

Il linguaggio SFC (Sequential Functional Chart)

Il linguaggio SFC

- introduzione -

Negli anni 70 del secolo scorso (cioè circa 10 anni dopo il lavoro di C.A. Petri) venne sviluppato un linguaggio grafico - inizialmente di specifica, poi anche di programmazione quando se ne crearono dei compilatori - per il controllo logico.

Tale linguaggio venne chiamato **Grafcet** (**G**raphe de **c**oordination **é**tapes-**t**ransitions).

Grafcet è profondamente legato alle reti di Petri, in quanto ne costituisce una semplificazione dotata di una specifica formale dell'evoluzione e del comportamento ingresso/uscita, in modo che da esso si possa ottenere senza ambiguità del codice eseguibile da una macchina sincrona interfacciata verso il mondo esterno (il PLC).

Grafcet è tuttora uno dei linguaggi standard per la modellizzazione di processi discreti ed è stato incluso in IEC 1131-3 con il nome di **S**equential **F**unctional **C**hart.

Le principali linee guida per la definizione di SFC sono le seguenti:

- rappresentare le funzioni di controllo logico con modelli formali e ben leggibili da chi conosce il sistema oggetto del controllo,
- superare i limiti degli automi a stati finiti (ad esempio permettendo il parallelismo),
- adattare le reti di Petri ad un'implementazione industriale.

Il linguaggio SFC

- elementi fondamentali -

Gli elementi fondamentali di SFC sono i seguenti:

- fasi o passi o tappe (étapes),
- transizioni,
- archi orientati che connettono fasi a transizioni o viceversa,
- regole di evoluzione, che definiscono senza ambiguità il comportamento dinamico del programma nella sua veste d'implementazione di un DES,
- azioni associate alle fasi,
- condizioni logiche associate alle transizioni.

Lo stato di attivazione delle fasi rappresenta lo stato del sistema. Esso viene modificato dall'occorrenza di eventi, che mediante le transizioni portano il sistema in una nuova condizione (con altre fasi attive).

Il linguaggio SFC

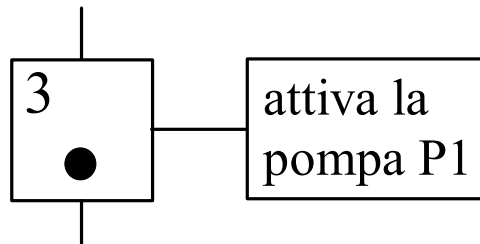
- elementi fondamentali -

Fase.

E' concettualmente correlata al posto nelle reti di Petri:

- è simboleggiata da un **quadrato** (con bordo doppio se **iniziale**, ovvero attiva all'inizio dell'esecuzione del programma),
- ha un'**etichetta**, in genere costituita da un numero,
- può essere **attiva** o **inattiva**; quando è attiva contiene **un marcatore** (gettone),
- può avere delle **azioni** associate.

Esempio:



In uno schema SFC si dicono **variabili di stato** delle variabili logiche associate allo stato di attivazione delle fasi, intendendo con ciò che l'esistenza della fase i implica quella di un simbolo X_i che vale 1 se la fase i è attivata e 0 altrimenti. Le variabili di stato hanno questo nome perché **l'insieme delle fasi attive rappresenta lo stato del (DES descritto dal)lo schema SFC** tranne per quel che attiene alle variabili, che però non hanno a che fare con l'evoluzione del DES in termini di eventi se non in virtù degli eventi stessi e delle azioni. Le variabili di stato possono essere usate nello schema: tipicamente compaiono nelle condizioni associate alle transizioni.

Il linguaggio SFC

- elementi fondamentali -

Transizione.

E' concettualmente correlata alla transizione nelle reti di Petri:

- è simboleggiata da una **barretta orizzontale**,
- è dotata di un **identificatore**,
- può avere una **condizione logica** associata,
- può essere **abilitata** oppure no, **superabile** oppure no,
- può **essere superata**.

Esempio:

 X1 AND X2

Arco orientato.

E' concettualmente correlato all'arco orientato nelle reti di Petri:

- è rappresentato da **segmenti** collegati con angoli retti,
- collega fasi a transizioni o viceversa (non fasi a fasi né transizioni a transizioni, come nelle reti di Petri),
- non esiste peso,
- si omette la **freccia** direzionale se è dall'alto verso il basso.

Il linguaggio SFC

- elementi fondamentali -

Regole di evoluzione.

Premesse.

Una transizione si dice **abilitata** se tutte le fasi a monte sono attive.

Una transizione si dice **superabile** se è abilitata e la condizione ad essa associata assume il valore "vero" (in tal caso si dice anche che la transizione **può scattare**).

Regola 1.

Quando una transizione è superabile, essa viene superata: tutte le fasi a monte vengono disattivate e tutte le fasi a valle vengono attivate.

Regola 2.

Se più transizioni sono superabili in un certo istante, esse vengono superate tutte contemporaneamente (quindi SFC dà del codice un modello **sincrono**).

Osservazione importante.

SFC è un linguaggio che descrive il ciclo operativo del sistema di controllo, nel senso che "in un programma SFC è scritto come cambia lo stato del sistema all'avvenire degli eventi". A differenza di LD, in SFC non si vede (a meno di un'eccezione che vedremo) il ciclo operativo del PLC.

Il linguaggio SFC

- elementi fondamentali -

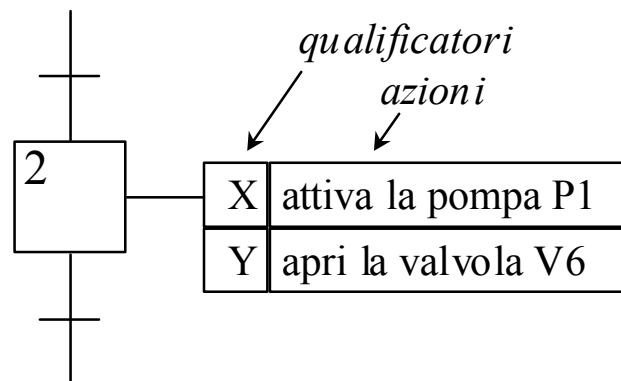
Azioni.

Le azioni sono associate ai passi e sono lo strumento con cui lo schema SFC appunto agisce, sulle variabili interne al PLC e/o interagendo con l'esterno.

Un passo può avere più azioni associate, e in tal caso esse sono eseguite in parallelo.

Le azioni sono dotate di attributi (qualificatori) per descrivere in modo non ambiguo l'andamento temporale dell'effetto da esse prodotto.

Esempio:



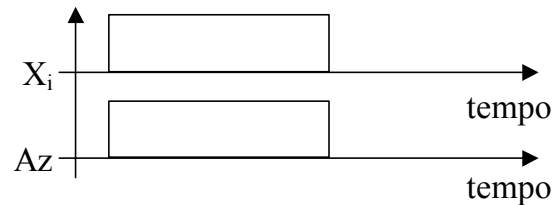
Il linguaggio SFC

- elementi fondamentali -

Qualificatori delle azioni.

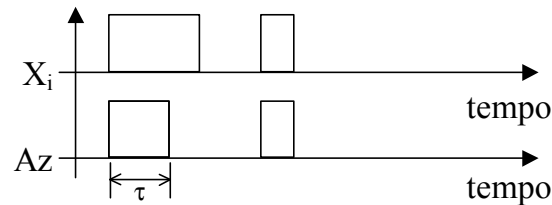
N (non-stored, azione continua).

L'azione (**Az**) inizia quando la fase cui è associata (**X_i**) si attiva e termina quando la fase medesima si disattiva.



L (time-limited, azione limitata nel tempo).

L'azione (**Az**) inizia quando la fase cui è associata (**X_i**) si attiva e termina quando la fase medesima si disattiva oppure è trascorso un assegnato intervallo di tempo t .



Il linguaggio SFC

- elementi fondamentali -

P (pulse, azione impulsiva).

L'azione comincia quando la fase si attiva (o si disattiva, secondo un'apposita opzione), ovvero sul fronte di salita (o di discesa) della variabile booleana associata allo stato della fase, e viene eseguita una volta sola, indipendentemente da quanto a lungo la fase resta attiva.

Un'azione siffatta si deve poter descrivere come l'esecuzione di un programma, dimodoché abbia senso "farla una volta sola". Le azioni **P** si usano quindi di solito per calcoli, avvio di temporizzatori, aggiornamento di contatori, e così via.

S (stored o set, azione memorizzata).

L'azione comincia quando la fase si attiva e dura, indipendentemente da quanto a lungo la fase resta attiva, finché non viene resettata con un'azione uguale ma con qualificatore **R**.

R (reset, azione cancellata).

Fa terminare un'azione memorizzata.

Il linguaggio SFC

- elementi fondamentali -

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

L'azione comincia dopo un intervallo di tempo assegnato dacché la fase è diventata attiva, anche se la fase nel frattempo si è disattivata. L'azione dura finché non viene resettata con un'azione uguale ma di tipo **R**.

DS (delayed and stored, azione ritardata e memorizzata)

L'azione comincia dopo un intervallo di tempo assegnato dacché la fase è diventata attiva, se la fase lo è ancora in quel momento. Se inizia, l'azione dura finché non viene resettata con un'azione uguale ma di tipo **R**.

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

L'azione comincia quando la fase diventa attiva e termina quando viene resettata con un'azione uguale ma di tipo **R** o quando è trascorso un assegnato intervallo di tempo.

Il linguaggio SFC

- elementi fondamentali -

Condizioni logiche associate alle transizioni.

Le condizioni associate alle transizioni sono espressioni logiche di tipo generale e possono coinvolgere variabili **d'ingresso**, **di stato**, **temporali** e **condivise**.

Le **variabili di ingresso** possono essere booleane, intere o reali. Le variabili booleane possono essere **semplici** (e in tal caso semplicemente riportano quanto arriva dal corrispondente sensore) oppure **a riconoscimento di fronte**. Questo tipo di variabile s'indica premettendo al nome il simbolo **↑** e vale 1 per la durata di un ciclo di scansione **del PLC** (ecco l'eccezione) quando il segnale del sensore passa da 0 a 1. Ovviamente col simbolo **↓** si ha lo stesso effetto, ma quando il segnale va da 1 a 0.

Delle **variabili di stato** si è già detto.

Le **variabili temporali** sono necessarie per l'implementazione dei sistemi logici. In SFC il tempo è reso presente tramite temporizzatori associati alle fasi, che contano il tempo trascorso dall'ultima attivazione della fase. Quando la fase si disattiva, ovviamente, il suo timer si azzerà. Convenzionalmente indicheremo il timer della fase **i** con **$X_i.t$** , col che l'espressione **$X5.t > 2min$** diviene vera (al primo ciclo del PLC che avviene più di) 2 minuti dopo l'ultima attivazione della fase 5, se questa nel frattempo non si è disattivata.

Le **variabili condivise** sono semplicemente variabili definite in una zona di memoria visibile da tutto lo schema SFC. E' possibile definire un'azione che agisce su una variabile condivisa, che a sua volta può essere usata nella condizione logica associata a qualunque transizione.

Il linguaggio SFC

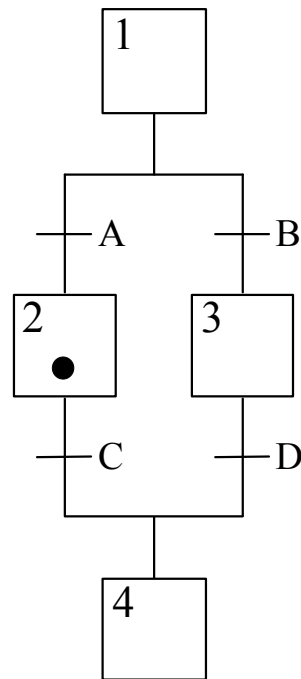
- strutture base -

Scelta e convergenza (if-then-else).

Si usano due **nodi diramatori**:

- un **nodo di scelta**, ovvero una fase seguita da più transizioni **con condizioni mutuamente esclusive**;
- un **nodo di convergenza**, ovvero più transizioni che portano nella (cioè scattando attivano la) stessa fase.

Esempio.



Osservazione.

Se nell'esempio le condizioni **A** e **B** non fossero mutuamente esclusive, la fase 4 potrebbe "venir attivata due volte", il che a rigore **non ha senso** (il marcatore di attività della fase è un ente booleano) e comunque è ambiguo.

Questa è una prima differenza tra SFC e reti di Petri (torneremo sull'argomento).

Il linguaggio SFC

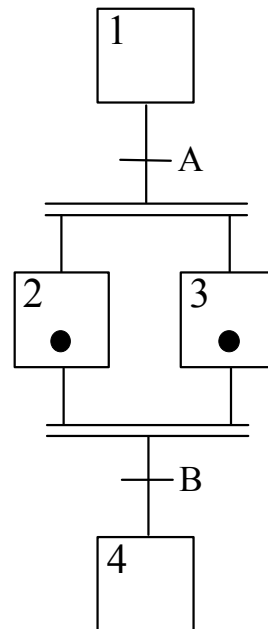
- strutture base -

Parallelo (attivazione e disattivazione simultanea di più fasi).

Si usano due **nodi diramatori**, indicati però in questo caso con **barre doppie**:

- un **nodo di parallelismo**, ovvero una transizione seguita da più fasi (che quindi essa, scattando, attiva tutte);
- un **nodo di concorrenza**, ovvero una transizione preceduta da più fasi (e quindi superabile solo se esse sono tutte attive).

Esempio.



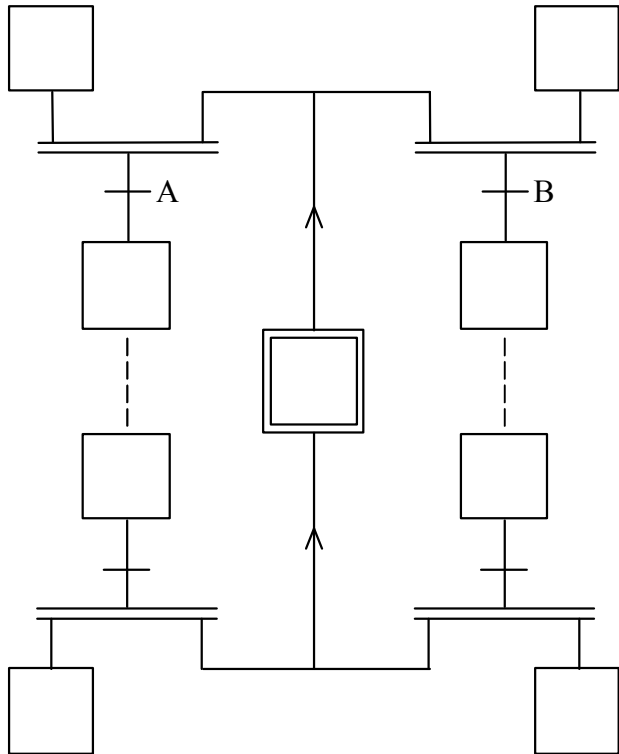
Il linguaggio SFC

- strutture base -

Semaforo di mutua esclusione.

E' una struttura simile al modello di una risorsa condivisa nelle reti di Petri.

Esempio.



Osservazioni.

Ovviamente le condizioni **A** e **B** devono essere mutuamente esclusive e la fase che costituisce il semaforo, ovvero quella che attivandosi rappresenta la disponibilità della risorsa, dev'essere una fase iniziale (nell'esempio è la fase al centro). Quindi, l'analogia con le reti di Petri c'è nel solo caso di una risorsa esistente in quantità unitaria e inizialmente disponibile.

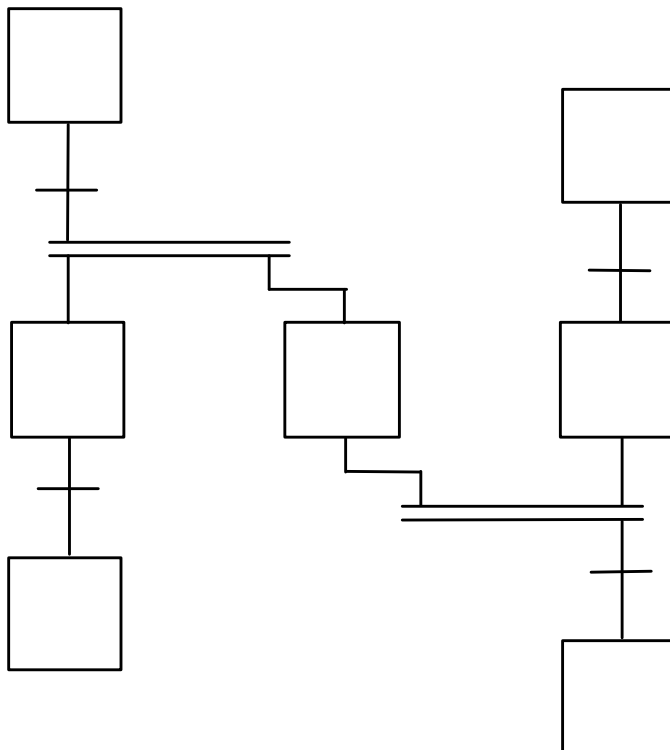
Il linguaggio SFC

- strutture base -

Sincronizzazione locale.

E' una struttura in cui due sequenze si devono aspettare ad una data fase.

Esempio.



La sequenza a destra deve attendere quella a sinistra.

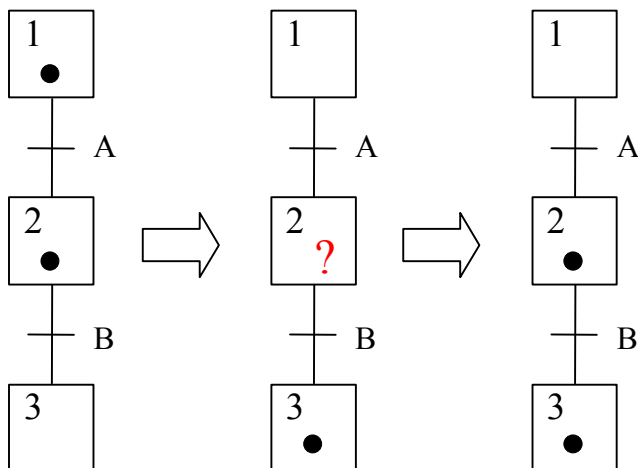
Il linguaggio SFC

- situazioni e costrutti da evitare (errori tipici) -

Disattivazione e attivazione contemporanea di una fase.

Può accadere che una fase debba essere contemporaneamente disattivata e attivata: convenzionalmente tale fase rimane attiva, ma è bene evitare questa situazione (e quella simmetrica nel caso di attivazione con contemporanea disattivazione).

Esempio.



Inizialmente sono attive le fasi 1 e 2.
Se le condizioni logiche **A** e **B** sono entrambe vere, entrambe le transizioni sono superabili.
Per superare la transizione in alto bisogna disattivare la fase 1 e attivare la 2.
Per superare la transizione in basso bisogna disattivare la fase 2 e attivare la 3.

C'è una potenziale confusione: la fase 2 è attiva o inattiva?

La si assume attiva, ma è bene evitare la situazione (cioè o non usare il costrutto, oppure se è necessario farlo, come ad esempio succede spesso nel descrivere i nastri trasportatori, garantire che **A** e **B** diventino vere nell'ordine giusto) perché ci possono essere altre confusioni.

Ad esempio, se nella situazione iniziale **A** è vera ma **B** è falsa, si tenta di attivare due volte la fase 2...

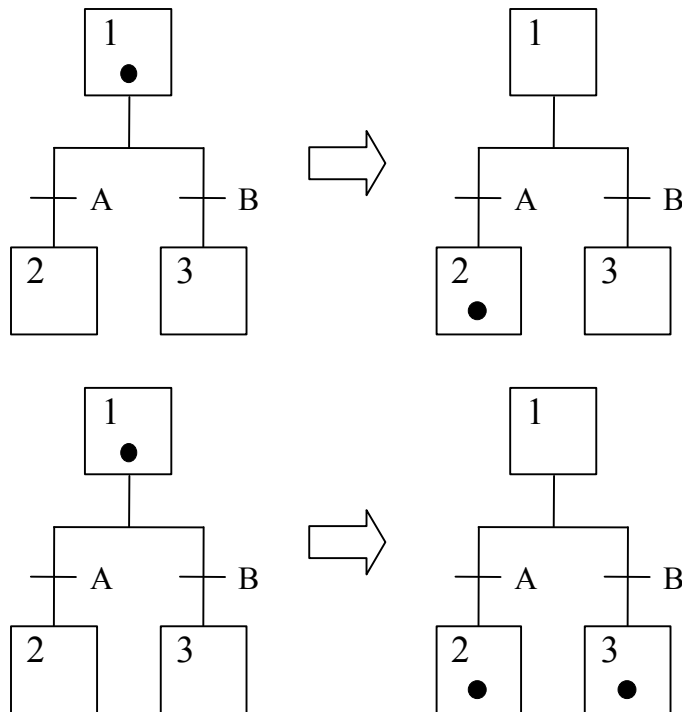
Il linguaggio SFC

- situazioni e costrutti da evitare (errori tipici) -

Nodo divergenza con condizioni non mutuamente esclusive.

Se non si presta attenzione, possono essere attivati più di uno dei rami che seguono il nodo, con le potenziali inconsistenze di cui ai punti successivi.

Esempio.



Supponiamo che **A** e **B** non siano mutuamente esclusive.

Allora, se **A** è vera e **B** falsa (o viceversa), si ha un'evoluzione corretta ("di tipo OR")

mentre, se **A** e **B** sono ambedue vere, si ha un'evoluzione scorretta ("di tipo AND").

Il linguaggio SFC

- situazioni e costrutti da evitare (errori tipici) -

Per evitare il problema occorre rendere mutuamente esclusive le condizioni, ma così s'introduce un'altra differenza con le reti di Petri.

Infatti, la cosa si può fare in diversi modi.

- sostituendo **B** con **B and not A**: così però si dà priorità alla transizione di sinistra, perché se **A** e **B** sono entrambe vere essa sola risulta superabile.
- sostituendo **A** con **A and not B**: così però si dà priorità alla transizione di destra, per lo stesso motivo.
- sostituendo **B** con **B and not A** e **A** con **A and not B**: così però, se **A** e **B** sono entrambe vere, nessuna delle due transizioni è superabile (discuteremo tra poco se fare così ha senso).
- sostituendo **B** con **B and (not A or ScegliB)** e **A** con **A and (not B or ScegliA)**, dove **ScegliA** e **ScegliB** sono due variabili che il programma può assegnare in qualsiasi modo, pur di non porle entrambe a 1. Così si sceglie "dall'esterno" se far superare la transizione di sinistra o quella di destra.

Il linguaggio SFC

- situazioni e costrutti da evitare (errori tipici) -

Osservazione importante.

L'arbitrarietà di cui sopra equivale a scegliere come risolvere un conflitto tra le due transizioni (pensate ora come parte di una rete di Petri).

Infatti, riconsiderando nell'ordine le quattro scelte proposte, si possono dar loro i seguenti significati.

- Se c'è conflitto effettivo vince la transizione di sinistra.
- Se c'è conflitto effettivo vince la transizione di destra.
- Se c'è conflitto effettivo si sta fermi finché una delle due non viene disabilitata, e a quel punto scatta l'altra. Discutiamo ora se ciò ha senso.
Perché abbia senso bisogna che qualcuno possa disabilitare (o, più esattamente, rendere non superabile) almeno una delle due, se no si ha il blocco.
Allora, posto che **A** e **B** possano essere ambedue vere (se no tutto questo discorso perde significato),
 - o le condizioni **A** e **B** sono "sbagliate", nel senso che devono contenere anche altre quantità logiche in modo che la disabilitazione possa avvenire (e in tal caso la rete di Petri "origine" dello schema SFC non è a scelta libera perché in essa almeno una delle due transizioni ha in preset altri posti che non sono stati "tradotti" nelle condizioni logiche medesime),
 - oppure **A** e **B** sono corrette, e allora è giusto che se nessuno risolve il conflitto (a questo punto però in modo arbitrario, corrispondentemente al fatto che in questo caso la rete di Petri "origine" è a scelta libera) tutto stia fermo.

Il linguaggio SFC

- situazioni e costrutti da evitare (errori tipici) -

- Se c'è conflitto, e assumendo che **A** e **B** siano corrette, si decide (in modo arbitrario, poiché assumere la correttezza di **A** e **B** vuol dire assumere che la rete di Petri "origine" sia a scelta libera) chi scatta, assegnando le variabili **ScegliA** e **ScegliB**. Tipicamente, questo compito spetta ad un supervisore.

Tocchiamo quindi con mano il fatto che, nello scrivere il programma di controllo, i conflitti vanno (e possono essere) risolti.

Apprezziamo anche l'opportunità del concetto di rete a scelta libera, perché è a partire da esso che abbiamo concepito un metodo per risolvere i conflitti in modo sistematico e, soprattutto, senza ambiguità nel passaggio dal modello al codice.

Infine, abbiamo così individuato e trattato un'altra importante differenza tra reti di Petri e SFC.

Il linguaggio SFC

- situazioni e costrutti da evitare (errori tipici) -

Altri errori tipici.

- Introdurre una scelta tra sequenze e, a valle, una sincronizzazione tra le stesse sequenze. In tal caso, poiché solo una sequenza sarà attiva, la transizione a valle del nodo di sincronizzazione non sarà mai superabile.
- Introdurre un parallelismo tra sequenze e, a valle, una convergenza delle stesse sequenze. In tal caso la fase a valle del nodo di convergenza può essere attivata più di una volta, il che non ha senso (o quantomeno è ambiguo).

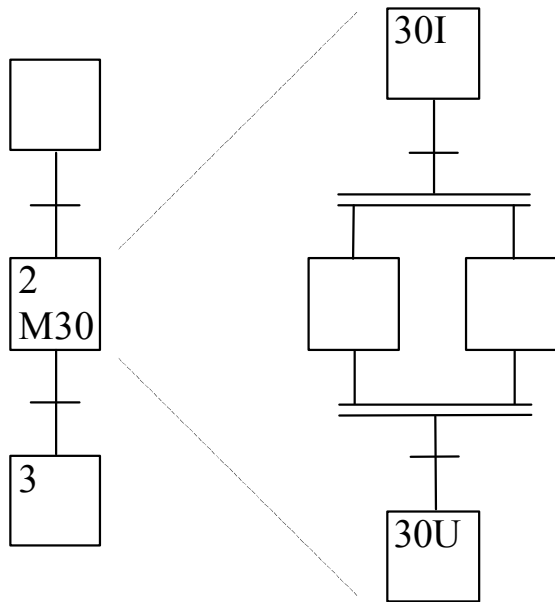
Il linguaggio SFC

- strutture complesse: gerarchia e modularità -

Macrofasi.

Le macrofasi funzionano come i sottoprogrammi: quando si attiva una fase associata ad una macrofase, quest'ultima comincia ad evolvere e termina quando la prima si disattiva.

Esempio.



Osservazioni.

Sulle macrofasi, la IEC1131-3 è ambigua. Infatti, da quanto detto non si capisce

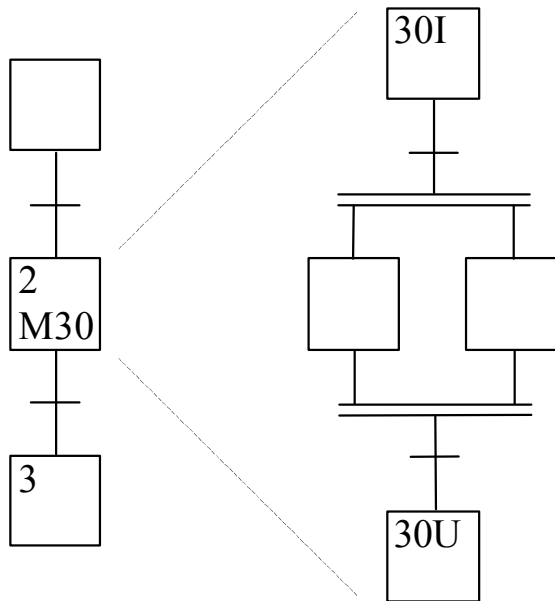
- cosa succede alle azioni presenti nella macrofase associata alla fase 2 quando quest'ultima si disattiva,
- cosa succede a quelle azioni se la transizione a valle della fase 2 è superabile quando tale fase si attiva,
- come sono gestite le variabili usate nella macrofase,
- come sono gestite le azioni eventualmente settate nella macrofase con i qualificatori S, SD, DS, SL.

Il linguaggio SFC

- strutture complesse: gerarchia e modularità -

Questi “dettagli” non sono standardizzati e vanno chiariti verificando le regole di evoluzione SFC dello specifico ambiente di sviluppo usato.

Occorre stare molto attenti, dato che questo ha grande influenza sulla portabilità del codice.



La regola di evoluzione più usata, comunque, è la seguente.

Con riferimento all’esempio, se deve essere attivata la fase 2, associata alla macrofase M30, si attiva in realtà la fase 30I della macrofase; l’evoluzione continuerà sullo schema SFC della macrofase fino ad arrivare alla fase 30U, dopodiché riprenderà a seguire lo schema SFC originario dalla fase 3 (quando la condizione associata alla transizione a monte di tale fase sarà verificata).

Il linguaggio SFC

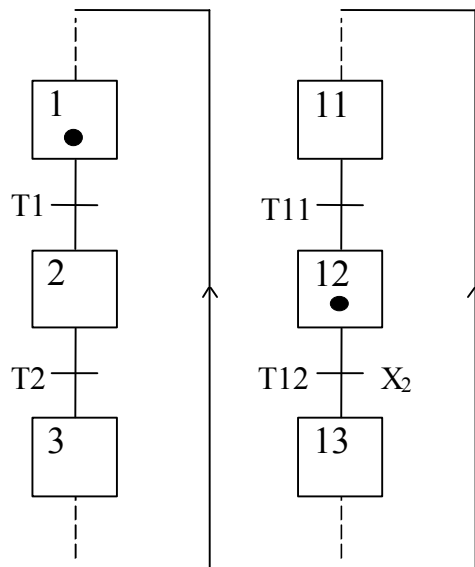
- strutture complesse: gerarchia e modularità -

Sincronizzazione di più schemi SFC eseguiti in parallelo.

Molti PLC permettono di eseguire in parallelo più programmi, anche con tempi di ciclo diversi, che comunicano tramite una memoria condivisa (ovvero con variabili globali).

Un altro metodo per implementare delle gerarchie tra schemi SFC consiste quindi nell'usare variabili (tipicamente di stato e condivise) per influenzare l'evoluzione di uno schema in base allo stato corrente di un altro schema. Occorre però prestare attenzione all'uso di riferimenti incrociati, che non devono essere posizionati in modo tale da determinare deadlock. Naturalmente, se l'insieme dei programmi viene da un modello in termini di rete di Petri, questo si può vedere facendone l'analisi.

Esempio.



Il superamento della transizione T12 dipende dallo stato di attivazione della fase 2.

L'evoluzione dello schema di destra, quindi, si blocca in attesa di quella dello schema di sinistra.

Questa è di fatto una sincronizzazione, che a volte si dice **implicita** per distinguerla da quella locale (allo schema) vista in precedenza.

Il linguaggio SFC

- strutture complesse: gerarchia e modularità -

Forzatura, sospensione e bloccaggio (cenni).

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

Per semplificare la notazione grafica, questi costrutti sono realizzati per mezzo di **macroazioni**. La macroazione è dunque, per definizione, un'azione operata da uno schema SFC che ha effetti sulla condizione di un altro schema SFC.

Trattare a fondo le macroazioni esula dal corso, salvo dire che (come tutte le sincronizzazioni implicite) possono ridurre la leggibilità e persino introdurre blocchi. La seconda cosa, al solito, si evita facendo provenire il codice da un modello su cui si sono fatte analisi (comunque complesse). La prima si evita usando le macroazioni soltanto per compiti di livello molto alto, tipo "attiva una ricetta" o simili.

Detto ciò, vediamo *per exemplum* come funzionano le tre macroazioni SFC.

Macroazione (presente in uno schema SFC)

FORZARE Pippo:{2,5}

FORZARE Pippo:{}

FORZARE Pippo:{*}

Effetto su un altro schema SFC, ovvero quello (qui di nome Pippo) oggetto della macroazione

Si attivano le fasi 2 e 5 e si disattivano tutte le altre, col che si **forza** lo stato di Pippo.

Si disattivano tutte le fasi, col che si **sospende** l'evoluzione di Pippo.

L'insieme delle fasi attive non può cambiare, col che si **blocca** lo stato di Pippo.

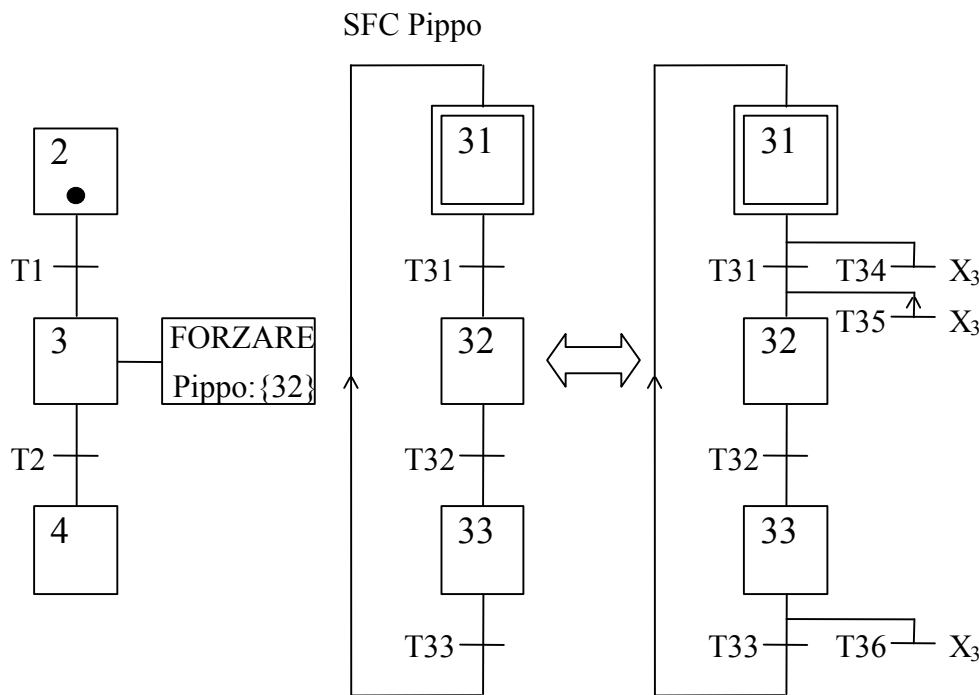
NOTA: la dicitura "FORZARE" è, ovviamente, convenzionale.

Il linguaggio SFC

- strutture complesse: gerarchia e modularità -

Esempio d'interpretazione di una macroazione.

A puro titolo di esempio, per renderci conto di come una macroazione complica il modello, proviamo semplicemente a trascrivere l'effetto di una forzatura in termini di soli elementi SFC base (fasi, transizioni e variabili).



La macroazione del primo schema forza lo stato del secondo schema (Pippo) ad avere attiva la sola fase 32.

Ciò è equivalente ad utilizzare la variabile **X₃** come nel terzo schema.

Si noti la complessità, anche in schemi semplici come questi.

Davvero, è bene non usare le macroazioni se non a livello molto alto, ad esempio per attivare le ricette di produzione.

Il linguaggio SFC

- conclusioni -

- Il linguaggio SFC è molto legato alle reti di Petri. Le differenze fondamentali e critiche sono state viste; ve ne sono alcune di minore importanza, che vedremo al momento di mettere insieme SFC, LD e reti di Petri.
- SFC è un linguaggio procedurale il cui modello del codice ricalca in modo stretto il ciclo operativo dell'impianto.
Quindi è innaturale esprimere in SFC dei vincoli ma è invece naturalissimo esprimere sequenze.
- Come abbiamo preannunciato dopo la trattazione di LD, quindi, useremo SFC essenzialmente per il controllo.
- Il linguaggio SFC non è presente in tutti i PLC ma, stante i suoi legami naturali col paradigma modellistico delle reti di Petri, è comunque utile di minimo come linguaggio di specifica.
- Esiste un metodo per la traduzione da SFC a LD, il che rende di fatto gli schemi SFC implementabili ovunque senza ambiguità. Su questo metodo (che peraltro è semplicissimo da usare) daremo alcuni cenni e vedremo qualche esempio.