CAP. 3

3. TECNICHE DESCRITTIVE, MODELLI E FORMALISMI

Il problema centrale di questo capitolo è la **descrizione** di cosa si vuole, cosa è disponibile, cosa si sta facendo, cosa si è fatto; a livello HW e SW in piccolo ed in grande.

Nella prima parte del capitolo si evidenzia la necessità di descrizioni nelle varie fasi dello sviluppo di un progetto.
Vengono analizzati in questa parte:

motivazioni e livelli delle descrizioni,
entità e aspetti da descrivere,
tecniche di descrizione

Nella seconda parte vengono brevemente richiamati alcuni tra i formalismi più adatti per descrivere specifiche o progetti di sistemi per cui l'evoluzione nel tempo costituisce un aspetto fondamentale della correttezza.
È interessante notare che questi strumenti sono utilizzabili per descrivere

- comportamenti voluti per un sistema
- comportamenti ottenuti mediante esecuzione di programmi (SW)
- comportamenri ottenuti dalla funzionalità intrinseca di macchine e dispositivi.

Gli strumenti di modellizzazione o formalismi di descrizione a cui si fa cenno sono:

Diagrammi di Flusso di Dati - DFD
Automi a Stati
Statecharts
Reti di Petri
Schemi GRAFCET
Linguaggio logico con metrica temporale TRIO

3.1 PROBLEMATICHE E TECNICHE DESCRITTIVE DI SPECIFICHE E PROGETTI

In tutte le discipline è essenziale disporre di strumenti atti a descrivere le entità di cui si vuole trattare. In particolare le discipline ingegneristiche necessitano di strumenti per descrivere **oggetti** e **comportamenti** nelle seguenti situazioni.

- SPECIFICA - Si descrivono le caratteristiche di ciò che si vuole ottenere.
- PROGETTO - Si descrivono entità astratte che si stanno inventando, pensate per poter essere realizzate in modo da soddisfare le esigenze espresse dalle specifiche.
- IMPLEMENTAZIONE - Si descrivono entità concrete e realizzate, o pensate come tali.

Nell'ambito delle applicazioni di calcolatori per automazione industriale le descrizioni riguarderanno:
- dispositivi, macchine e fenomeni del cosiddetto "mondo esterno (al calcolatore)"
- aspetti architetturali e circuitali (hardware) e programmativi (software) dei calcolatori.

È superfluo sottolineare l'importanza degli strumenti di descrizione (detti in generale **linguaggi**) che costituiscono il veicolo di comunicazione tra persone e tra uomo e macchina, ma che sono anche un importante supporto all'attività intellettuale e creativa di studiosi e tecnici.
Si ritiene quindi utile una, sia pur succinta, analisi di problemi e tecniche connessi con le descrizioni.

Si noti che i sistemi informatici corredati di opportuni strumenti CAD (Computer Aided Design) costituiscono il miglior supporto per la gestione delle descrizioni di cui stiamo parlando. In particolare prendono il nome di CASE (Computer Aided Software Engineering) gli strumenti specificatamente orientati alla gestione degli aspetti SW di un progetto. È anche interessante notare che alcune tecniche di descrizione, astratte rispetto agli aspetti implementativi, sono atte a rappresentare funzionalità che potranno essere realizzate con opportune ripartizioni tra HW e SW (HW/SW *codesign*).

3.1.1 MOTIVAZIONI

I moventi per fare una descrizione possono essere svariati:

- Per **comunicare** - È il motivo principale, a cui anche i successivi possono essere ricondotti.

- Per **supportare il pensiero** - È una specie di comunicazione che si fa a se stessi. Molto spesso una descrizione pregnante costituisce un valido aiuto alla mente che sta cercando di districarsi in problemi complessi ed affollati da numerose entità e relazioni tra esse.

- Per **evidenziare e chiarire** - È una variante del punto precedente, che può essere rivolta anche ad altri soggetti diversi da chi esegue la descrizione.

- Per facilitare la **verifica di proprietà** - Alcune tecniche formali di descrizione rendono più rigoroso e talora anche automatizzabile, il procedimento di verifica che il sistema descritto gode di proprietà volute o desiderabili.

- Per **realizzare** - Una descrizione di ciò che deve essere realizzato è una fase indispensabile se il progettista ed il costruttore sono persone diverse, ed è comunque un valido strumento per creare un supporto di comunicazione tra gli stati mentali, molto diversi tra loro, dell'inventiva e della realizzazione.

- Per **documentare e ricordare** - Anche quando qualcosa è stato realizzato, le sue descrizioni di progetto e realizzative, mantengono un'importanza fondamentale per consentire più facilmente modifiche, manutenzione e, non ultimo, riutilizzo in altre applicazioni degli sforzi di sviluppo.

- Per **insegnare** - Anche l'insegnamento, e non solo quello scolastico, può costituire un importante motivo per effettuare descrizioni di problemi, soluzioni, realizzazioni, ecc., ovviamente con gli opportuni criteri "didattici", cioè con un percorso di argomenti più orientato al meccanismo di apprendimento che alla struttura del problema. Si pensi ad esempio al *training* di tecnici e operatori.

3.1.2 LIVELLI DI DESCRIZIONE

Le descrizioni di sistemi complessi sono necessariamente anch'esse complesse. Per maneggiare in modo efficace la complessità si ricorre solitamente ad una scomposizione della descrizione in più livelli (o piani).

3.1.2.1 LIVELLI DI ASTRAZIONE - dall'aggregato al dettaglio.

Costituisce probabilmente la più importante scomposizione, basata sul classico concetto di "black-box (scatola nera)" che alleggerisce la descrizione limitandosi agli aspetti esterni degli elementi considerati e nascondendo i dettagli interni che sono considerati irrilevanti al livello corrente e vengono invece presentati, "aprendo le scatole", al livello successivo. Corrisponde al tipico processo "top-down (discendente)" spesso suggerito per le fasi di progetto, ma molto apprezzabile anche per le fasi di studio.
Ad es. un impianto sarà descritto ad alto livello come un insieme di blocchi funzionali (macchine) e flussi (di materiali, energia e informazioni) caratterizzati a livello macroscopico. A livelli inferiori le singole macchine saranno “esplose” evidenziando le componenti e le funzionalità interne.

3.1.2.2 LIVELLI DI COMPLETEZZA - da più importante a tutto

Quando ciò che si deve descrivere è composto da numerosi elementi può essere comodo presentare anche delle viste "spopolate" in cui compaiono solo gli elementi più importanti rispetto alla funzionalità complessiva.
Ad es. di un pannello operatore si può fare una descrizione semplificata che riporti solo i comandi di start (marcia) e di stop (arresto) e le visualizzazioni di marcia e allarme, rimandando al piano della descrizione completa per ritrovare tutti gli ulteriori comandi ed indicatori.

3.1.2.3 LIVELLI DI RIGOROSITÀ - da generico a precisato

È frequente il caso che alcuni aspetti debbano, prima o poi, essere descritti in modo rigoroso ma che tale rigore possa essere in qualche modo "sottinteso" in alcuni piani di presentazione (ai livelli superiori) e completamente specificato in altri (ai livelli inferiori).
Corrisponde al classico approccio per "raffinamenti successivi".
Ad es. in una descrizione si potrà parlare di "acquisizione periodica" di una grandezza fisica, mentre in un livello successivo si preciserà che "il periodo **P** di acquisizione dovrà soddisfare la relazione: 10 ms < **P** < 20 ms".

3.1.2.4 LIVELLI DI RISOLUZIONE - da grossolano a preciso

Talvolta può essere comodo e significativo riportare valori numerici approssimativi nei livelli più generali e lasciare ai livelli di dettaglio la loro precisazione.
Ad es. si potrà citare una velocità di trasmissione di "alcune migliaia di caratteri al secondo" che, ad un livello di maggior dettaglio potrà essere precisata in "375 Kbit/s".

3.1.3 ENTITÀ DA DESCRIVERE

3.1.3.1 FENOMENI

Cioè il comportamento (behavior) o evoluzione temporale di stati, grandezze, eventi e correlazioni temporali e funzionali tra questi.
Con le equazioni integro-differenziali rispetto al tempo, si esprimono contemporaneamente relazioni di valori e temporali, mentre le funzioni logiche ed aritmetiche richiedono in genere una descrizione degli aspetti temporali separata ed aggiuntiva rispetto a quella dei valori.

3.1.3.2 OGGETTI (DISPOSITIVI)

Per oggetti o dispositivi si intendono gli elementi considerati "atomici", di cui cioè si ritiene sufficiente, al livello di descrizione considerato, la caratterizzazione fornita dal comportamento, opportunamente modellizzato, "ai morsetti".
Alcuni oggetti sono sede di fenomeni locali (come le macchine, i circuiti, ecc.), altri presentano delle proprietà statiche (come gli elementi strutturali, architettonici, ecc.).

3.1.3.3 INTERAZIONI (COLLEGAMENTI)

I collegamenti (in senso lato) tra oggetti descrivono le interazioni tra i fenomeni che in tali oggetti hanno sede.
Talvolta le interazioni presentano complessità che vanno oltre il semplice "collegamento" e si basano su trasformazioni e funzioni particolari, assumendo così il ruolo di "interfacce" con i relativi “protocolli”.

3.1.3.4 SOTTOSISTEMI

Sono gruppi di oggetti tra loro collegati che presentano nel loro insieme un grado di autonomia sufficiente per rendere significativa l'aggregazione, anche se presentano interazioni con altri oggetti (o sottosistemi) considerati "esterni".

I sottosistemi costituiscono spesso dei **moduli** per i quali è molto importante la descrizione dei collegamenti di interfaccia con altri sottosistemi.

3.1.3.5 SISTEMI

Sono gruppi di oggetti (o più spesso di sottosistemi) che non presentano significative (cioè rilevanti agli effetti della descrizione) interazioni con l'esterno.

3.1.4 CONTESTI DI DESCRIZIONE

Come si è accennato, si ha la necessità di descrizioni in diverse situazioni che ora analizziamo un po' più in dettaglio.

3.1.4.1 REQUISITI - SPECIFICHE

Requisiti globali Processo+controllore
Requisiti (dedotti) sul controllore

La descrizione delle **specifiche** di un sistema (o sottosistema) astratto costituisce un **criterio di accettabilità** di un (sotto)sistema concreto rispetto a determinati obbiettivi.
Spesso tale descrizione è la base di un contratto tra chi commissiona, ed è disponibile a pagare per ottenere, tale (sotto)sistema e chi è chiamato a renderlo disponibile.
Perchè possano svolgere appieno il loro ruolo le **specifiche** devono essere descritte in modo **chiaro**, **completo**, **rigoroso** e **consistente**, oltre ad essere **soddisfacibili**, cioè portare ad una realizzabilità fisica.

È interessante notare che in molti casi, tra cui l'automazione di processi industriali, si possono individuare due livelli di specifica:

- Specifica **globale** (a livello di sistema) che precisa quali sono i comportamenti voluti per un sistema automatizzato.

- Specifica del sottosistema **controllore**, che precisa la funzionalità che il sottosistema (spesso in buona parte informatico) di automazione deve presentare perchè la sua interazione col sottosistema controllato renda soddisfatta la specifica globale.

La scomposizione in due sottosistemi (impianto e controllore) appena delineata è spesso **particolarmente significativa**, in quanto corrisponde a due diverse forniture, basate su competenze anche molto diverse, di cui quella relativa all'impianto è in molti casi vincolante (per motivi tecnologici , economici, ecc.) o già effettuata.

Si pone quindi spesso il problema di **dedurre** le specifiche di un sottosistema di automazione (calcolatore e relativi accessori) **dalle specifiche globali astratte** (ciò che si vuole ottenere) e **dalle specifiche concrete del sottosistema impianto** già realizzato, o pensato come tale (ciò che si ha già fissato), e tale problema non è sempre semplice e non è detto che abbia soluzione.

Le specifiche dovrebbero essere stese, in linea di principio, dal committente, ma è importante notare che spesso un buon livello di chiarezza, completezza e rigorosità

delle specifiche stesse si ottiene solo con l'apporto cooperativo di chi è chiamato a progettare e realizzare l'oggetto, di cui è esperto.

Da quanto detto risulta evidente che i linguaggi utili per la descrizione di specifiche devono essere particolarmente **orientati al problema**, così da appartenere ai domini linguistici propri sia dei committenti che dei progettisti di impianti o macchine.

3.1.4.2 SOLUZIONI - PROGETTO, IMPLEMENTAZIONE

Necessità di linguaggio orientato al problema ma con piccolo *gap* rispetto al mappaggio sulle risorse per consentire visibilità ed efficienza.
Documentazione

Sono necessarie **descrizioni delle soluzioni** al problema che è stato posto dalle specifiche. Tali soluzioni sono:

- prospettate nella fase di progetto,
- *in fieri* nella fase di implementazione
- realizzate al completamento del lavoro.

Queste descrizioni hanno diversi scopi, come verrà analizzato nel punto seguente, ma presentano in comune la necessità di descrivere con quali mezzi (cioè **come**) si ottengono i requisiti delle specifiche (che dichiarano **cosa** si vuole). In particolare i linguaggi utili per queste descrizioni dovranno trattare delle **risorse** su cui si basano le soluzioni e del **mappaggio delle funzionalità** su tali risorse. Dovranno cioè descrivere i vari dispositivi in gioco e le loro connessioni, e precisare come le loro funzionalità contribuiscano a quella globale.

È quindi desiderabile che tali linguaggi costituiscano il miglior compromesso tra "orientamento al problema" e "orientamento alle risorse".

Si noti che le risorse che qui ci interessano maggiormente sono sia di tipo HW, cioè processori, memorie, collegamenti, sensori, attuatori, ecc., sia di tipo SW, cioè strutture dati, costrutti di programmazione, istruzioni, servizi di sistemi operativi, ecc.
Il caso ideale corrisponde a linguaggi di descrizione **orientati al problema** e che siano anche "**eseguibili**", cioè comprensibili al sistema esecutore.

3.1.5 ASPETTI DA DESCRIVERE

Sia in sede di specifica che di progetto e realizzazione, vi sono diversi aspetti che può essere necessario descrivere e che sono tra loro complementari.

3.1.5.1 STRUTTURE

descrizioni dichiarative
relazioni
connettività
topologia
materiali

Le strutture sono presentazioni statiche che evidenziano la topologia, o comunque l'adiacenza, tra le varie parti che costituiscono l'oggetto della descrizione. Possono essere descritte anche le relazioni, le forme ed i materiali di queste parti.

In alcuni settori (ad es. edilizia, design, ecc.) le strutture costituiscono l'aspetto principale, a cui fanno riferimento anche le descrizioni degli altri aspetti.

Anche nell'ambito dell'informatica le strutture assumono un'importanza sostanziale: strutture dei **dati**, strutture dei **programmi** e loro organizzazione modulare e strutture **circuitali** ed architetturali sono aspetti di evidente importanza in ogni progetto informatico.

3.1.5.2 PROPRIETÀ

descrizioni dichiarative - formali

Le proprietà sono dichiarazioni assunte come vere o da dimostrarsi vere, relative a caratteristiche generali come funzionalità, temporizzazioni, tolleranze, usabilità, robustezza, disponibilità, sicurezza, costo, ecc.

La descrizione delle proprietà è particolarmente importante in sede di specifica, e spesso si presta all'uso di formalismi che possono costituire un valido strumento per dimostrarne la consistenza ed inferirne la validità.

3.1.5.3 COMPORTAMENTO

descrizioni algoritmiche - successioni di operazioni
descrizioni funzionali - espressioni logiche e matematiche
dominio del tempo
dominio della frequenza

Per i sistemi dinamici la descrizione del comportamento è una componente essenziale. In alcuni casi può essere utile evidenziare diversi

- **modi** (ad es. prova, manuale, automatico, ecc.),
- **fasi** (ad es. avviamento, taratura, regime, arresto, ecc.) e
- **passi** (sequenze di operazioni considerate atomiche).

Nel campo informatico gli algoritmi sono tipiche descrizioni di comportamenti.

Per i sistemi continui si possono avere descrizioni di comportamento mediante funzioni (nel dominio del tempo o della frequenza), che però possono essere pensate anche come *proprietà* del tipo visto al punto precedente.

3.1.6 PARADIGMI

I paradigmi di descrizione si riferiscono alla scelta di quali entità semantiche **evidenziare** in modo esplicito e quali lasciare implicite. I diversi paradigmi quindi spostano l'accento su diversi aspetti, privilegiando in alcuni casi la descrizione di **cosa è** l'oggetto in esame, in altri la descrizione di **come funziona**. È comunemente accettato che in diverse situazioni, anche in diverse parti dello stesso progetto, sia significativo usare di volta in volta i paradigmi più adatti.

3.1.6.1 IMPERATIVO

La maggior parte dei linguaggi di programmazione “classici” seguono il paradigma imperativo in cui si descrive come ottenere il comportamento voluto, mediante una successione di comandi (istruzioni) pensati come elementari, a cui si suppone che una macchina opportuna sia in grado di reagire effettuando le azioni richieste (esecuzione dei comandi).
Non si descrive cioè come è fatta la macchina, ma come deve operare per ottenere la funzionalità voluta.
Il paradigma imperativo presenta una stretta relazione con la funzionalità di base dei tipici calcolatori digitali, e consente quindi semplici ed efficienti traduzioni dei linguaggi di alto livello in istruzioni macchina. In molte situazioni, e tipicamente per i processi discreti, il paradigma imperativo è anche quello più spontaneo per la mente umana, dato che presenta il vantaggio di **scomporre la complessità temporale**, consentendo di focalizzare l’attenzione di volta in volta sul prossimo passo da eseguire.
È però abbastanza intuitivo che la citata semplificazione temporale cade quando non si debba descrivere un comportamento strettamente sequenziale, ma sia necessario, o anche solo conveniente, descrivere evoluzioni concorrenti o parallele nel tempo.

3.1.6.2 PROCESSI CONCORRENTI

Questo paradigma può essere considerato un’estensione del precedente, con aggiunte destinate a rappresentare la competizione e la comunicazione tra attività concorrenti, e che possono essere attribuite talvolta ai linguaggi e talaltra ai sistemi operativi.
Alcuni linguaggi sono stati dotati di costrutti che descrivono esplicitamente l’esecuzione concorrente di diversi processi (più specificamente detti *thread*), come i `cobegin` e `coend`, o come par in Occam.
Maggiore diffusione hanno avuto le soluzioni descrittive che basano le interazioni tra processi concorrenti su primitive fornite da appositi sistemi operativi, che verranno discusse più a fondo nel capitolo 10.

3.1.6.3 FUNZIONALE

Il paradigma funzionale è tipicamente rappresentato dalle formule matematiche che descrivono le operazioni per ottenere un certo risultato, ma lasciano largamente non specificata la sequenza temporale (quando il risultato è invariante rispetto a tale sequenza) delle operazioni elementari, e spesso lasciano implicitte le operazioni stesse.
Ad esempio *sin(x)/(a*b)* rappresenta un valore, senza precisare se prima calcolare *sin(x),* poi il prodotto *a*b* e infine la divisione, o prima calcolare il prodotto, ecc. Inoltre non dice nulla sul metodo per calcolare sin(x), se con sviluppo in serie, con ricerche su tabelle, o con altre tecniche.
Sono assenti i concetti di variabile e di comando e quindi anche di effetto collaterale (*side effect*) caratteristici dei paradigmi imperativi.

3.1.6.4 DICHIARATIVO

Le descrizioni che seguono il paradigma dichiarativo presentano i sistemi come insiemi di elementi, relazioni tra elementi e loro proprietà, lasciando completamente implicito il comportamento dei sistemi stessi.
Queste descrizioni rappresentano quindi la struttura del sistema, che è generalmente invariante rispetto alla sequenza o all’ordinamento spaziale delle varie componenti descrittive.

Dato un insieme di condizioni iniziali, il comportamento del sistema deriva "intrinsecamente" dalla struttura descritta.
Un esempio ben noto di descrizioni di questo tipo è costituito dagli schemi dei circuiti elettronici i cui simboli sono i **componenti**, con annesse anche se sottintese proprietà, e le **connessioni** tra i loro morsetti.
Un esempio di tipo software è costituito dai linguaggi logici, come il Prolog. In questo caso è compito del motore inferenziale inglobato nel sistema di esecuzione, realizzare su una macchina fisica sequenziale, come il calcolatore, una macchina virtuale che presenti il comportamento "spontaneo", trovando automaticamente la sequenza di azioni elementari, non esplicitamente descritte, che risolve il problema.
È abbastanza evidente come il parallelismo, anche a grana fine, trovi una naturale espressione nei paradigmi dichiarativi, nei quali semmai possono essere carenti gli aspetti di univocità e determinismo.
L'analogia di questi paradigmi con descrizioni costruttive di macchine, ne fa uno strumento familiare e ben accetto a molte categorie di tecnici non informatici. Infatti uno dei vantaggi di questi paradigmi è la loro attitudine a **scomporre la complessità strutturale**.
Sono interessanti le potenzialità offerte da questo paradigma, ma occorre notare anche che si ottiene poca efficienza se si virtualizza un sistema a funzionalità intrinseca su macchine sequenziali e non associative, e questo è uno dei motivi dell'impiego limitato a particolari problemi, delle applicazioni del paradigma dichiarativo a livello software.

3.1.6.5 ORIENTATO AGLI OGGETTI

L'approccio *object oriented*, applicato sia alle fasi di analisi che a quelle del progetto e dell'implementazione ha suscitato grande interesse negli ultimi anni, anche se alcuni concetti erano già noti e praticati.
Molti aspetti di questo paradigma presentano interessanti caratteristiche concettuali e anche implementative. È molto interessante l'idea di incapsulare negli oggetti sia la particolare rappresentazione delle informazioni ad essi associate (strutture dei dati) che le funzionalità (dette in gergo "metodi") che ne determinano la personalità. Sono inoltre peculiari di questo approccio i concetti di ereditarietà (*inheritance*) e polimorfismo (*polymorphism*) che facilitano la creazione di nuove classi di oggetti da quelle già definite. Si tratta di concetti abbastanza naturali e, in linea di principio, ben mappabili sui modelli umani dei sistemi, anche se talora la loro attuazione pratica non è altrettanto immediata e con i risultati attesi.
Per certi aspetti si può considerare il paradigma ad oggetti come un approccio che confina, nascondendoli all'interno delle classi, gli aspetti procedurali e presenta esternamente una visione di tipo dichiarativo.
Sono estremamente attuali diversi studi sull'approccio ad oggetti per lo sviluppo di sistemi real-time.

3.1.7 TECNICHE DI DESCRIZIONE

Data la notevole varietà di informazioni da descrivere e di obbiettivi perseguiti con tali descrizioni, che si desume dalla breve discussione dei punti precedenti, è naturale che sia opportuno disporre di diverse tecniche descrittive, ognuna delle quali privilegi diversi aspetti.

In particolare, adottando in modo "integrato" tecniche complementari è possibile offrire un più ricco (e quindi chiaro e completo) percorso di lettura a chi deve utilizzare le informazioni descritte.

3.1.7.1 GRAFICHE 2 D e 3 D

Schemi (grafi)
Andamenti nel tempo e nello spazio
Diagrammi (torte, sequenze, sinottici, ...)

Le tecniche grafiche hanno sempre avuto una loro notevole parte nelle comunicazioni di informazioni (supportata anche dal detto “vale più un'immagine di mille parole”) grazie ad alcune notevoli doti, di cui elenchiamo nel seguito le principali.

- **Immediatezza:** gli esseri umani (come del resto molti animali) hanno una particolare capacità di cogliere rapidamente informazioni descritte con immagini, che danno un notevole senso di incisività.

- **Parallelismo**: il “lettore” può cogliere le informazioni nel loro complesso o seguire il filo di indagine che più ritiene significativo in quel momento, libero di commutare abbastanza facilmente il livello di astrazione della sua chiave di lettura dal generale al dettaglio e viceversa.

- **Multidimensionalità**: due dimensioni (e con qualche artificio anche tre) sono meglio di una, soprattutto quando si debba "mappare" la descrizione su oggetti concreti che siamo abituati a pensare in un mondo tridimensionale (anche se spesso proiettato su due dimensioni).
Anche per informazioni astratte è spesso chiarificante poter sviluppare una descrizione in diverse "direzioni ideali".

- **Analogia**: costituisce un ulteriore rinforzo del punto precedente. Indica la capacità di perseguire la somiglianza, almeno “logica”, tra la rappresentazione e l’oggetto rappresentato (ad esempio negli schemi costruttivi).

- **Sinteticità**: proprio grazie al fatto di poter sfruttare con notevole libertà forme, dimensioni, colori, posizioni, ecc. dei simboli, a pari numero di simboli utilizzati si convoglia una maggior quantità di informazione. Si noti la particolare predilezione dell’uomo per insiemi con pochi simboli “ricchi” (motivo per cui, ad es., nei manuali si utilizza spesso la rappresentazione esadecimale invece di quella binaria).

- **Robustezza**: le notevoli ridondanze informative delle tecniche grafiche, che pure non inficiano la sinteticità, sono un valido supporto affinchè anche in presenza di degrado dell’immagine si evitino errori catastrofici di interpretazione.

Ovviamente il successo di una tecnica dipende molto anche dalla disponibilità e dal costo degli strumenti che la supportano, e la ricchezza delle informazioni grafiche e la loro notevole ridondanza porta ad elevati costi e complessità degli strumenti necessari.

La recente evoluzione dell'elettronica e dell'informatica ha però contribuito non poco a rendere accessibili, se non addirittura convenienti, le tecniche grafiche, conferendo loro

anche nuove ed interessantissime doti di facilità di modifica e riproduzione, possibilità di "animazione", ecc.

Le tecniche grafiche sono particolarmente interessanti per la descrizione di aspetti **strutturali**, dove le doti di analogia possono essere ben sfruttate, ma anche per **comportamenti** discreti (Reti di Petri, Automi) e continui (funzioni del tempo, ecc.).

È importante notare che gli strumenti cosiddetti "grafici" possono essere usati per rappresentare veri e propri **grafici** (cioè schemi nelle loro varie forme di aggregazione di "simboli") oppure delle **immagini** che tendono a riprodurre "fotograficamente" degli oggetti.
Quest'ultimo aspetto è attualmente ancora poco utilizzato in campo tecnico e generalmente relegato a quei settori in cui l'immagine assume un'importanza preponderante (es. design di forme).

Si noti anche che molto spesso il ruolo degli strumenti informatici rispetto alla grafica si limita ad un supporto agli uomini disegnatori di descrizioni destinate ad altri uomini, mentre ancora molto resta da fare perchè le tecniche grafiche costituiscano la base di linguaggi di comunicazione dagli uomini alle macchine. Ciò in buona parte è dovuto alla diversità attitudinale tra persone e macchine.

Una tra le eccezioni interessanti nell'ambito dell'informatica industriale è costituita dai linguaggi grafici di programmazione dei PLC (*Programmable Logic Controller*) come gli schemi a contatti (*Ladder diagram*) o a blocchi funzionali.
In questi ed in altri casi (come le reti di Petri, ecc.) si ha anche una definizione (abbastanza) rigorosa della semantica associata agli elementi grafici.

3.1.7.2 FORMULE 1.5 D

Matematiche e logiche

Le formule costituiscono una tecnica ampiamente utilizzata quando si voglia privilegiare il rigore e la sintesi soprattutto per proprietà o comportamenti continui.
In particolare le formule "classiche" possono essere considerate strutture ad una dimensione e mezzo (1.5 D) in quanto ottengono in parte la loro sinteticità dal parziale sfruttamento della seconda dimensione (linee di frazione, simboli di radice, apici e pedici, ecc.) che, tra l'altro, rende migliore la loro immediatezza rispetto alla versione "linearizzata" su una sola dimensione come è generalmente imposto dagli strumenti informatici (linguaggi di programmazione).

Le formule sono generalmente molto adatte ad esprimere le proprietà matematiche, logiche e relazionali.
Il rigore delle formule costituisce, in un certo senso, anche il loro punto debole: non è in genere conveniente formalizzare tutti gli aspetti di una descrizione, dato che ogni scostamento della realtà dai casi "ideali" porta a complessità spesso intrattabili delle formule. Quindi, tranne particolari casi specifici, si utilizzano le descrizioni mediante formule come completamento e precisazione di altre tecniche descrittive, o comunque per descrivere casi "ideali".

I linguaggi di programmazione possono essere considerati una variante "procedurale" delle formule, che invece sono tipicamente di tipo "dichiarativo". Naturalmente

l'estensione generalmente molto maggiore dei programmi rispetto alle normali formule suggerisce l'uso di termini meno succinti e più mnemonici oltre ad una maggior attenzione al "formato", come elemento di chiarezza per il lettore umano ed in taluni casi, come per esempio il linguaggio di programmazione Occam, anche come ulteriore elemento di informazione sintattica per il compilatore.

3.1.7.3 TESTUALI 1 D

monodimensionali
ad albero (indentazione)

I **linguaggi naturali**, sotto forma di testo, costituiscono naturalmente un tipico veicolo di comunicazione di informazioni anche tra persone di diverse estrazioni tecniche e culturali (manager, tecnico, commerciale, cliente, progettista, fornitore, ecc.).

Tra i pregi delle descrizioni sotto forma di testo possiamo citare i seguenti.
- Generalità di comprensione.
- Ricchezza di concetti di "default", cioè pregressi ed evocabili anche con poche parole.
- Possibilità di adattamento a diversi livelli di astrazione, formalismo, ecc.
- Facilità di gestione anche con semplici ed economici strumenti.

Le descrizioni testuali sono tipicamente monodimensionali. Talvolta ciò costituisce un limite che però può venire in parte superato mediante un opportuno uso del "formato". Ad esempio l'indentazione consente una ragionevolmente chiara rappresentazione di strutture ad albero.

3.1.7.4 TABELLARI 2 D

Le tabelle ad una o due dimensioni costituiscono una tecnica molto interessante di descrizione quando la struttura sottostante le informazioni sia particolarmente regolare.
Le descrizioni tabellari si mappano molto bene su strutture dati come ad esempio gli array e gli array di record.
Interessante è anche l'uso di tabelle per descrivere sequenze di azioni e transizioni tra stati, così da costituire un anello di congiunzione tra le descrizioni grafiche e l'eseguibilità da parte di un calcolatore, corredato ovviamente di opportuno interprete di tali tabelle.

3.1.8 FORMALITÀ

Un aspetto importante delle descrizioni è costituito dal loro livello di rigore o, come si suol dire, di "formalità".

3.1.8.1 FORMALI

Sono descrizioni basate su simboli e linguaggi a cui per convenzione è associata una sintassi e una semantica rigorosamente definita.
Su queste descrizioni possono essere basate dimostrazioni di teoremi sulle proprietà.

3.1.8.2 SEMIFORMALI (STRUTTURATE)

Sono descrizioni che utilizzano solo termini e costrutti linguistici che ragionevolmente non presentano ambiguità di interpretazione perchè lasciano imprecisati aspetti su cui esiste un tacito ma generalizzato consenso.

3.1.8.3 INFORMALI

Sono le descrizioni basate sui linguaggi naturali o su disegni che privilegiano una comprensibilità generale e l'immediatezza, spesso evocando concetti non precisi ma di uso comune.

CONSIDERAZIONI.
Di primo acchito sembrerebbe che il pregio universale del rigore renda sempre preferibili le descrizioni formali e quindi meno valide le descrizioni imprecise ed incomplete.
In realtà vi sono validi motivi per preferire in vari casi un livello semiformale o informale di descrizione.

- Generalità: l'uditorio in grado di recepire alcuni formalismi è piuttosto ridotto.
- Molto spesso le informazioni da descrivere non possono raggiungere un completo livello di precisione perchè non sono (ancora) sufficientemente note.
- In diverse situazioni non si ritiene conveniente dedicare un particolare impegno per precisare tutti i dettagli.
- I formalismi più chiari ed eleganti in genere descrivono bene solo situazioni "ideali" ma non rigorosamente realizzabili a livello ingegneristico (superfici -*piane*-, tensioni -*sinusoidali*-, ecc.). D'altra parte formalizzare aspetti come tolleranze, distribuzioni statistiche, overhead, ecc. porta, quando sia anche possibile, a descrizioni praticamente non gestibili.

3.2 FORMALISMI GRAFICI DESCRITTIVI

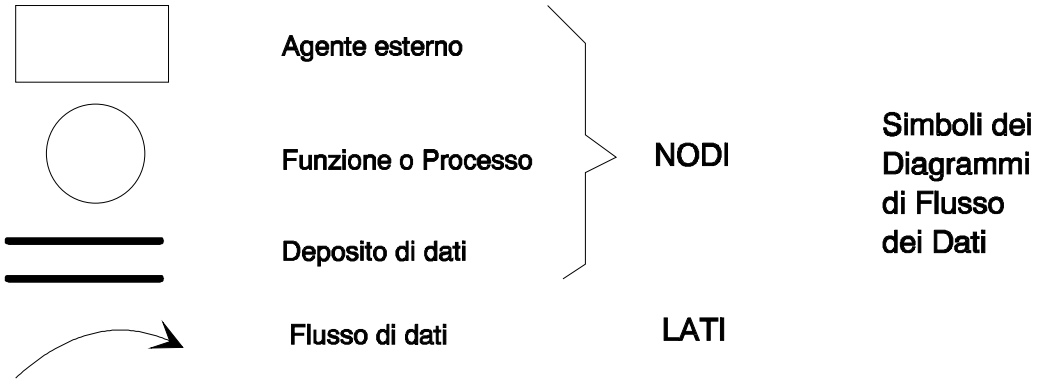
Esistono diversi formalismi espressi da linguaggi grafici e testuali, che si propongono di descrivere le operazioni svolte e il comportamento di sistemi. Le brevi considerazioni che seguono sono rivolte soprattutto a formalismi basati sul paradigma dichiarativo in cui le sequenze evolutive sono espresse in forma potenziale invece che esplicita.

In molti casi questi formalismi sono nati per descrivere ed analizzare proprietà logiche relative alle sequenze di eventi, trascurando gli aspetti di metrica temporale (cioè la capacità di esprimere numericamente durate temporali) che invece sono essenziali per la descrizione e la verifica del comportamento di sistemi in tempo reale. Si tratta comunque di modellizzazioni utili ai livelli di astrazione più elevati e per la verifica di importanti proprietà di correttezza logica e che possono generalmente essere arricchite con precisazioni sugli aspetti temporali quando ciò sia necessario.
Una buona parte dei formalismi è basata su grafi, ed in questi casi è interessante riflettere sulle diverse possibilità di mappaggio di entità (dispositivi, interazioni, azioni, attività, relazioni, ecc.) sui nodi e sui lati di questi grafi.

In questa sede i formalismi sono considerati noti e se ne faranno solo sintetici richiami con alcune considerazioni sulla loro applicabilità per descrivere o realizzare processi e sistemi di automazione.

3.2.1 DATA FLOW DIAGRAMS - DFD

I Diagrammi di Flusso dei Dati sono dei grafi i cui nodi rappresentano trasformazioni o depositi di informazioni (dati) e i cui lati orientati descrivono le “migrazioni” delle informazioni, e quindi implicitamente le relazioni di precedenza tra le operazioni.



Il parallelismo, cioè la concomitanza tra diverse elaborazioni, trova una naturale interpretazione nei DFD, che quindi si prestano bene a descrivere sistemi composti di diversi dispositivi interconnessi, come i sistemi hardware analogici e digitali. Anche la scomposizione gerarchica delle operazioni in operazioni più semplici è facilmente applicabile, consentendo procedimenti di specifica o progetto per raffinamenti successivi.

Il flusso dei segnali di controllo, necessari per descrivere comportamenti discreti, è generalmente sovrapposto (con l’estensione degli schemi di trasformazione - TS = *Transformation Schema*) ed è contrassegnato con linee tratteggiate, ma la sua rappresentazione evidenzia solo quali elementi influenzano quali altri, senza consentire una precisa ricostruzione dell’evoluzione del sistema. Con i DFD viene quindi

privilegiato l'aspetto trasformazionale del sistema, cioè le sue attività, mentre rimangono nascosti gli effetti degli eventi.

3.2.2 AUTOMI

Gli automi a stati finiti costituiscono uno degli strumenti descrittivi più utilizzati, e possono essere definiti come segue.

- Un insieme di **Stati**, di cui uno qualificato come **Stato Iniziale**, a cui appartiene in ogni istante lo stato corrente dell'automa.
- Un insieme di **Simboli** che rappresentano gli ingressi dell'automa, cioè gli stimoli che esso riceve dall'esterno. Questi simboli sono intesi come eventi.
- Una **Funzione di Transizione** che associa uno Stato Prossimo ad una coppia (Ingresso, Stato). Se gli ingressi non sono mutuamente esclusivi si hanno più stati prossimi tra cui scegliere con un criterio (FIFO, priorità, non determinismo, ecc.).
- Un insieme di **Uscite** che, a seconda del tipo di automa, potranno rappresentare dei valori o degli eventi, e una **Funzione di Uscita** che associa le Uscite agli Stati (Automa di Moore) o alle Transizioni (Automa di Mealy).

Una rappresentazione grafica degli automi è costituita dai diagrammi Stati / Transizioni, in cui gli stati sono rappresentati dai nodi di un grafo e le transizioni dai lati orientati e contrassegnati dal simbolo che le attiva.

Gli automi, all'opposto dei DFD, evidenziano l'aspetto sequenziale e reattivo del sistema descritto, trascurando (o rendendo estremamente macchinosa la rappresentazione del) le operazioni di trasformazione delle informazioni. Queste elaborazioni possono però essere associate a procedure opportunamente attivate dalle Uscite.

Un'interessante e generale tecnica implementativa a livello SW basata su automi e che ben si presta, ad esempio, a gestire l'interazione con l'operatore è la seguente.

- Si associa ad ogni **transizione** un'azione, cioè una procedura chiamata quando si esegue la transizione.
- Si associa ad ogni **stato**:
    - Un'azione di **ingresso** che predispone le operazioni. Ad esempio apre una finestra sul video con una maschera.
    - Un'attività **ciclica** che esegue le trasformazioni proprie dello stato. Ad esempio acquisisce da tastiera i caratteri per impostare un parametro della maschera.
    - Un'azione di **uscita** che conclude le operazioni. Ad esempio assegna il valore acquisito alla variabile pertinente e chiude la finestra su video.

I principali **limiti** degli automi sono i seguenti.

- Incapacità di esprimere il parallelismo, se non adottando tanti automi quanti sono i processi sequenziali da rappresentare.
- Rappresentazione "piatta", cioè ad un unico livello di astrazione, che non prevede gerarchia e modularità, e che porta quindi ad un numero di stati intrattabile nei casi di sistemi complessi.
- Assenza del concetto di tempo ed in particolare di durate temporali.

3.2.3 STATECHARTS: un formalismo grafico per la specifica di sistemi complessi

Gli Statecharts sono stati proposti per mantenere gli interessanti concetti di stati e transizioni, cercando però di superare i limiti espressivi degli automi classici.
Si tratta di un formalismo grafico per descrivere stati e transizioni con le seguenti caratteristiche principali.

- Orientato alla definizione di ***sistemi reattivi*** *event driven* continuamente sollecitati da stimoli interni e esterni
- Rappresentazione di gerarchie tra stati che evidenziano particolari relazioni tra stati e facilitano la visione a diversi livelli di astrazione con aggregazioni modulari. In particolare consente ad esempio la rappresentazione della gerarchia tra *modi*, *fasi* e *passi* dei sistemi, a cui si è accennato nel cap. 2.
- Rappresentazione della concorrenza (parallelismo tra stati) secondo il concetto di ortogonalità, che rende più naturale e basata su un minor numero di stati la descrizione di sistemi anche relativamente complessi.
- Il concetto di *history* associabile ad ogni gruppo di stati, che rende più compatta e naturale la descrizione di comportamenti che dipendono dalla storia passata del gruppo.

Esempi telefonia, *automotive*, reti di comunicazione, sistemi operativi, sistemi militari e avionici, interfacce uomo macchina di **molti sistemi software** commerciali.
Problema descrivere il comportamento in maniera **chiara**, **realistica**, ma anche **formale** e **rigorosa** (vogliamo poter **simulare** la specifica).

3.2.3.1 Come si specifica il comportamento di un sistema reattivo

Bisogna indicare l'insieme di sequenze di ingresso/uscita accettate dal sistema, e cioè:

- gli eventi
- le condizioni
- le azioni

in più bisogna considerare i vincoli temporali.
C'è accordo nel mondo della ricerca a considerare stati ed eventi come una maniera sufficientemente naturale per descrivere il comportamento dinamico di un sistema complesso.

La forma tipica per esprimere il comportamento di un sistema è una collezione di frammenti del tipo: "quando nello stato *A* accade l'evento *γ,* se in quell'istante è verificata la condizione *P,* allora il sistema si sposta nello stato *C"*
Esempi:

- *Quando il pilota automatico è inserito e si tira la leva x si passa al controllo manuale*
- *Mentre viene visualizzata l'ora, se si preme il pulsante y l'orologio mostra la data*

Diagrammi Stati-Transizioni

Sono la maniera formale per catturare questi frammenti e darne una visione globale. Sono grafi orientati, i cui nodi denotano gli stati, e i cui archi indicano le transizioni.

Agli archi sono associate delle etichette che riportano gli eventi che li fanno scattare e le condizioni di *guardia* che ne consentono l'effettiva transizione.

In un sistema complesso il numero degli stati cresce esponenzialmente, quindi una rappresentazione *piatta* non è gestibile.
Una rappresentazione adatta anche per sistemi complessi deve essere **modulare, gerarchica e strutturata**.

3.2.3.2 Cosa deve poter gestire un buon formalismo

1. Mantenere e migliorare l'impatto grafico dei diagrammi stati-transizioni.
2. Raggruppare stati diversi in un *superstato*
   Es. in **ogni stato** dell'aereo, se si tira la leva gialla il sedile viene espulso
3. Introdurre *indipendenza* ovvero *ortogonalità* tra diversi moduli
   Es. la scatola del cambio può cambiare di stato **indipendentemente** dal sistema frenante
4. Consentire l'adozione di *transizioni più generali* di quelle etichettate con un solo evento
   Es. quando si preme il pulsante di selezione, si entra nello stato selezionato
5. Permettere di *raffinare* la definizione di uno stato
   Es. la modalità di visualizzazione consiste nella visualizzazione dell'ora, della data e del cronometro

STATECHARTS rappresenta un *formalismo grafico* per descrivere stati e transizioni

- in maniera modulare
- consentendo di raggruppare e raffinare gli stati
- garantendo la possibilità di definire moduli paralleli (ortogonalità)
- incoraggiando lo *zoom in/out* per valutare la specifica a diversi livelli di astrazione

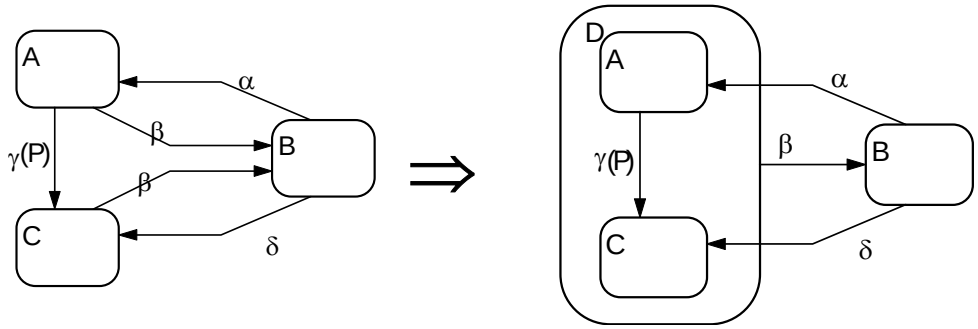
STATECHARTS = state diagrams + depth + orthogonality + broadcast

3.2.3.3 Aggregazione (clustering) di stati

L'aggregazione di stati è significativa quando gruppi di stati presentano un comportamento comune rispetto alle transizioni, azioni o attività.

Esempio.

> L'evento β porta il sistema in B sia da A che da C. Quindi si possono raccogliere (*cluster*) questi due stati in un nuovo super-stato D da cui esce un'unica transizione globale β.

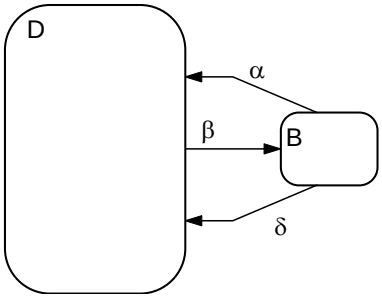


D corrisponde all'*OR esclusivo* di A e C (è una vera e propria *astrazione* di A e C). Cattura un aspetto comune ad A e C, cioè il fatto che l'evento β porta a B.

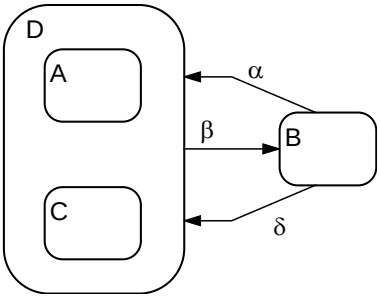
3.2.3.4 Raffinamento di stati - Decomposizione XOR

Il raffinamento di uno stato consiste nell'individuare degli stati interni che lo compongono. La decomposizione XOR (mutua esclusione) rappresenta gli stati interni con la convenzione che il sistema si trova in un certo istante in uno solo di essi.

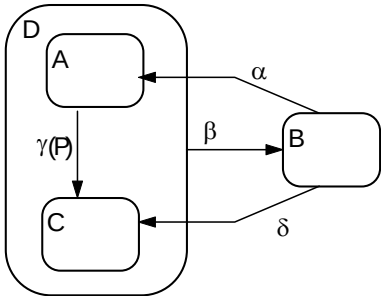
Esempio.



Situazione iniziale in cui D e B sono i soli stati



Raffinamento di D in A e C (α e δ non sono ancora del tutto specificate)



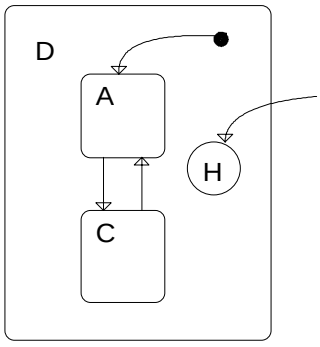
Ulteriore raffinamento di α e δ e introduzione di γ

Il **raffinamento** corrisponde ad un procedimento **Top down**

L'**aggregazione** corrisponde ad un procedimento **Bottom up**

3.2.3.5 Ingresso con storia (Enter by history)

Una situazione che si presenta spesso è la seguente: quando si entra in un gruppo di stati si deve entrare nello stato che è stato *visitato* più di recente. (o in quello di *deafault* se si entra per la prima volta)



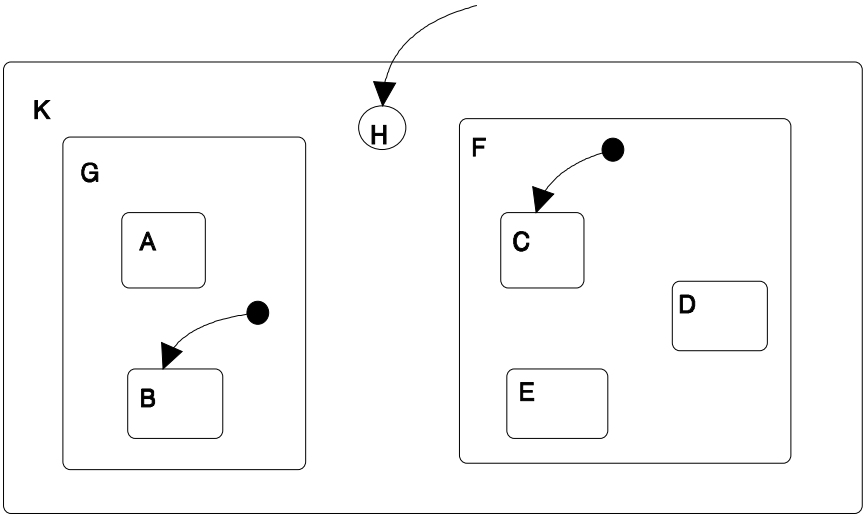
L'ingresso di *default* è indicato con una freccia originata da un pallino nero.

L'ingresso basato sulla storia viene indicato mediante una H (*history*) racchiusa in un circoletto

Nella figura accanto, la prima volta che si entra in D si va in A (*default*), mentre le volte successive si andrà in A o C in base allo stato in cui si era quando si è usciti da D l'ultima volta (*history*).

Storia e livelli

L'ingresso in base alla storia precedente si applica solo al livello in cui si trova l'ingresso con H, come descritto nell'esempio seguente.



In questo esempio la storia (transizione con **H**) consente di scegliere solo tra G e F, mentre le scelte di livello inferiore sono forzate dalle transizioni di default:

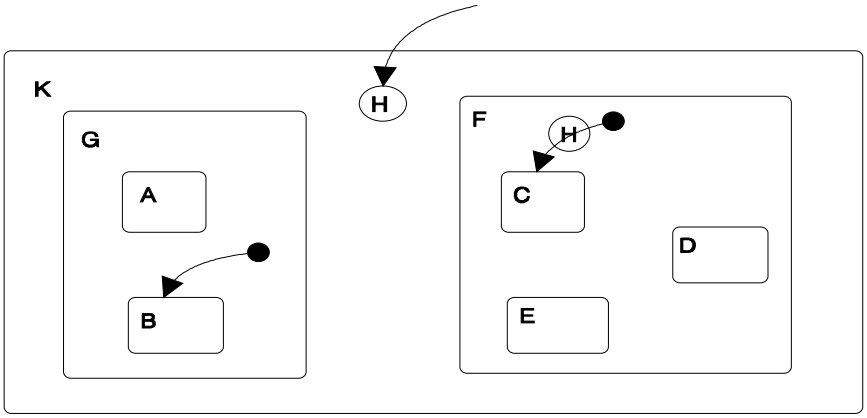
- se l'ultimo stato occupato in K è stato uno tra A e B, allora rientra in B
- se l'ultimo stato occupato in K è stato uno tra C, D ed E, allora rientra in C

Applicazione della scelta basata sulla storia a tutti i livelli sottostanti
Se si contrassegna con un asterisco (**H***) l'ingresso in base alla storia si applica al livello in cui si trova H* **e** a *tutti i livelli sottostanti*.

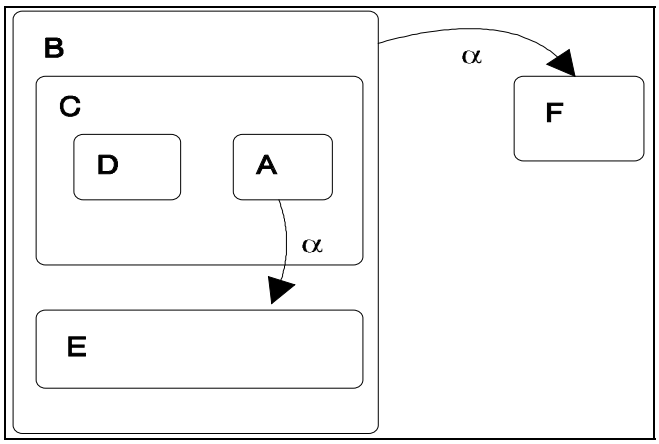
Immaginiamo uno schema come il precedente, ma con H* al posto di H. In questo caso il sistema entra nell'ultimo tra gli stati A, B, C, D ed E che ha visitato. La storia prevale su **entrambe** le frecce di default.

Storia e livelli: una soluzione intermedia
Aggiungendo degli elementi **H** si può specificare una situazione intermedia rispetto ai due estremi detti sopra.

- se l'ultimo stato occupato in K è stato G (cioè A o B), allora rientra in B (default)
- se l'ultimo stato occupato in K è stato C, D o E, allora rientra in questo (history)



Attenzione ad evitare contraddizioni !



Le transizioni che escono da uno stato sono quelle che partono dal suo contorno, **ma anche** quelle che partono dal bordo dei super-stati che lo contengono

α ha un comportamento ambiguo quando siamo in A

3.2.3.6 Decomposizione AND

Quando si raffina uno stato e si intende indicare che al suo interno il sistema si trova contemporaneamente (parallelismo) in più sottostati, si adotta la decomposizione AND.

> **Quando il sistema si trova in uno stato, allora si trova in tutti i suoi componenti AND**

Notazione: box diviso in due da una linea tratteggiata.

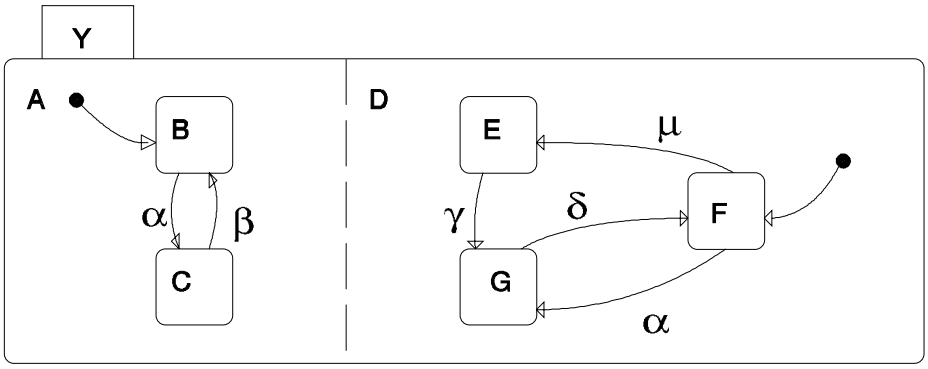
Y è composto da A e D, cioè lo stato Y corrisponde a una combinazione di B o C con E, F o G.

Decomposizione AND, ovvero *prodotto ortogonale*
Si dice che Y è il *prodotto ortogonale* di A e D.

Si noti che A e D sono super-stati normali. Quando si entra in Y ci si trova nella combinazione ⟨B,F⟩ a causa delle frecce di default.

Sincronizzazione: α provoca *contemporaneamente* le transizioni $B \to C$ e $F \to G$ ovvero $\langle B,F\rangle \to \langle C,G\rangle$

Indipendenza: μ influenza solo la parte D, provocando $F \to E$ quale che sia lo stato di A



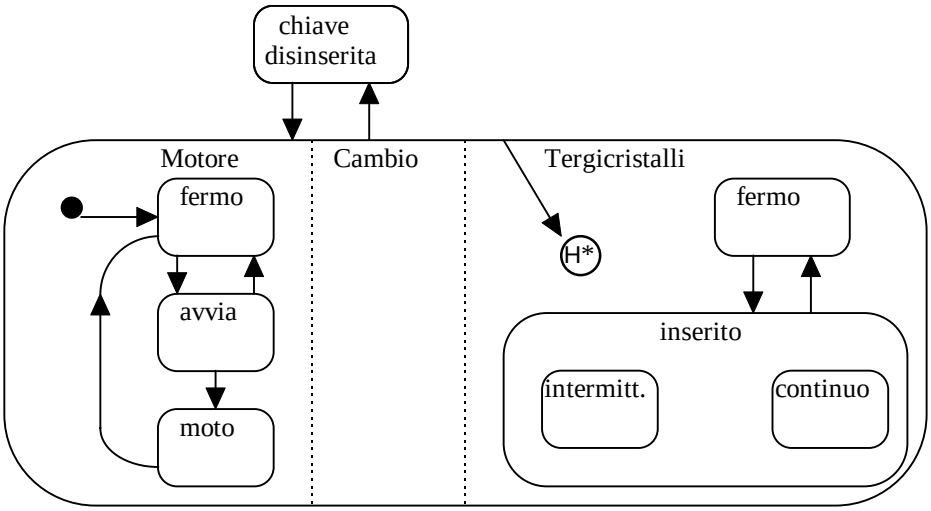
La descrizione dello stato Y in una notazione non provvista della decomposizione AND richiederebbe un numero totale di stati pari al prodotto degli stati ortogonali, invece che alla somma. Nel nostro esempio occorrerebbero 6 (= 2 × 3) stati e molte più transizioni.

Il normale prodotto tra automi, infatti, è di solito *disgiunto*, mentre in questo caso è possibile introdurre una forma di dipendenza tra i diversi componenti.

Applicazioni della *ortogonalità*

È possibile dividere uno stato considerando i diversi componenti fisici del sistema

Es. Automobile: modalità generale di funzionamemto + componenti ortogonali per i sottosistemi (motore, cambio, tergicristalli, ecc…)



3.2.3.7 Azioni ed attività

Quello che abbiamo visto fino ad ora è soltanto una rappresentazione della parte di controllo in cui la *reattività* del sistema è definita solo attraverso il suo cambiare di stato in risposta ad eventi esterni. Per completare la descrizione del comportamento del sistema occorre anche precisare cosa il sistema fa negli stati e nelle transizioni.

> Una ***azione*** generata dal sistema corrisponde a un *evento* in ingresso: rappresenta qualcosa che si considera avvenire **istantaneamente** (zero time).

Una ***attività*** corrisponde invece a una *condizione*: richiede sempre una certa quantità di tempo per essere svolta.

Azioni e attività sono introdotte sfruttando le idee degli automi di Mealy (azioni associate alle transizioni) e di Moore (azioni associate all'ingresso in uno stato). Inoltre è possibile associare anche attività alla permanenza in uno stato e azioni all'uscita da uno stato.

Come associare azioni e attività a transizioni e stati

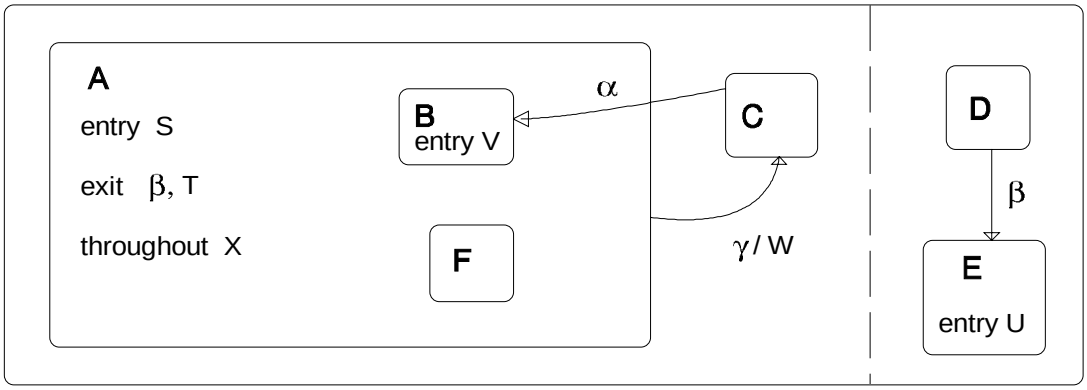
L'etichetta di una transizione può essere del tipo $\alpha(P)/S$

- α è l'evento che fa scattare la transizione
- P è la condizione di guardia che deve essere verificata affinché la transizione scatti
- S è l'azione che viene eseguita quando scatta la transizione

In uno stato possiamo invece definire

1. l'azione da effettuare all'ingresso (*entry*)
2. l'azione da effettuare in uscita (*exit*)
3. l'attività da svolgere finché si resta nello stato (*throughout*). Si noti come questo sia equivalente a specificare l'inizio dell'attività (*start*) all'ingresso dello stato e la fine (*stop*) all'uscita dallo stato.

Un semplice esempio



Nella figura dell'esempio abbiamo:

S - azione di ingresso in A

X - attività durante la permanenza in A

β - azione interna in uscita da A

T - azione esterna in uscita da A

In ⟨C,D⟩ α provoca ⟨C,D⟩ → ⟨B,D⟩ e l'esecuzione **simultanea** di *S* e *V*

In ⟨B,D⟩ γ provoca ⟨B,D⟩ → ⟨C,E⟩ e l'esecuzione di *W*, *T*, **β** e *U*, in particolare l'esecuzione dell'azione β (*interna* per la parte sinistra, ma *esterna* per la parte ortogonale destra) provoca la transizione D → E e la conseguente esecuzione di *U*.

3.2.4 RETI DI PETRI

Le reti di Petri hanno riscosso un notevole interesse per la loro capacità di descrivere l'evoluzione di sistemi concorrenti e per la loro rappresentazione grafica di facile comprensione. Sono state proposte numerose varianti di reti di Petri, in particolare per dotarle della capacità di descrivere aspetti temporali (*Timed Petri Nets*).
Nella rappresentazione grafica le reti di Petri sono dei grafi orientati, i cui nodi sono di due tipi: **Posti** (Place) e **Transizioni** (Transition) e i cui lati devono collegare nodi di tipo diverso.
Lo stato della rete è rappresentato da una **Marcatura**, che consiste in una particolare distribuzione di **Token** (contrassegni, gettoni) sui Posti e una particolare Marcatura rappresenta lo stato iniziale della rete. Lo stato delle reti di Petri è quindi di tipo **distribuito**, contrariamente al caso degli automi a stati.

- I Posti sono rappresentati con dei piccoli cerchi, al cui interno possono essere indicati con dei pallini neri gli eventuali *token* di marcatura.
- Le transizioni sono rappresentate da barrette.
- Le connessioni tra Posti e Transizioni sono rappresentate da frecce.

Le diverse varianti delle reti di Petri si differenziano per le regole di **Abilitazione** e di **Scatto** (Firing) delle Transizioni, che nelle reti temporizzate coinvolgono anche il trascorrere di opportuni intervalli di tempo.

- L'Abilitazione di una Transizione dipende solo dalla Marcatura dei Posti in ingresso e nel caso più semplice richiede la presenza di un Token in ognuno dei Posti in ingresso.
- Lo Scatto di una Transizione può avvenire solo se essa è abilitata e produce una variazione locale della Marcatura, ad esempio con il consumo di un Token da ogni Posto in ingresso e la produzione di un Token in ogni Posto in uscita della Transizione stessa.

1) - Un possibile mappaggio degli elementi di una rete di Petri su entità reali può essere il seguente.

- I Token rappresentano delle risorse concrete (periferiche, aree di memoria, messaggi, ecc.) o astratte (eventi verificatisi).
- I Posti rappresentano *attività* in esecuzione (se marcati) o latenti (se non marcati), oppure semplici contenitori di informazioni pieni o vuoti (se marcati o no).
- Le Transizioni operano i trasferimenti di risorse con le opportune sincronizzazioni.
- Questo mappaggio privilegia il ruolo dei Posti, ai quali sono associati i processi (software) che effettuano le elaborazioni utili impiegando un tempo non nullo, mentre le Transizioni rappresentano azioni ausiliarie di coordinamento che possono essere pensate come svolte (idealmente in modo istantaneo) da un Sistema Operativo. Questa interpretazione ricorda un pò un automa di Moore.

2) - Un mappaggio, in un certo senso duale del precedente, può attribuire le elaborazioni utili, che assumono così la forma di *azioni*, alle Transizioni, mentre considera i Posti con la loro Marcatura come dei semplici indicatori di eseguibilità delle azioni che il Sistema Operativo utilizza per lanciare correttamente l'esecuzione delle azioni. Con questa interpretazione è in genere richiesto che si possa attribuire alle Transizioni un tempo di scatto non nullo, mentre è irrilevante la durata delle marcature. Si può fare un parallelo con gli automi di Mealy.

3.2.4.1 ESTENSIONI TEMPORALI DELLE RETI DI PETRI

Ricordiamo brevemente le principali varianti delle reti di Petri con estensioni temporali, adatte alle diverse interpretazioni dei Posti e delle Transizioni.

Attività associate ai Posti.

Ad ogni posto è associato un tempo di esecuzione. Le Transizioni sono abilitate solo se in tutti i loro Posti di ingresso è trascorso il tempo minimo di marcatura.

Azioni associate alle Transizioni.

Ad ogni Transizione è associato un tempo di esecuzione, e lo scatto si svolge in tre fasi:

consumo dei Token dai Posti in ingresso
durata dello scatto per il tempo prefissato
produzione dei Token nei Posti in uscita.

Eventi (istantanei) associati alle Transizioni.

Ad ogni Transizione è associata una finestra temporale (Tmin, Tmax) che determina l'intervallo di tempo dopo la sua abilitazione in cui la Transizione può e deve scattare.

3.2.5 GRAFCET

Il GRAFCET è un linguaggio grafico di origine francese relativamente recente (standardizzato nel 1982 con norma UTE NFC 03-190), che si propone di evidenziare il "flusso di controllo", cioè l'evoluzione temporale sia sequenziale che parallela, e si propone come strumento di descrizione sia di specifiche che di progetti.

Per questo linguaggio sono definiti due livelli:

1) - Livello **funzionale** "astratto".
2) - Livello **tecnologico** che riporta i dettagli implementativi con riferimenti precisi a sensori, attuatori, tecnologie, ecc.

Il GRAFCET si ispira alle reti di Petri, che tanta attenzione hanno suscitato per la loro possibilità di descrivere evoluzioni temporali discrete in parallelo.
Analogamente alle reti di Petri si utilizzano grafi con archi orientati che collegano nodi di due tipi diversi:

- **Passi** o **Tappe** (Etapes in francese)
- **Transizioni** (Transitions).

3.2.5.1 -- SPECIFICHE DEL GRAFCET

3.2.5.1.1 Elementi del Grafcet.

Gli elementi che compongono uno schema Grafcet sono i seguenti.

- Passi (o fasi o tappe) a cui sono associate delle azioni.
- Transizioni a cui sono associate delle condizioni di transizione (o ricettività).
- Collegamenti orientati fra Passi e Transizioni.

--- PASSI

I Passi sono rappresentati da quadrati numerati nella parte superiore.

L'entrata in un Passo è rappresentata da un segmento verticale nella parte superiore del simbolo, mentre l'uscita è rappresentata da un segmento verticale nella parte inferiore.

I Passi Iniziali, che possono essere uno o più, sono rappresentati da quadrati con contorno a doppia linea.

-- AZIONI

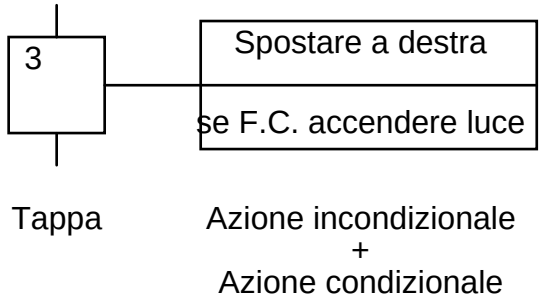


Fig. 3.1 Azioni associate a passi.

Le Azioni sono descritte in forma testuale all'interno di rettangoli alla destra del Passo corrispondente. La descrizione è astratta nel livello 1 (funzionale), mentre cita simboli concreti (I/O, ecc.) nel livello 2 (tecnologico).

Allo stesso Passo possono essere associate più azioni. La possibilità di indicare Azioni condizionali, con la condizione eventualmente data dall'attivazione di altri Passi, può rendere meno facile la lettura di uno schema, ma consente un risparmio anche notevole nel numero di Passi necessari alla descrizione del comportamento di un certo sistema.

Si noti che le Azioni del GRAFCET possono corrispondere sia ad azioni (considerate istantanee) che ad attività (considerate continuative), secondo la definizione data nel precedente capitolo 2.

-- TRANSIZIONI

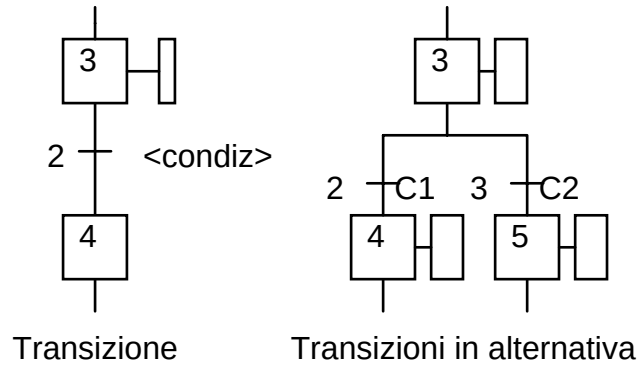


Fig. 3.2 Transizioni tra passi.

Una transizione tra due Passi si rappresenta con una sbarretta perpendicolare alla linea di collegamento.

Le Transizioni possono essere contrassegnate da un numero (univoco) posto a sinistra della sbarretta.

Le Condizioni di Transizione sono scritte a parole (livello 1) o come espressione booleana (livello 2) a destra del simbolo di Transizione e possono citare stati esterni o stati interni (passi attivi).

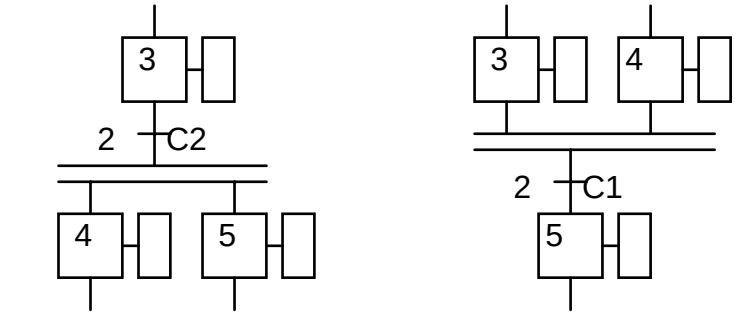


Fig. 3.3 - Attivazione e terminazione in parallelo.
L'attivazione in parallelo dei passi 4 e 5 avviene quando il passo 3 è attivo e la condizione C2 è verificata.
La terminazione parallela attiva il passo 5 quando entrambi i passi 3 e 4 sono attivi e si verifica la condizione C1.

Allorchè più Passi sono collegati a valle o a monte di una stessa Transizione, la confluenza è rappresentata con una doppia barra orizzontale (fig. 3.3).

--- COLLEGAMENTI ORIENTATI

Sono rappresentati con linee verticali e orizzontali. Per convenzione il senso è dall'alto verso il basso, a meno che non sia esplicitamente indicato con una freccia.

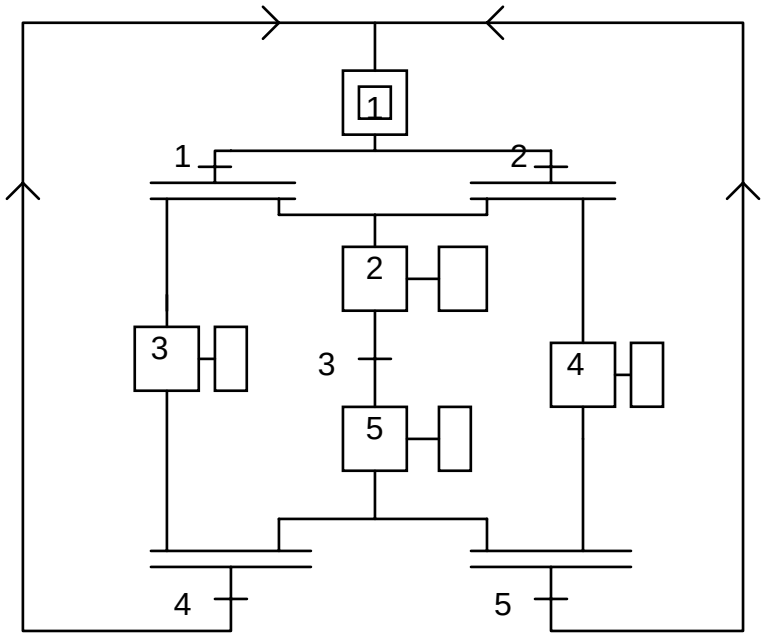


Fig. 3.4 - Esempio di schema GRAFCET.

3.2.5.1.2 Regole di evoluzione del Grafcet

1) - Situazione iniziale.
Generalmente è costituita dall'insieme dei Passi Iniziali che sono attivi nello stato di riposo.

2) - Superamento di una Transizione.
L'evoluzione della situazione corrisponde al superamento delle Transizioni, che si produce quando:
- la Transizione è valida, cioè il Passo (o i Passi) a monte è attivo;
- la condizione associata alla Transizione è vera.

3) - Evoluzione dei Passi attivi.
Il superamento di una Transizione comporta contemporaneamente l'attivazione di tutti i Passi immediatamente seguenti e la disattivazione di tutti i Passi immediatamente precedenti.

4) - Evoluzione simultanea.
Più Transizioni contemporaneamente superabili sono evase simultaneamente.
Questa regola consente di scomporre un Grafcet in più diagrammi non connessi graficamente, pur di introdurre esplicitamente nelle Condizioni di Transizione gli stati attivi dei passi che in tal modo determinano le interazioni tra le diverse parti.

5) - Attivazione e disattivazione simultanea.
Se in seguito al simultaneo superamento di più Transizioni uno stesso Passo deve essere disattivato e attivato, esso resta attivo.

3.2.5.2 LIVELLO 1 - SPECIFICHE FUNZIONALI

Il primo livello di Grafcet descrive il comportamento della parte di comando nei confronti della parte operativa. Si parla qui di "ricettività" del sistema in termini di capacità di ricevere informazioni dall'esterno, e di "attività" nel fornire comandi ed informazioni all'esterno.
Queste capacità si trasformeranno, nel livello 2, in ben precisi sensori (ricettività) ed attuatori (attività).
Le specifiche funzionali permettono quindi, in fase di ideazione, di analizzare:
- ciò che l'automatismo deve fare, di fronte alle varie situazioni che si possono presentare;
- quale dovrà essere il comportamento della parte di comando da costruire.

Occorre dunque definire in modo chiaro e preciso le diverse funzioni, le informazioni e i comandi coinvolti nell'automazione della parte operativa e nella sicurezza del funzionamento previsto.

Per contro, nel livello 1, la scelta della tecnologia non è necessaria. Non importa sapere, a questo punto, se uno spostamento sarà realizzato con un organo pneumatico o con un motore elettrico; ciò che interessa è determinare in quali circostanze questo dovrà verificarsi. (ad es. azione = muovi slitta).

3.2.5.3 LIVELLO 2 - SPECIFICHE TECNOLOGICHE

Il secondo livello completa le esigenze funzionali con specifiche tecnologiche ed operative, precisando le condizioni di funzionamento dei sensori ed attuatori reali.

Le specifiche tecnologiche precisano cioè le condizioni nelle quali l'automatismo dovrà fisicamente inserirsi nell'ambiente del sistema da automatizzare (ad es. attiva uscita U).

A queste specifiche di interfacciamento possono aggiungersi anche specifiche più generali come: temperatura, umidità, tensione di alimentazione, rischi di esplosione, ecc.

3.3 LOGICA, CALCOLO DEI PREDICATI E TRIO

La logica è uno strumento per formalizzare il ragionamento umano e in particolare si presta a descrivere proprietà e relazioni di individui e oggetti.

In questo paragrafo sono riportati brevi richiami di Calcolo Proposizionale e di Calcolo dei Predicati del Primo Ordine, come introduzione al linguaggio logico TRIO.

3.3.1 Richiami di Calcolo Proposizionale

La versione più semplice della Logica è costituita dal *calcolo proposizionale* i cui operandi sono asserzioni elementari, atomiche, *vere* o *false* e i cui operatori sono i connettivi logici.
Le proposizioni sono rappresentate simbolicamente da identificatori
Esempi:

- "oggi piove" ⇔ *piove*
- "oggi il tempo è nuvoloso" ⇔ *nuvoloso*
- "oggi è soleggiato" ⇔ *sole*

Connettivi logici

Le proposizioni complesse si ottengono per composizione di altre proposizioni usando i connettivi logici.

nome	simbolo	significato
not	$\neg$	negazione
and	$\wedge$	congiunzione
xor	$\oplus$	disgiunzione esclusiva
or	$\vee$	disgiunzione inclusiva
implies	$\rightarrow$	implicazione
coimpl	$\leftrightarrow$	doppia implicazione

Significato visualizzato con *tabelle di verità*

Il significato (valore di verità) delle proposizioni complesse è calcolabile in base al valore di verità delle proposizioni componenti e al tipo di connettivi. Un modo classico per descrivere i risultati di questo calcolo, definendo così il significato dei connettivi, è basato sulle cosiddette tabelle di verità, di cui si riportano quelle relative ai connettivi più comunemente usati.

TABELLA DELLA VERITA'

A	B	$\neg$ A	A $\wedge$ B	A $\vee$ B	A $\oplus$ B	A $\rightarrow$ B	A $\leftrightarrow$ B
T	T	F	T	T	F	T	T
T	F	F	F	T	T	F	F
F	T	T	F	T	T	T	F
F	F	T	F	F	F	T	T

Priorità degli operatori logici

Come in aritmetica, viene assegnata una *priorità* relativa tra gli operatori

Operatori logici: priorità massima $\neg$,
poi $\wedge$,
poi $\oplus$ e $\vee$,
poi $\rightarrow$ e $\leftrightarrow$.

Esempio $\neg$ piove $\wedge \neg$ nuvoloso $\rightarrow$ sole

ovvero: se non piove e non è nuvoloso allora è soleggiato, viene interpretata come

$$((\neg \text{ piove}) \wedge (\neg \text{ nuvoloso})) \rightarrow \text{sole}$$

Note sull'**implicazione logica**
L'implicazione '$\rightarrow$' modella il ragionamento deduttivo

I seguenti significati sono tutti equivalenti:

1. se A, allora B;
2. A vale solo se vale B;
3. A è condizione sufficiente per B (o una sua possibile causa);
4. B è condizione necessaria per A (B segue necessariamente da A).

NB A $\rightarrow$ B è vera anche se A è falsa e B è vera.
Esempio piove $\rightarrow$ bagnato (*bagnato* $\equiv$ **"il cortile di casa mia è bagnato")**
Il cortile può essere bagnato anche se non piove (il vicino di casa ha lavato la macchina).

Limitazioni del calcolo proposizionale:

- ogni asserzione riferita a un oggetto è rappresentata in modo elementare con una formula atomica distinta che "incapsula" l'oggetto facendone perdere l'identirà ed "esporta" esclusivamente un valore di verità.
- non possiede quindi formule che parlino, o predichino su insiemi di individui, per esprimere relazioni o proprietà universali (vere per tutti) o esistenziali (verificate da almeno un individuo)
- non può descrivere insiemi di oggetti, (eventualmente infiniti) né modellare numerosi aspetti della matematica

3.3.2 La logica dei predicati del primo ordine

Le formule atomiche della Logica dei Predicati sono più articolate rispetto a quelle della Logica delle Proposizioni, ed in particolare possono

- riferirsi a individui (costanti o variabili),
- calcolare nuovi valori a partire da valori dati
- mettere in relazione (insiemi di) individui.
- generalizzare asserzioni (universalmente ed esistenzialmente)

3.3.2.1 Alfabeto della logica del primo ordine

- **Variabili** (*w, x, y, ...*): denotano individui.
  A ogni variabile viene assegnato un solo valore.
  Universo o *dominio*: insieme di tutti i possibili valori di una variabile.

  Esempio:
  variabile *x*,
  universo = insieme $\mathbb{Z}$ dei numeri interi
  x può avere il valore -5.

- **Funzioni** (*f, g, h, ...* o simboli appositi: '+' per la somma).
  Ogni funzione ha una sua "arità" *n* e restituisce un valore $f(x_1,.. x_n)$
  Per le funzioni sono adottate sia la notazione *prefissa* (es. *add(x, y)*), che la notazione *infissa* con operatori (es. $x + y$)

  Esempio:
  $add(n1, n2) \equiv n1 + n2$
  funzione '+' tra interi: operazione binaria (arità 2) di somma.

  Caso particolare: funzioni ad arità nulla: le *costanti*

  Esempio**:**
  costante '3'

- **Predicati** (*p, q, r, ...*o con i simboli appositi, es '>'): rappresentano relazioni (sottoinsiemi del prodotto cartesiano dei domini degli argomenti)

  Predicato *n*-ario applicato ad *n* argomenti ≡ formula atomica

  Un predicato costituisce la più semplice formula del primo ordine.
  Anche per i predicati sono adottate sia la notazione *prefissa* (es. *maggiore(x, y)*), che la notazione *infissa* con operatori (es. $x > y$)

  Esempio:
  relazione '>' tra numeri interi: '>' $\subset Z \times Z$
  Tutte le coppie *(a,b)* tale che *a* è maggiore di *b*

  - 3 > -1
  - x > 4 è vera solo se x ha un valore maggiore di 4.

- **Connettivi proposizionali** ($\neg$, $\wedge$, $\vee$, $\oplus$, ecc.)
  Le formule atomiche vengono composte con i connettivi proposizionali

  Esempio: $(x > 3 \wedge 3 > z) \rightarrow x > z$

- **Quantificatore universale '$\forall$'**
  Date la variabile *x* e la formula *A*, la formula

$\forall$xA (per ogni x vale A)

è vera se la formula A è vera per qualsiasi valore di x

- **Quantificatore esistenziale '$\exists$'**

  $\exists xA$ (esiste x tale che *A*)

  è vera se esiste almeno un valore di x che rende vera A

  Esempio**:** $\forall x \, \forall z \, ((x > 3 \wedge 3 > z) \rightarrow x > z)$ è vera sugli interi

La **priorità** dei quantificatori è inferiore a '$\vee$' e superiore a '$\rightarrow$'.

Commutatività dei quantificatori

$\forall x \forall y \, Q(x,y)$	è equivalente a	$\forall y \forall x \, Q(x,y)$
$\exists x \exists y \, Q(x,y)$	è equivalente a	$\exists y \exists x \, Q(x,y)$
$\forall y \, \exists x \, P(x,y)$	**non** è equivalente a	$\exists x \, \forall y \, P(x,y)$
ognuno y ha un padre x		esiste qualcuno x che è padre di tutti y

Distributività dei quantificatori

- Il quantificatore universale è distributivo rispetto alla congiunzione
- Il quantificatore esistenziale è distributivo rispetto alla disgiunzione

$\forall x \, (P(x) \wedge Q(x))$	è equivalente a	$\forall x \, P(x) \wedge \forall x \, Q(x)$
$\exists x \, (P(x) \vee Q(x))$	è equivalente a	$\exists x \, P(x) \vee \exists x \, Q(x)$
$\forall x \, (P(x) \vee Q(x))$	**non** è equivalente a	$\forall x \, P(x) \vee \forall x \, Q(x)$
$\exists x \, (P(x) \wedge Q(x))$	**non** è equivalente a	$\exists x \, P(x) \wedge \exists x \, Q(x)$

3.3.2.2 Formule

Una formula con **tutte le variabili quantificate** è detta **formula chiusa**, o **sentenza**.

La verità di una formula chiusa non dipende dal valore delle variabili, ma **definisce una proprietà** delle costanti e del dominio di interpretazione.

Una formula che contiene *n* variabili non quantificate **definisce una relazione *n*-aria** sull'universo di interpretazione

Esempi**:**

$\exists$ k (x * k = 294) definisce i divisori del numero 294

relazione 1-aria su x

$\forall$ x (x $\neq$ 1 $\rightarrow$ $\neg$ $\exists$ y (y $\neq$ 1 $\wedge$ x * y = 193)) asserisce che 193 è un numero primo

proprietà della costante 193

$\exists$ z $\forall$ x (x $\neq$ 1 $\rightarrow$ $\neg$ $\exists$ y (y $\neq$ 1 $\wedge$ x * y = z)) asserisce che esistono numeri primi

proprietà dei numeri naturali

La verità di una formula dipende anche dal dominio di interpretazione

Esempio: $\forall$ x $\exists$ y (y < x) ($\equiv$ non esiste un elemento minimo)
è vera per i numeri interi **Z**, non per i naturali N.

3.3.3 Logica tipizzata

- Contempla l'esistenza di più domini di interpretazione
- Ad ogni variabile è associato un tipo ($\equiv$ dominio)

Esempio di logica tipizzata

P insieme delle persone

N insieme dei numeri naturali (età delle persone)

Variabili

p ed *f* di tipo persona (p, f $\in$ P)

ep ed *ef* di tipo età (ep, ef $\in$ N)

Predicati

genitore $\subseteq$ P^2 relazione tra persone

età $\subseteq$ P $\times$ N assegna a ogni persona la sua età

Formula

$\forall$p $\forall$f $\forall$ep $\forall$ef (genitore (p, f) $\wedge$ età (p, ep) $\wedge$ età (f, ef) $\rightarrow$ ep > 10 + ef)

Che significa: ogni genitore ha almeno dieci anni più di ogni proprio figlio.

Notazioni alternative per indicare i tipi delle variabili:

$\forall_P$ p $\forall_P$ f $\forall_N$ ep $\forall_N$ ef (genitore (p, f) $\wedge$ età (p, ep) $\wedge$ età (f, ef) $\rightarrow$ ep > 10 + ef)

Oppure si indica esplicitamente il tipo con l'appartenenza ad un insieme

$\forall$p$\in$P $\forall$f$\in$P $\forall$ep$\in$N $\forall$ef$\in$N
(genitore (p, f) $\wedge$ età (p, ep) $\wedge$ età (f, ef) $\rightarrow$ ep > 10 + ef)

3.3.4 Definizione del linguaggio TRIO

TRIO è un linguaggio del primo ordine aumentato con operatori temporali per esprimere proprietà che cambiano nel tempo.

Mentre la Logica Temporale (qui non richiamata) esprime solo relazioni temporali (prima, dopo, sempre, mai, ecc.) TRIO ha la capacità, essenziale per poter descrivere proprietà di sistemi in tempo reale, di far riferimento ad una **metrica del tempo**, cioè di poter esprimere quantitativamente distanze temporali.

Ogni formula di TRIO ha un significato (valore di verità) rispetto a un **istante di tempo corrente** che viene lasciato **implicito** nella formula

3.3.4.1 Sintassi di TRIO

3.3.4.1.1 *Variabili*:

- dipendenti dal tempo (locali) - DT
- indipendenti dal tempo (globali) - IT

ogni variabile α è associata a un *dominio* (il linguaggio è tipizzato)

$D(\alpha)$ ≡ insieme dei suoi possibili valori.

Un dominio particolare è il *Dominio Temporale T*, di natura numerica, che può essere discreto o denso.

3.3.4.1.2 *Funzioni e predicati*:

- dipendenti dal tempo (locali) - DT
- indipendenti dal tempo (globali) - IT

3.3.4.1.3 *Simboli*:

- proposizionali $\neg, \wedge$
  da cui sono derivabili gli altri, come $\vee$, etc.
- quantificatore universale $\forall$
  da cui è derivabile quello esistenziale $\exists$
- operatori temporali *Futr* e *Past*
  da cui sono derivabili gli altri operatori presentati nel seguito.

3.3.4.1.4 Definizione induttiva di *termini*

Sono termini di TRIO:

- una variabile o una costante
- una funzione *n*-aria applicata ad *n* termini
- se *s* è un termine e *t* un termine di tipo temporale
  sono termini anche *Futr*(*s*,*t*) e *Past*(*s*,*t*)

3.3.4.1.5 Definizione induttiva di *formule*

Sono formule di TRIO:

- formula atomica: un predicato *n*-ario applicato ad *n* termini
- se *A* e *B* sono formule,
  sono formule anche $\neg A$, $A \wedge B$, $A \vee B$, etc.
- se *A* è una formula e *x* una variabile indipendente dal tempo,
  sono formule anche $\forall x A$ e $\exists x A$
- se *A* è una formula e *t* un termine di tipo temporale,
  sono formule anche *Futr*(*A*,*t*) e *Past*(*A*,*t*)

3.3.4.1.6 Operatori temporali Futr e Past

Abbiamo visto che gli operatori temporali *Futr* e *Past* possono essere applicati a termini (producendo un termine) o a formule (producendo una formula). Data la loro simmetria le definizioni date per *Futr* sono immediatamente trasformabili in quelle per *Past*, sostituendo al vocabolo "futuro" il vocabolo "passato".

Futr e Past applicati a Termini
Futr(s, t) ≡ valore del termine s all'istante futuro distante t dall'istante corrente.

Futr e Past applicati a Formule
Futr(A, t) ha nell'istante corrente il valore di verità che la formula A assume all'istante futuro distante t dall'istante corrente.

NOTA - L'istante corrente non è globale per tutte le componenti di una formula, ma è locale ad ogni componente, ed anzi l'effetto degli operatori temporali è quello di "spostare" l'istante corrente della sottoformula a cui sono applicati.

3.3.4.2 Operatori temporali derivati

AlwF(A)	$\stackrel{def}{=}$	$\forall t\ (t > 0 \rightarrow Futr\ (A, t))$
AlwP(A)	$\stackrel{def}{=}$	$\forall t\ (t > 0 \rightarrow Past\ (A, t))$
SomF (A)	$\stackrel{def}{=}$	$\neg\ AlwF\ (\neg A) \equiv \exists t\ (t > 0 \wedge Futr\ (A, t))$
SomP (A)	$\stackrel{def}{=}$	$\neg\ AlwP\ (\neg A) \equiv \exists t\ (t > 0 \wedge Past\ (A, t))$
Lasts (A, t)	$\stackrel{def}{=}$	$\forall t'\ (0 < t' < t \rightarrow Futr\ (A, t'))$
Lasted (A, t)	$\stackrel{def}{=}$	$\forall t'\ (0 < t' < t \rightarrow Past\ (A, t'))$
Sometimes (A)	$\stackrel{def}{=}$	$SomP\ (A) \vee A \vee SomF\ (A)$
Always (A)	$\stackrel{def}{=}$	$AlwP\ (A) \wedge A \wedge AlwF\ (A)$
Until (A_1, A_2)	$\stackrel{def}{=}$	$\exists t\ (\ t > 0 \wedge Futr\ (A_2, t) \wedge Lasts\ (A_1, t)\ )$
Since (A_1, A_2)	$\stackrel{def}{=}$	$\exists t\ (\ t > 0 \wedge Past\ (A_2, t) \wedge Lasted\ (A_1, t)\ )$
$Until_w$ (A_1, A_2)	$\stackrel{def}{=}$	$AlwF\ (A_1) \vee Until\ (A_1, A_2)$
$Since_w$ (A_1, A_2)	$\stackrel{def}{=}$	$AlwP\ (A_1) \vee Since\ (A_1, A_2)$
NextTime (A, t)	$\stackrel{def}{=}$	$Futr\ (A, t) \wedge Lasts\ (\neg A, t)$
LastTime (A, t)	$\stackrel{def}{=}$	$Past\ (A, t) \wedge Lasted\ (\neg A, t)$

3.3.4.3 Esempio: linea di trasmissione con un ritardo fissato.

Descrizione informale

I messaggi entranti nella linea vengono emessi con un ritardo di 5 unità di tempo.

Formalizzazione

$in\ (m) \rightarrow Futr\ (out\ (m), 5)$	il messaggio m non viene perso e viene emesso dopo 5 unità di tempo
$\forall m\ (in\ (m) \rightarrow Futr\ (out\ (m), 5))$	specifica - nessun messaggio viene perso
$out\ (m) \rightarrow Past\ (in\ (m), 5)$	il messaggio m non è stato generato in modo spurio
$\forall m\ (out\ (m) \rightarrow Past\ (in\ (m), 5))$	specifica - nessun messaggio è generato in modo spurio

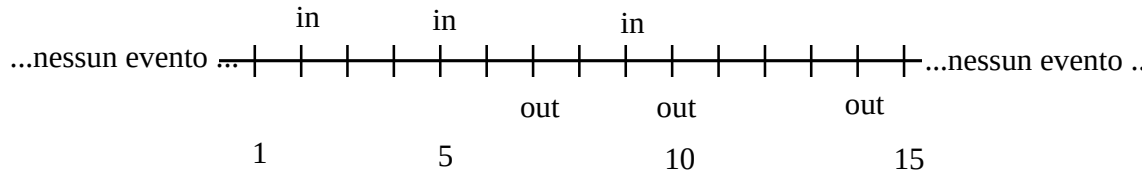
Composizione delle due (sotto)specifiche

$\forall m\ (out\ (m) \leftrightarrow Past\ (in\ (m), 5))$	tutti i messaggi vengono emessi se e solo se sono stati ricevuti 5 istanti prima

Chiusura *temporale* della specifica

$Always\ (\forall m\ (out\ (m) \leftrightarrow Past\ (in\ (m), 5)))$	La proprietà vale sempre (in qualsiasi istante)

Visualizzazione delle strutture di interpretazione



3.3.4.4 Always & Sometimes

L'operatore *Always* esprime una **quantificazione universale di tipo temporale**

L'operatore *Sometimes* è duale e dà una **quantificazione temporale esistenziale**

Esempio:
c'è un istante in cui arriva un messaggio

$$\text{Sometimes } (\exists m \ \text{in}(m))$$

Formule classicamente chiuse
sono tutte le formule in cui tutte le variabili sono quantificate.

Formule temporalmente chiuse
se tutti i predicati e tutte le variabili sono indipendenti dal tempo
oppure se sono delle forme Always(A) o Sometimes(A)

- Il loro valore di verità non dipende dall'istante in cui vengono valutate perciò sono dette tempo invarianti
- Rappresentano proprietà *temporali* dei sistemi specificati (assunti invarianti nel tempo)

Una SPECIFICA è una formula chiusa (classicamente e temporalmente)

Si ricordi che solo le formule chiuse hanno un valore di verità, mentre le formule con variabili libere esprimono relazioni.

3.3.4.5 Esempi di specifiche in TRIO

- Tutti gli **eventi** sono rappresentati con predicati dipendenti dal tempo, con eventuali argomenti caratterizzanti l'evento, che assumono valore *true* negli istanti di tempo in cui si verificano.
- Gli **stati** variabili nel tempo (cioè la maggior parte) sono rappresentati con predicati dipendenti dal tempo che assumono il valore true in tutti gli istanti dell'intervallo (discreto o denso o continuo) in cui lo stato è verificato.

3.3.4.5.1 Lampada a pulsante

La pressione del pulsante causa l'accensione, due secondi dopo, della lampada, che rimane accesa per cinque secondi. Il pulsante rimane disattivato per 8 secondi.

Formalizzazione:

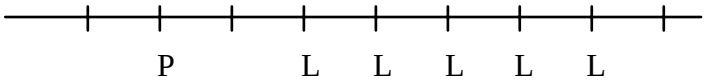
pressione del pulsante	$\Leftrightarrow$	predicato di evento *push*
lampadina accesa	$\Leftrightarrow$	predicato di stato *lamp*

Accensione causata dalla pressione

$$push \rightarrow Futr\ (Lasts\ (lamp, 5), 2)$$

Si noti che l'operatore di implicazione non esclude che la lampadina possa essere accesa anche per altre cause e non solo in seguito alla pressione del pulsante.
Si noti inoltre che l'istante corrente di Lasts(lamp, 5) è spostato nel futuro di 2 unità di tempo rispetto all'istante corrente di push, per effetto dell'operatore Futr.

Modello



P = push L = lamp

A rigore, con un modello di tempo discreto, se si intende indicare che la lampadina è accesa fin dal primo istante del periodo di 5 unità di tempo, occorre usare una variante dell'operatore temporale Lasts (A,t) come nella formula

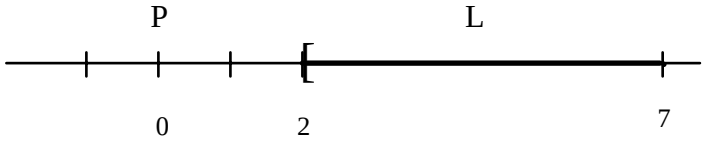
$$push \rightarrow Futr\ (Lasts_{ie}\ (lamp, 5), 2)$$

dove $Lasts_{ie}(A, t) \stackrel{def}{=} \forall s\ (0 \leq s < t \rightarrow Futr\ (A, s))$

include anche l'istante presente (s = 0).

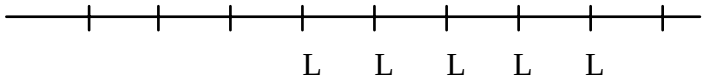
Analogamente si possono definire $Lasts_{ii}$, $Lasts_{ei}$, etc. per includere o escludere uno o entrambi gli estremi di un intervallo (ii = incluso, incluso; ei = escluso, incluso).

La formula è corretta anche in riferimento a un dominio temporale denso.



La formula dà una condizione *sufficiente* ma non necessaria per l'accensione della lampadina.

È vera (con l'implicazione soddisfatta in modo *vacuo*) anche nel modello seguente.



Disattivazione del pulsante.
Specifichiamo ora che una volta premuto il pulsante non può essere premuto per 8 istanti

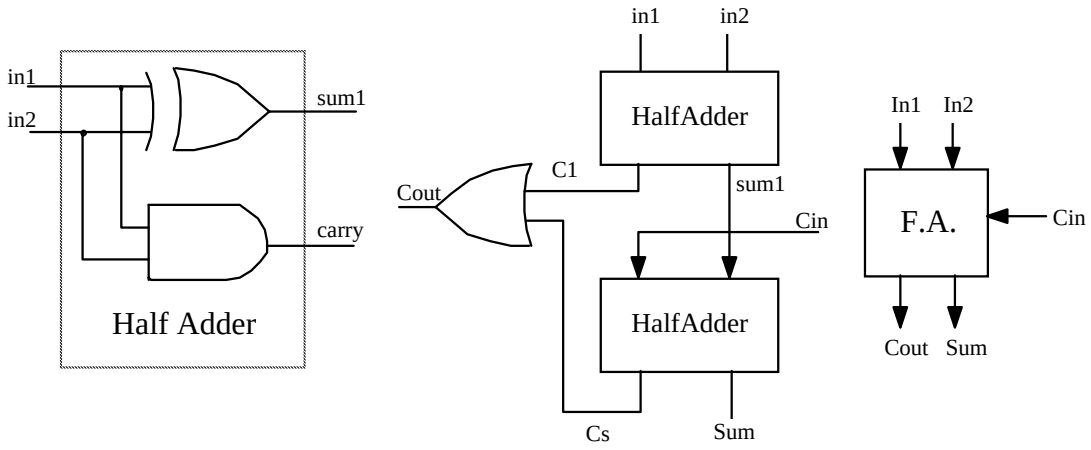
$$push \rightarrow Lasts (\neg push, 8)$$

Per una descrizione più raffinata dovremo tenere conto del fatto che nulla impedisce che il pulsante sia premuto nuovamente prima di 8 secondi, ma specificheremo che tali pressioni non saranno efficaci. Per fare ciò introduciamo un nuovo predicato *validPush*: = pressione col pulsante abilitato

$$validPush \leftrightarrow push \wedge Lasted (\neg validPush, 8)$$

Per tener conto di questo raffinamento della specifica baster‡ sostituire nelle formule precedenti *validPush* al posto di *push*

3.3.4.5.2 Un esempio hardware: half e full adders



Specifica TRIO per half adder

I ritardi delle porte XOR e AND sono rappresentati dalle costanti $del_{\oplus}$ e $del_{\wedge}$

$$\text{HalfAdder}(in1,in2,sum1,carry) \stackrel{def}{=}$$
$$\text{Alw}((sum1 \leftrightarrow \text{Past}(in1 \oplus in2, del_{\oplus})) \wedge$$
$$(carry \leftrightarrow \text{Past}(in1 \wedge in2, del_{\wedge})))$$

Specifica composita per full adder

$$\text{FullAdder}(in1,in2,C_{in},Sum,C_{out}) \stackrel{def}{=}$$
$$\text{HalfAdder}(in1,in2,sum_1,C_1) \wedge$$
$$\text{HalfAdder}(C_{in},sum_1,Sum,C_s) \wedge$$
$$\text{Alw}\ (C_{out} \leftrightarrow \text{Past}\ (C_1 \vee C_s, del_{\wedge}))$$

Descrizione algoritmica del full adder
Si considerano due distinti ritardi del_s e del_c per le operazioni di somma e generazione del riporto.

$$\text{FullAdder}(in1,in2,C_{in},Sum,C_{out}) \stackrel{def}{=}$$
$$\text{Alw}\ ((C_{out} \leftrightarrow \text{Past}\ (in1 \wedge in2 \vee (in1 \oplus in2) \wedge C_{in}\ ,\ del_c)) \wedge$$
$$\wedge\ (Sum \leftrightarrow \text{Past}\ (in1 \oplus in2 \oplus C_{in},\ del_s)))$$

3.3.4.5.3 Esempio: Ascensore

Questo esempio è tratto da: “Sistemi in Tempo Reale” di Angelo Morzenti, ed. Clup.
Un ascensore serve *N* piani; all'interno ci sono *N* bottoni, uno per ogni piano. A ogni piano sono disposti due bottoni, per i piani superiori e inferiori. Si trascura la descrizione dei meccanismi di apertura e di chiusura delle porte. La strategia di gestione è descritta informalmente dai seguenti punti:

- Mantiene la stessa direzione finché non occorre più proseguire.
- Inverte il senso di marcia se deve visitare piani in direzione opposta
- Se non occorre espletare alcun servizio, rimane fermo in sosta all'ultimo piano toccato fino alla successiva chiamata o richiesta.
- Si ferma a un piano se e solo se il piano è stato richiesto dall'interno, oppure dal piano è avvenuta una chiamata, e la direzione è la stessa in cui si sta muovendo l'ascensore, oppure la direzione è opposta ma l'ascensore non è richiesto a nessuno dei piani successivi.

Alfabeto della specifica: predicati DT

- **pushLiftButton(floor)**
  predicato di evento, con floor∈[1..N], indica lo schiacciamento del bottone dall'interno dell'ascensore, per richiedere la visita del piano *floor* ;
- **pushFloorButton(floor, way)**
  predicato di evento, con floor∈[1..N], way∈{up, down}, indica la pressione del bottone al piano, per la richiesta di un passaggio in una data direzione;
- **arrives(floor)**
  predicato di evento, con floor∈[1..N], indica l'arrivo dell'ascensore al piano floor;
- **stops(floor)**
  predicato di evento, con floor∈[1..N], indica la fermata dell'ascensore al piano *floor*;
- **leaves(floor)**
  predicato di evento, con floor∈[1..N], indica la dipartita dell'ascensore, dal piano *floor*, in una direzione non direttamente specificata;
- **floorButton(floor, way, s)**
  predicato di stato, con floor∈[1..N], way∈{up, down}, s∈{on, off} indica lo stato acceso/spento dei bottoni ai piani;
- **liftButton(floor, s)**
  predicato di stato, con floor∈[1..N], s∈{on, off} indica lo stato acceso/spento dei bottoni interni all'ascensore;
- **liftPosition**
  variabile di stato, appartenente all'intervallo [1..N], indica la posizione corrente dell'ascensore, o più precisamente, come risulterà dalla specifica, l'ultimo piano toccato;
- **dir**
  variabile di stato, appartenente all'insieme {up, down, stnd}, indica il senso di marcia dell'ascensore, che non coincide necessariamente con il senso della

velocità cinematica: per esempio, dir = up se l'ascensore è arrivato, salendo, a un piano, e vi si è fermato, ma al momento dell'arrivo era già prenotato a un altro piano che sta più in alto, e quindi dovrà proseguire.

Stato ascensore ed espletamento dei servizi

D.1 Un piano è attualmente *richiesto* se è stato schiacciato, in passato, il bottone corrispondente dall'interno dell'ascensore, ma da allora non è mai avvenuta una sosta a quel piano:

$$\text{requestedFloor}(floor) \stackrel{def}{=} \exists t\ (\text{Past}(\text{pushLiftButton}(floor), t) \wedge \text{Lasted}(\text{liftPosition} \neq floor, t+\Delta))$$

D.2 L'ascensore è attualmente *chiamato* da un certo piano in una direzione se il bottone al piano è stato schiacciato in passato e da allora l'ascensore non si è più trovato al quel piano, avendo la stessa direzione:

$$\text{calledFrom}(floor, way) \stackrel{def}{=} \exists t\ (\text{Past}(\text{pushFloorButton}(floor, way), t) \wedge \text{Lasted}(\neg(\text{liftPosition} = floor \wedge dir = way), t+\Delta))$$

D.3 Al momento corrente, all'ascensore è stato richiesto di spostarsi ai piani inferiori, o rispettivamente superiori

$$\text{neededDown} \stackrel{def}{=} \exists floor\ (floor < \text{liftPosition} \wedge (\text{requestedFloor}(floor) \vee \exists d\ (\text{calledFrom}(floor, d))))$$

$$\text{neededUp} \stackrel{def}{=} \exists floor\ (floor > \text{liftPosition} \wedge (\text{requestedFloor}(floor) \vee \exists d\ (\text{calledFrom}(floor, d))))$$

Formule di specifica

R.1 Parte cinematica: l'ascensore arriva a un certo piano se e solo se è disceso da un piano superiore o è salito da un piano inferiore; il tragitto da un piano all'altro viene svolto in un tempo costante Δt_b, indipendente dalla direzione di moto.

R.1.1 $\forall floor\ \text{Always}(\text{leaves}(floor) \wedge dir = up \leftrightarrow \text{Futr}(\text{arrives}(floor+1) \wedge dir = up, \Delta t_b) \wedge \text{Lasts}(dir = up, \Delta t_b))$

R.1.2 $\forall floor\ \text{Always}(\text{leaves}(floor) \wedge dir = down \leftrightarrow \text{Futr}(\text{arrives}(floor-1) \wedge dir = down, \Delta t_b) \wedge \text{Lasts}(dir = down, \Delta t_b))$

si noti come la direzione di moto rimane invariata fino all'istante di arrivo al piano, incluso.

R.2 L'ascensore, arrivato a un piano, si ferma se e solo se: il piano è stato prenotato dall'interno dell'ascensore, oppure al piano è avvenuta una chiamata nella stessa direzione in cui l'ascensore si sta muovendo, oppure l'ascensore non deve proseguire ai piani successivi.

$$\forall floor\ Always\ (stops(floor) \leftrightarrow arrives(floor) \wedge (requested(floor) \vee \vee \exists way(calledFrom(floor, way) \wedge way = dir) \vee \vee (dir = down \wedge \neg neededDown \vee dir = up \wedge \neg neededUp)))$$

R.3 Se l'ascensore si ferma a un piano, vi rimane almeno per un tempo uguale alla costante Δt_s. Inoltre, mantiene la direzione di marcia invariata per tutto il tempo Δt_s della sosta: la direzione di marcia sarà "ridecisa" allo scadere dell'intervallo Δt_s in funzione delle chiamate da soddisfare

R.3.1 $\forall floor\ Always\ (stops(floor) \wedge dir = up \leftrightarrow Lasts(liftPosition = floor \wedge dir = up, \Delta t_s))$

R.3.2 $\forall floor\ Always\ (stops(floor) \wedge dir = down \leftrightarrow Lasts(liftPosition = floor \wedge dir = down, \Delta t_s))$

R.4 Quando, al termine dell'intervallo minimo di sosta a un piano, l'ascensore non è richiesto in nessun altro piano, esso cambia la propria direzione in *stnd*, e si ferma al quel piano fino a che non viene nuovamente chiamato.

$$\forall floor\ Always\ (Becomes(dir = stnd) \wedge Until_w(dir = stnd \wedge liftPosition = floor, neededUp \vee neededDown) \leftrightarrow Past(stops(floor), \Delta t_s) \wedge \neg(neededUp \vee neededDown))$$

L'operatore Becomes viene qui introdotto per ragioni di leggibilità. *Becomes (P)* significa che *P* è diventato vero al momento corrente, cioè che è vero ora, e non era vero negli ultimi istanti precedenti. Sicuramente, si ha $\Delta t_s > \Delta$, e possiamo quindi assumere per *Becomes* una definizione approssimata quale la seguente:

$$Becomes(P) \stackrel{def}{=} P \wedge Past(\neg P, \Delta)$$

R.5 Nel caso di sosta a un piano con chiamate o richieste pendenti da altri piani, l'ascensore si ferma al piano per un tempo costante, Δt_s. Allora l'ascensore parte da un certo piano in una data direzione, ad es. verso il basso, se e solo se: è arrivato Δt_s istanti prima allo stesso piano provenendo dall'alto, e ci sono chiamate pendenti nella stessa direzione di marcia, oppure è arrivato dal basso ma tutte le chiamate pendenti sono in direzione opposta, oppure se è stato fermo per un certo tempo e si verifica una richiesta di servizi ai piani inferiori.

$$\forall floor\ Always(\ leaves\ (floor) \leftrightarrow Past\ (stops\ (floor) \wedge dir = down, \Delta t_s) \wedge neededDown \wedge dir = down \vee \vee Past\ (stops\ (floor) \wedge dir = up, \Delta t_s) \wedge neededUp \wedge dir = up \vee$$

$\vee$ Past (stops (floor) $\wedge$ dir = up, Δt_s) $\wedge \neg$neededUp $\wedge$ neededDown $\wedge$ dir = down $\vee$
$\vee$ Past (stops (floor) $\wedge$ dir = down, Δt_s) $\wedge \neg$neededDown $\wedge$ neededUp $\wedge$ dir = up $\vee$
$\vee$ Since ($\neg$(neededUp $\vee$ neededDown), Becomes(dir = stnd)) $\wedge$
$\wedge$ ((neededDown $\wedge \neg$neededUp $\wedge$ dir = down) $\vee$
$\vee$ (neededUp $\wedge \neg$neededDown $\wedge$ dir = up) $\vee$
$\vee$ (neededDown $\wedge$ neededUp)))

R.6 Lo stato di accensione dei bottoni ai piani e nell'ascensore può essere immediatamente specificato in funzione dei predicati *calledFrom* e *requestedFloor*, rispettivamente: il bottone è illuminato se e solo se la richiesta originata dall'ultima pressione non è ancora stata soddisfatta.

$\forall$floor Always ((liftButton(floor, on) $\leftrightarrow$
requestedFloor(floor)) $\wedge \forall$d(floorButton(floor, d) $\leftrightarrow$
calledFrom(floor, d)))

R.7 Condizione al contorno per il primo e l'ultimo piano, dai quali non è ovviamente possibile scendere e salire, e quindi non ha senso premere i corrispondenti bottoni esterni:

Always($\neg$pushFloorButton(1, down) $\wedge \neg$pushFloorButton(N, up))

La specifica del sistema dell'ascensore consiste nella congiunzione delle regole R.1 ÷ R.7.

Proprietà del sistema
Liveness: qualsiasi richiesta viene, prima o poi, soddisfatta:

$$\forall floor\ Always(pushLiftButton(floor) \wedge liftPosition \neq floor \rightarrow SomF(stops(floor))).$$

Precedenza: un "equo" trattamento delle richieste di servizio. Viene servito prima un piano che è stato richiesto prima.

$$\forall floor_1\ \forall floor_2\ Always(before(pushFloorButton(floor_1), pushFloorButton(floor_2)) \wedge floor_1 \neq floor_2 \rightarrow before(stops(floor_1), stops(floor_2)))$$

Vincoli di tipo quantitativo
Qualsiasi richiesta di visita di un piano può essere servita entro un tempo massimo inferiore a quello necessario a percorrere l'intero edificio in entrambi i sensi, fermandosi a ogni piano:

$$\forall floor\ Always(pushLiftButton(floor) \wedge liftPosition \neq floor \rightarrow \exists t(t < 2 * (\Delta t_b+\Delta t_s) * N \wedge Futr(stops(floor), t)))$$

Condizioni sotto le quali il tempo di attesa per una chiamata da un piano può essere uguale a quello minimo: l'ascensore si sta muovendo nella stessa direzione indicata dalla chiamata, avvicinandosi al piano:

$$\forall floor\ \forall way\ Always(pushFloorButton(floor, way) \wedge floor \neq liftPosition \wedge \exists t(t \leq |floor-liftPosition| * (\Delta t_b+\Delta t_s) \wedge Futr(stops(floor), t)) \leftrightarrow dir=way \wedge (dir = up \wedge liftPosition < floor \vee dir = down \wedge liftPosition > floor))$$

3.4 ESEMPI DI DESCRIZIONI

%%%%%%%%%%%%%%%%
FORME D'ONDA
SEQUENZE
EVOLUZIONI PARALLELE
CORRELAZIONI CAUSE-EFFETTI
FLUSSI DI DATI
EVOLUZIONE DI VALORI NUMERICI
- DIGITALI
- CON BARRE
- CON ANDAMENTO (TREND)

ANIMAZIONE

3.5 ESERCIZI

-- Discutere le diverse proprietà desiderabili in un linguaggio di specifica e in un linguaggio adatto a descrivere implementazioni.

-- Perchè sono desiderabili descrizioni a più livelli?.

-- Analizzare il problema delle specifiche indirette, dedotte cioè da specifiche dirette e vincoli realizzativi.

-- Qual'è il paradigma più adatto a descrivere strutture?.

-- Qual'è il paradigma più adatto a descrivere comportamenti?. Discutere la differenza tra comportamenti trasformazionali e comportamenti reattivi.

-- E' più facile mappare su un sistema distribuito (vari processori interconnessi) un comportamento descritto con un diagramma Data-flow o con uno schema Grafcet?. Motivare la risposta.

-- Provare a descrivere con un automa il comportamento di un ascensore che debba servire 4 piani.

-- Individuare una tecnica descrittiva ad automa, adatta a descrivere il comportamento di un ascensore che debba servire N (generico) piani. Si suggerisce di adottare di utilizzare variabili ausiliarie, pile e code per ridurre l'esplosione del numero di stati.

-- Si raffini la descrizione degli esempi precedenti, per tener conto delle fasi di accelerazione e di frenata dell'ascensore, e per avere la chiusura delle porte ad ascensore fermo.

-- Si descriva il comportamento di un ascensore con uno schema Statecharts, e si discutano pro e contro rispetto alla descrizione con un automa.

-- Si descriva mediante Statecharts e mediante Grafcet l'evoluzione di un sistema composto da un trapano e da due carrelli che rispettivamente consegnano e asportano i pezzi da perforare e perforati. Si adottino due livelli di astrazione, con il livello più raffinato che evidenzi anche fasi intermedie di avvicinamento, estrazione, accelerazione e decelerazione.

-- Rispetto al problema precedente (trapano con carrelli) si individui il massimo numero possibile di anomalie rilevabili dal comportamento temporale, e si scrivano in TRIO le formule che le descrivono. Esempio di anomalia: dopo 3 secondi dalla completa estrazione della punta il carrello evacuatore non ha ancora asportato il pezzo.

3.6 ALTRE LETTURE

C. Ghezzi, A. Fuggetta, S. Morasca, A. Morzenti, M. Pezzè
INGEGNERIA DEL SOFTWARE. - Progettazione, sviluppo e verifica.
Mondadori. 1991

M. Thomas
The industrial use of formal methods.
Microprocessors and Microsystems,Vol.17,N.1,1993
Interessante articolo tutorial sull'uso di metodi formali nell'informatica industriale.

D. Harel
Statecharts: a visual formalism for complex systems
Science of Computer Programming - N. 8 - 1987 - North Holland

P.T. Ward
The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing.
IEEE Trans. on Software Eng. - Vol. 12 - N.2 - Feb. 1986

M. Blanchard
Comprendre maitriser et appliquer le GRAFCET
Cepadues Editions 1979
Presentazione del linguaggio grafico GRAFCET.

A. Morzenti
SISTEMI IN TEMPO REALE. Proposta di un formalismo logico di specifica e convalida.
Ed. Clup 1991
162 pagg. Teorico generale.

Jean Paul Calvez
EMBEDDED REAL-TIME SYSTEMS
John Wiley & Sons 1993
Metodologie e modelli - Fasi di vita Sw.Eng.
Spunti ed informazioni interessanti. Poco operativo

Alan Burns, Andy Wellings
REAL-TIME SYSTEMS AND THEIR PROGRAMMING LANGUAGES
Addison-Wesley 1990
Problemi applicativi SW. Impostazione culturale.
Ada, Modula-2 Occam-2. Molto interessante.