delle variabili logiche (Xi) associate allo stato di attivazione delle fasi (Xi è associata alla fase *i*-esima)
- le variabili di stato, una volta definite, possono comparire nelle condizioni associate a qualunque transizione dello schema

# Variabili temporali:

- il tempo è modellizzato tramite *temporizzatori* (*timer*), associati alle fasi, la cui uscita vale inizialmente 0, rimane a 0 quando si attiva la fase associata, e commuta a 1 (se e) quando è trascorso un intervallo di tempo $\tau$ dall’attivazione della fase




- una variabile temporale è indicata con t/Xi/$\tau$, dove Xi è la fase a cui si riferisce e $\tau$ è l’intervallo temporale associato al timer
- il sistema operativo definisce automaticamente per ogni fase una variabile di tipo `TIME`, che rappresenta
  - il tempo trascorso dalla attivazione della fase, se la fase è attiva
  - la durata dell’ultima attivazione della fase, se la fase è inattiva

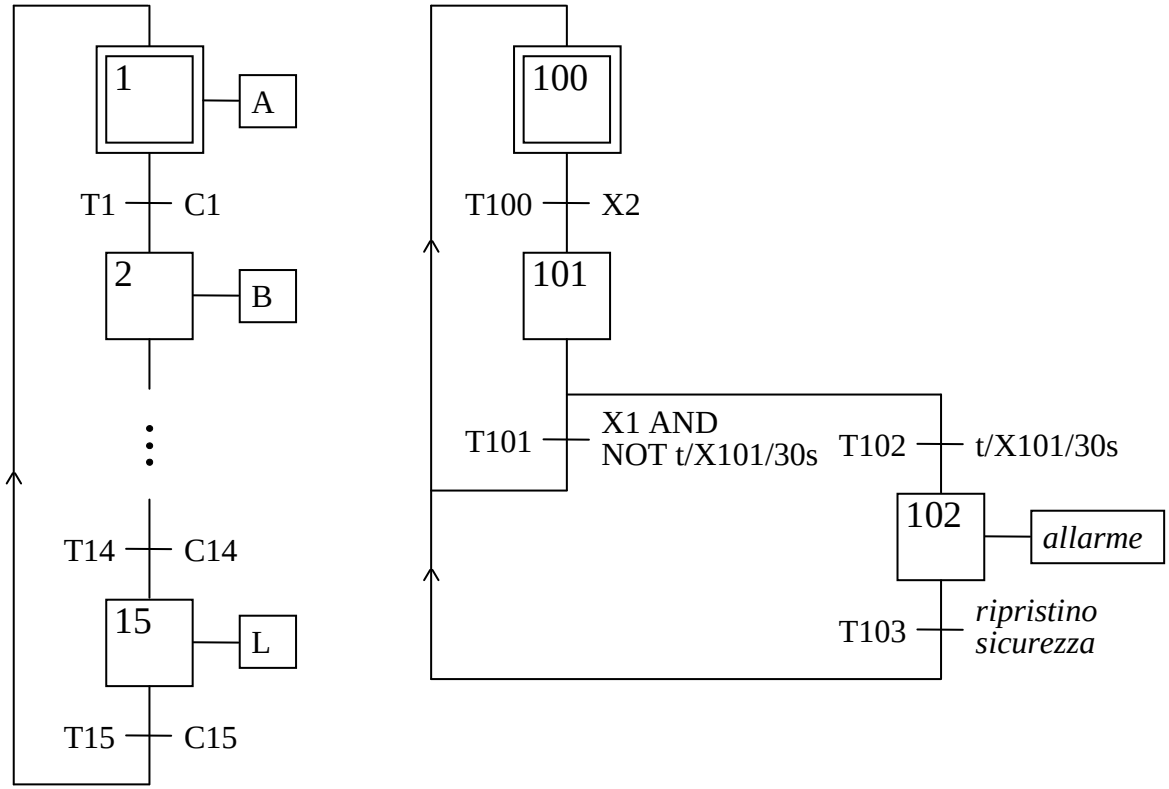
  tale variabile è utilizzabile, ma non modificabile dall’utente

  se alla fase è stato assegnato un nome, tale variabile è indicata con `nome_fase.T`

## Variabili “condivise”:

- sono variabili definite in una zona di memoria condivisa, visibile da tutte le porzioni di SFC definite
- è possibile definire un’azione fittizia che non agisce su una variabile di uscita, ma su una variabile interna;
  tale variabile può poi essere usata nella condizione logica associata a qualunque transizione
- ad esempio, in una fase è possibile incrementare un contatore e in una transizione leggere il valore corrente del contatore

# Esempio uso variabili di stato e temporali: watchdog




L’SFC di destra realizza un watchdog per il ciclo di produzione vero e proprio (SFC di sinistra).

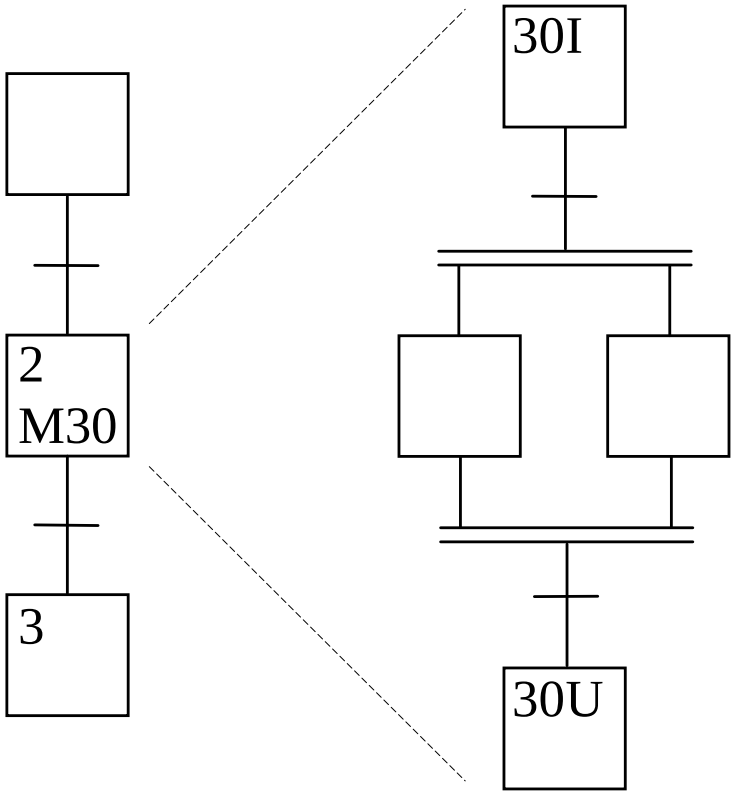
Se il ciclo non viene completato entro 30 secondi viene generato un allarme.

# Strutture complesse: gerarchia e modularità

Si può implementare una struttura gerarchica utilizzando diversi tipi di costrutti:

- macrofasi
- sincronizzazione implicita
- macroazioni

# Macrofasi




Le macrofasi (o azioni aggregate) sono azioni complesse costituite dalla combinazione di azioni semplici o aggregate e funzionano come i sotto-programmi.

Quando si attiva la fase associata ad un’azione aggregata, essa comincia ad evolvere e termina quando la fase si disattiva.

Problematiche di implementazione:

- cosa succede alle azioni elementari presenti nell’espansione della fase 2, quando questa si disattiva?
- cosa succede alle azioni elementari presenti nell’espansione della fase 2, se la transizione a valle della fase è superabile quando questo si attiva?
- come vanno gestite le variabili locali dell’azione aggregata?
- come vanno gestite le azioni eventualmente settate (con un qualificatore di tipo S, SD, DS, SL)?

Questi dettagli non sono standardizzati, ma vanno comunque chiariti definendo opportunamente la regola di evoluzione.

Regola di evoluzione (v. Chiacchio):

- se deve essere attivata la fase 2, associata alla macrofase M30, si attiva in realtà la fase 30I dell’espansione
- l’evoluzione continuerà sull’SFC di espansione fino ad arrivare alla fase 30U da cui riprenderà l’SFC originario dalla fase 3 (quando la condizione associata alla transizione a monte di tale fase risulterà verificata)

## Sincronizzazione implicita

Un altro metodo per implementare delle gerarchie tra schemi SFC si basa sull’uso di variabili condivise o di stato per condizionare l’evoluzione di uno schema sulla base dello stato corrente di un altro schema.

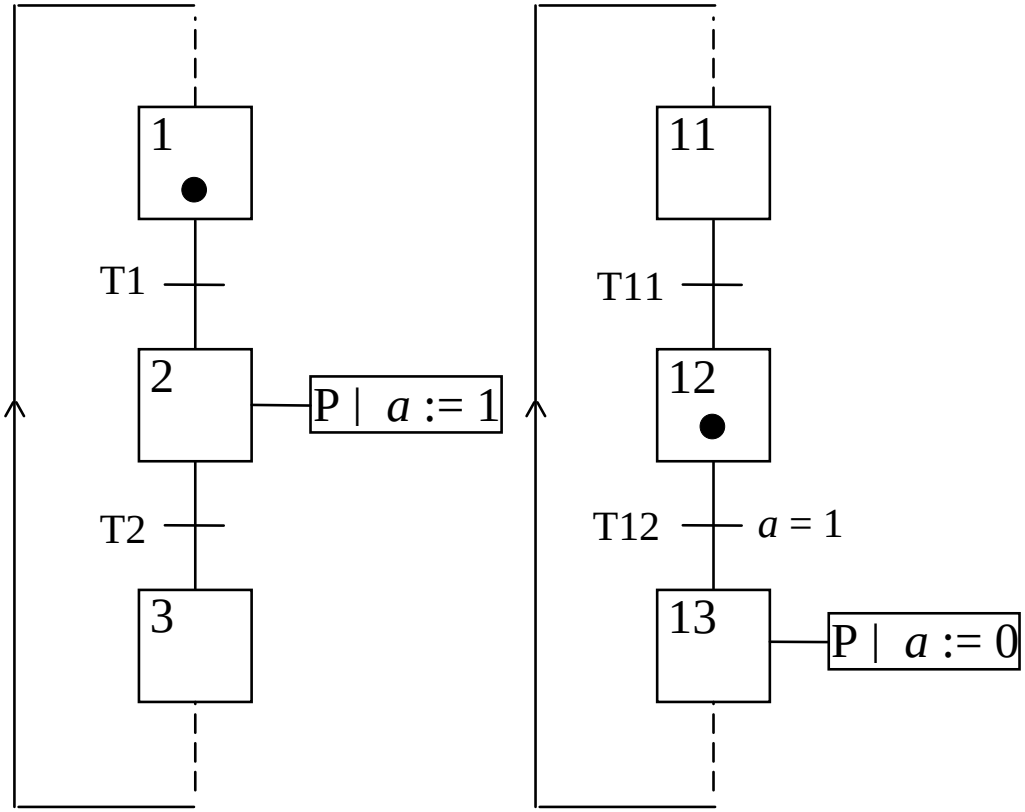
Sincronizzazione mediante variabili condivise:

- uno schema SFC setta una variabile binaria nell’azione associata ad una fase Xi
- un altro schema ha la stessa variabile nella condizione associata ad una transizione
- per evitare che l’evento venga consumato più volte, il secondo schema SFC dovrà anche resettare la medesima variabile nell’azione associata ad una fase a valle della transizione

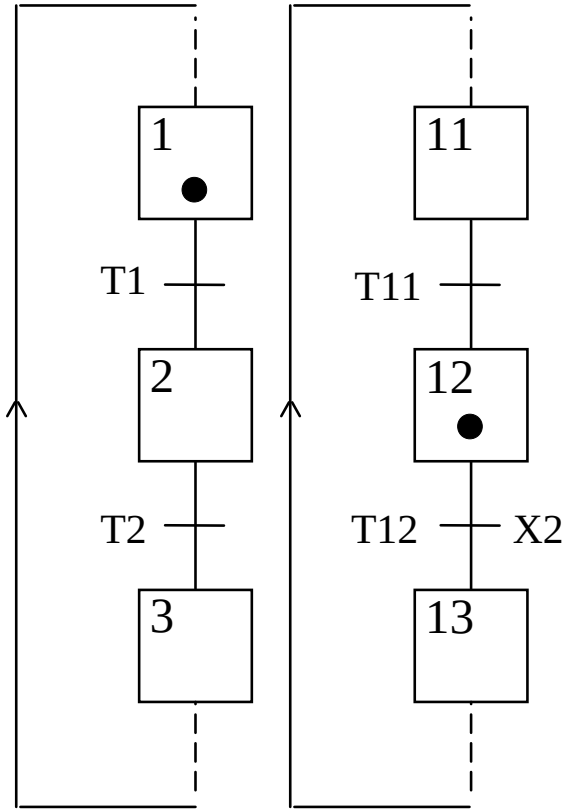
Sincronizzazione mediante variabili di stato:

- nel secondo schema SFC la condizione associata alla transizione è espressa in termini della fase Xi del primo schema
- tale modalità di sincronizzazione evita l’introduzione di variabili aggiuntive, ma occorre prestare attenzione quando si modifica uno schema SFC: se si modifica la numerazione degli stati occorre correggere anche i riferimenti corrispondenti

## Sincronizzazione mediante variabili condivise:

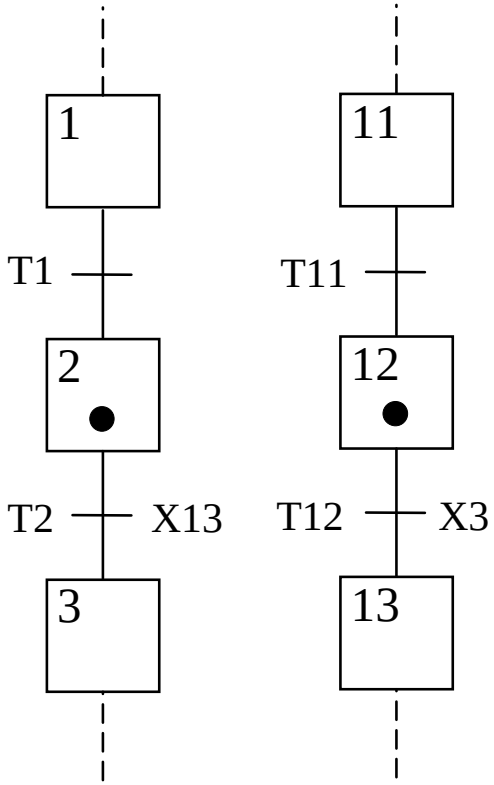



## Sincronizzazione mediante variabili di stato:




Il superamento della transizione T12 dipende dallo stato di attivazione della fase 2. L’evoluzione dello schema di destra si blocca in attesa di quello di sinistra.

Occorre prestare particolare attenzione all’uso di riferimenti incrociati, che non devono essere posizionati in modo tale da determinare un circolo vizioso nell’evoluzione del sistema (con conseguente deadlock).




T12 è condizionata all’attivazione della fase 3, mentre T2 è condizionata all’attivazione della fase 13.

Quindi, poiché la fase 3 non si attiva se non scatta T2 e la fase 13 non si attiva se non scatta T12, *lo schema è bloccato* e non è in grado di evolvere!

L’SFC non è uno strumento pensato per l’analisi e la verifica dei modelli (come le reti di Petri). Pertanto, individuare i deadlock non è immediato.

Tuttavia se il modello è derivato direttamente da un modello a reti di Petri, la presenza di deadlock può essere evidenziata analizzando quest’ultimo.

# Macroazioni

Sempre usando variabili interne è possibile *forzare*, *sospendere* o *bloccare* l’esecuzione di uno schema SFC all’attivazione di una fase di un altro SFC.

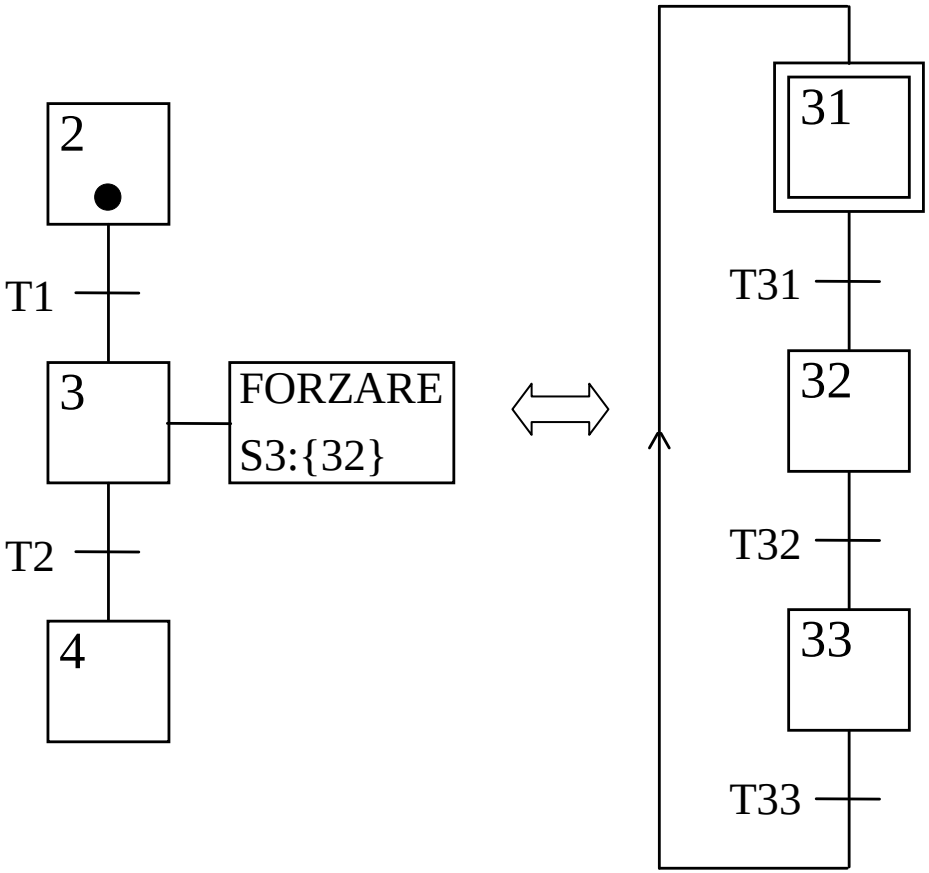
Per semplificare la notazione grafica questi costrutti sono realizzati per mezzo di *macroazioni*.

La macroazione è un’azione operata da un SFC che ha effetti sulla condizione di un altro SFC.

Tipi di macroazioni:

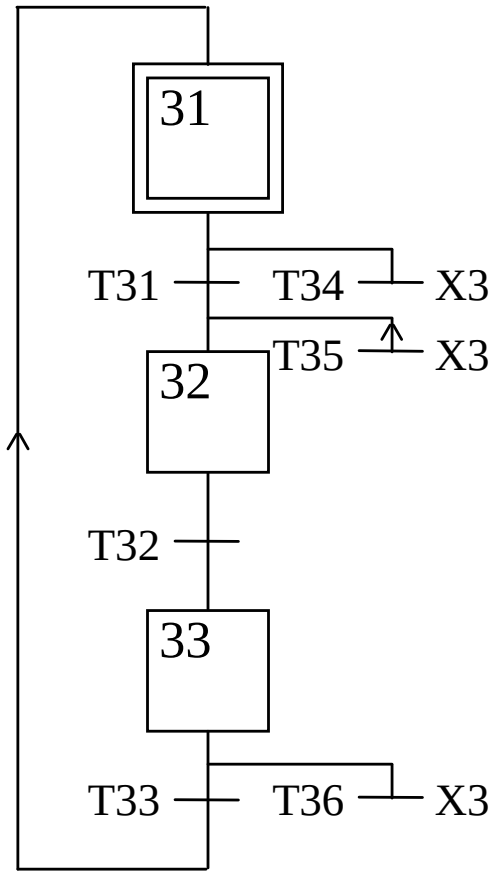
- forzatura
- sospensione
- bloccaggio

# Macroazione di forzatura




La macroazione del primo schema forza lo stato del secondo schema, attivando la fase 32.

## Schema equivalente:




La macroazione è equivalente ad utilizzare la variabile interna X3 come nello schema a lato.

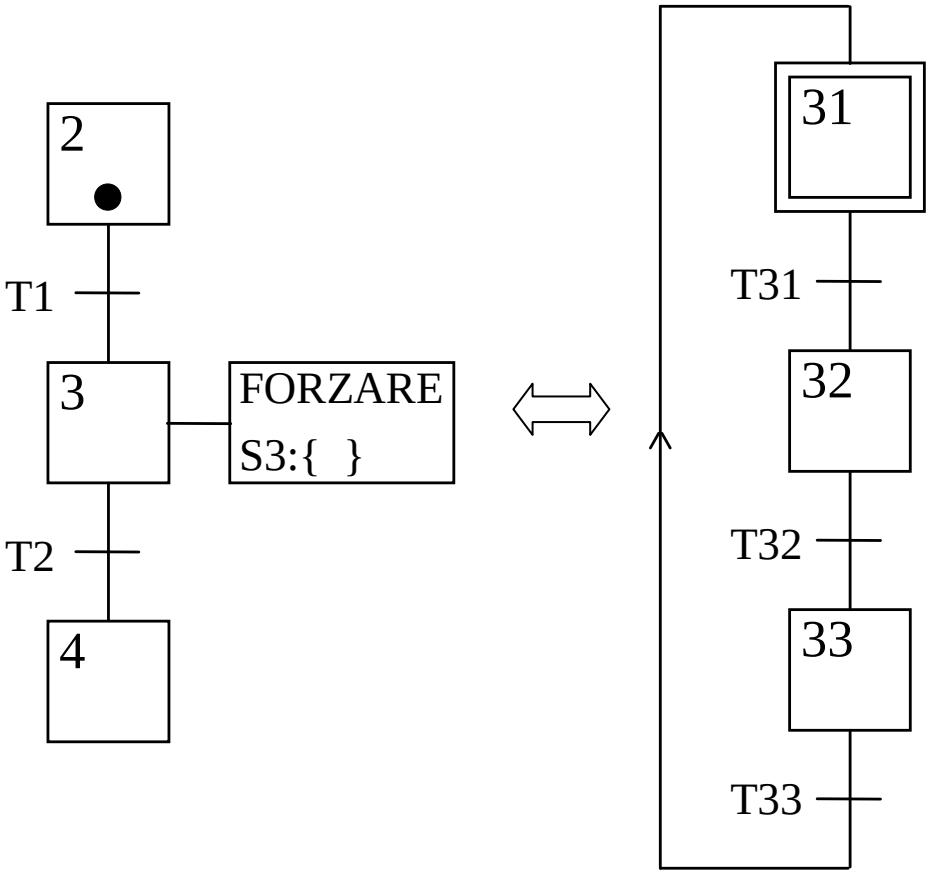
Le transizioni T34, T35, T36 sono superabili solo se la fase 3 è attiva.

In particolare, la transizione T35 non richiede altre condizioni per scattare, mentre T34 e T36 richiedono l’attivazione delle fasi 31 e 33, rispettivamente.

Quando è attiva la fase 3 nel primo schema, lo stato del terzo schema viene resettato:

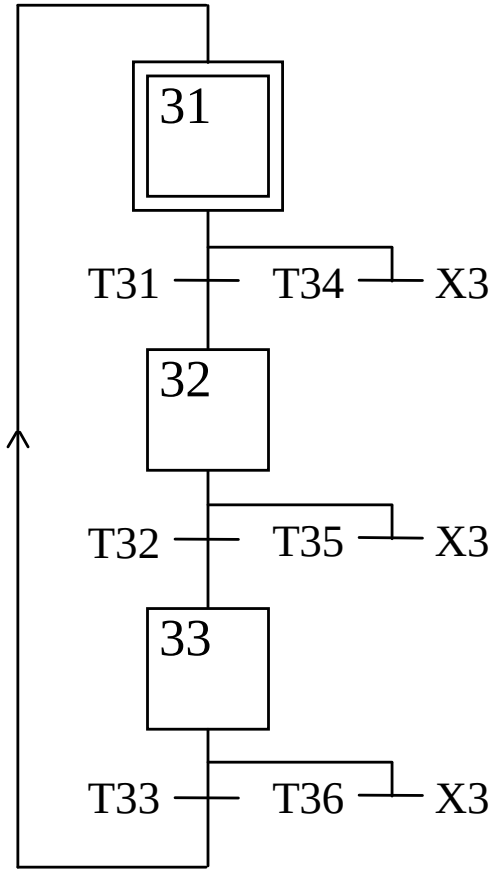
- scatta T35 e attiva la fase 32
- se la fase 31 è attiva, si disattiva per effetto dello scatto di T34
- se la fase 33 è attiva, si disattiva per effetto dello scatto di T36

# Macroazione di sospensione




Se la macroazione di forzatura ha una condizione vuota, l’effetto è quello di disattivare tutte le fasi dell’SFC forzato, sospendendone così l’esecuzione.

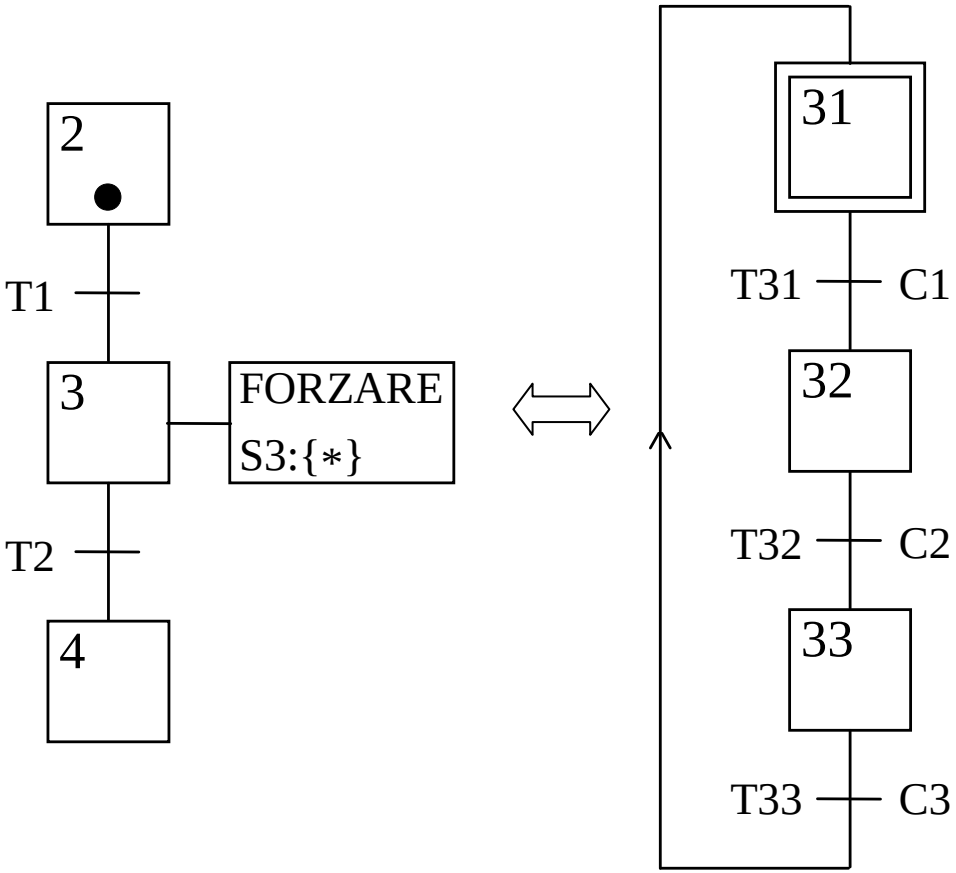
# Schema equivalente:




La macroazione è equivalente ad utilizzare la variabile interna X3 come nello schema a lato.

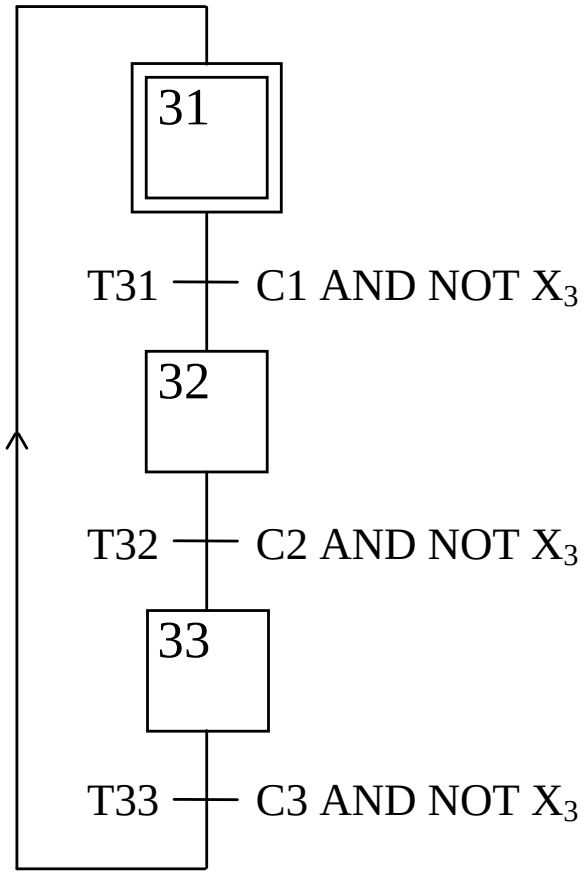
Le transizioni T34, T35, T36 diventano superabili se la fase 3 è attiva, disattivando così tutte le fasi dell’SFC.

# Macroazione di bloccaggio




Se la macroazione ha un simbolo ‘*’ nella condizione, impedisce lo scatto di qualunque transizione dell’SFC forzato, senza cambiarne lo stato corrente.

# Schema equivalente:




La macroazione è equivalente ad utilizzare la variabile interna X3 come nello schema a lato.

Le transizioni T31, T32, T33 sono superabili solo se la fase 3 non è attiva.

Tutte le volte che è attiva la fase 3 nel primo schema, lo stato del secondo schema si congela nello stato corrente.