Cap. 11

11. MODELLI E TECNICHE IMPLEMENTATIVE DI BASE

In questo capitolo vengono presentate tecniche implementative adatte per realizzare sistemi reattivi. Sono detti sistemi reattivi quelli caratterizzati da un comportamento che è dettato da eventi, che costituiscono stimoli provenienti dai sistemi esterni che interagiscono con essi.

Le tecniche presentate consentono di descrivere in modo **regolare**, generalmente con tabelle o altre strutture dati, il comportamento voluto, in modo che un processo software, che potremmo chiamare in generale "motore", impostato in modo indipendente dall'applicazione possa funzionare come un **interprete di queste tabelle** e con la loro esecuzione produrre il comportamento voluto.

Come è naturale aspettarsi le strutture dati costituiscono i legami tra gli ingressi e i comportamenti che il sistema reattivo deve assumere.

AUTOMI

Gli automi a stati sono adatti a descrivere e realizzare comportamenti che reagiscono ad una sequenza di stimoli visti come simboli di ingresso.

La loro memoria storica è basata sullo stato interno, che naturalmente può cambiare come effetto degli stimoli.

SISTEMI A REGOLE

I sistemi a regole considerano il mondo esterno come rappresentativo di una collezione di "fatti" di volta in volta verificati o meno. Sulla base di un elenco di regole si deducono eventuali fatti derivati e si produce l'esecuzione di azioni.

Le regole, che determinano il comportamento, sono considerabili come una base di conoscenza, e come tale, oggetto di eventuali elaborazioni di apprendimento.

TABELLE DI DECISIONE

Le tabelle di decisione sono un'interessante e compatta variante per la rappresentazione delle regole, che facilita la verifica di proprietà di completezza e consistenza.

SISTEMI FUZZY

I sistemi a regole *fuzzy* sono un'estensione molto interessante dei sistemi a regole. Il loro pregio sta nella capacità di modellare l'imprecisione in modo vicino alle attitudini umane.

La loro capacità di trattare anche valori numerici ne fa un interessante modello implementativo per sistemi di automazione e controllo, non solo per modellare le azioni discrete ma anche per modellare le attività continue anche con dipendenze non lineari dagli ingressi.

11.1 AUTOMI

Gli automi a stati finiti sono stati presentati nel cap. 3 come modello **descrittivo** di evoluzione sequenziale, ma si prestano molto bene ad essere utilizzati anche come riferimento **implementativo**.
Nel seguito vengono brevemente presentati gli approcci funzionali sincrono e asincrono e le tecniche implementative a programma distribuito e a tabelle.

11.1.1 AUTOMI SINCRONI (Time Driven)

Il clock è rappresentato dalla chiamata ciclica della procedura che esegue l'automa.
Gli ingressi sono degli **stati**.
I simboli di ingresso sono rappresentati da funzioni combinatorie degli ingressi.
Si possono avere più simboli di ingresso contemporaneamente TRUE: si risolve il non determinismo concettuale con scelte implementative.
Si presta ad una gestione con temporizzatore e polling.

La funzionalità sincrona in generale prevede un segnale di *clock* che determina gli istanti temporali in cui sono valutati gli ingressi ed effettuate le eventuali transizioni in modo concettualmente istantaneo.
Nella realizzazione SW un automa sincrono è costituito da una procedura ed il *clock* è rappresentato dall'invocazione periodica di tale procedura.
Gli **ingressi**, come si conviene ad un sistema *time-driven,* sono dei **valori di stato** numerici, ordinali o booleani. I valori che non siano già di tipo booleano vengono trasformati in booleano mediante operatori di confronto.
Es.

Ingressi esterni

chiave inserita Y/N	booleano
LIV (livello carburante)	numerico
(LIV > SOGLIA)	booleano
modo [manuale, automatico, manutenzione]	enumerativo
(modo = manuale)	booleano

I **simboli di ingresso** all'automa sono associati ad altrettante **funzioni combinatorie** dei valori booleani ricavati dagli ingressi.
Es.

Simboli	**Funzioni**
IS1	(chiave inserita) and (LIV > SOGLIA)
IS2	not (LIV > SOGLIA) or (modo = manutenzione)

Naturalmente alcune configurazioni di valori degli ingressi possono portare ad avere più di un simbolo di valore TRUE. Il non determinismo che ne può derivare concettualmente, viene in genere risolto implicitamente dall'ordine con cui vengono valutate le funzioni che producono i simboli di ingresso per decidere la transizione da effettuare. Si noti comunque che in ogni stato potranno essere significativi, ed associati ad altrettante transizioni uscenti dallo stato, eventualmente solo alcuni dei simboli (funzioni combinatorie), e il fatto che alcuni di tali simboli non siano mutuamente esclusivi è indicativo di una descrizione di comportamento ambigua.
Questo approccio si presta ad una realizzazione basata su un temporizzatore ciclico con la primitiva `WAIT(TIMER)` e con l'acquisizione degli ingressi effettuata mediante *polling*.

La struttura del motore dell'automa **sincrono** (*time driven*) può essere la seguente.

```
for (;;)
{
     WAIT (TIMER)
     leggi_stati_ingressi();
     calcola_simboli();       // funz. combinatorie
     esegui_transizione();
     loop_function(S);
}
```

L'esecuzione di azioni di transizione può essere effettuata dalla procedura `esegui_transizione()`, mentre le attività associate allo stato sono eseguite da una `loop_function(S)` associata ad ogni stato.

Questa struttura si presta alla gestione con un **unico processo** anche di un **array di automi** sincroni con lo stesso periodo di attivazione, secondo lo schema seguente in cui l'indice di automa in esame (i) è usato per specializzare le procedure `calcola_simboli()` e `esegui_transizione()` che dovranno utilizzare, ovviamente, i descrittori associati all'automa i-esimo.

```
for (;;)
{
     WAIT (TIMER);
     leggi_stati_ingressi();
     for (i=0; i< N_automi; i++)
          {calcola_simboli (i);
          esegui_transizione (i);
          loop_function(S[i]);}
}
```

Naturalmente ogni automa avrà il suo stato S[i] e una sua tabella di transizioni.

11.1.2 AUTOMI ASINCRONI (Event Driven)

I simboli di ingresso sono **eventi**, eventualmente accodati in una coda FIFO.
L'esecuzione dell'automa è sincronizzata, eventualmente in modo lasco tramite la coda FIFO, con gli eventi di ingresso.
Eventuali simultaneità sono risolte dall'accodamento.
Se gli eventi sono esterni la soluzione tipica è di recepirli mediante interrupt.

Con l'approccio asincrono la funzionalità dell'automa è attivata dagli stessi eventi che sono rappresentati con i simboli di ingresso e che l'automa utilizza come consumatore. Naturalmente la corretta gestione del consumo di eventi richiede un loro accodamento, dato che in generale si tratta di eventi provenienti da flussi mutuamente aperiodici, e dato che le operazioni associate alle transizioni impiegano un tempo finito per essere compiute, durante il quale possono verificarsi eventi che non si devono perdere.
La struttura di accodamento degli eventi in ingresso funge anche da meccanismo di sincronizzazione dell'attività dell'automa con gli eventi stessi e risolve implicitamente eventuali contemporaneità tra eventi. Sono adatte a questo scopo strutture di tipo **mailbox** o di tipo **buffer circolare**, che sono spesso offerte dai nuclei *multi-task* corredate delle relative primitive di accesso e sincronizzazione.

Per gli eventi esterni il meccanismo tipico di percezione è basato sulla generazione di richiesta di interruzione la cui routine di servizio funge da produttore dell'evento interno corrispondente, ad esempio con l'uso di una primitiva

```
PUT (SYMBOL, BUFFER),
```

che accoda nel buffer il simbolo identificatore dell'evento. Gli eventi interni sono invece direttamente prodotti dai processi, con l'uso della stessa primitiva.

La struttura del motore dell'automa **asincrono** (*event driven*) è sinteticamente la seguente.

```
for (;;)
{
     symbol = GET (BUFFER);         //attesa simbolo
     esegui_transizione (symbol);
}
```

Questa struttura richiede l'assegnazione di un processo ad ogni automa, dato che la primitiva `GET (BUFFER)` è bloccante se non ci sono simboli in coda, Inoltre questa semplice struttura si presta all'esecuzione di azioni associate alle transizioni (nell'ambito della funzione `esegui_transizione (symbol)`) ma non di attività durante la permanenza negli stati.

Una versione che prevede anche un'attività associata agli stati (esecuzione ciclica di una *loop-function*) con l'approccio asincrono, può essere la seguente, basata sull'invocazione di una primitiva

```
GET (BUFFER, timeout)
```

che ritorna anche per *time-out* se entro la scadenza non riceve alcun simbolo..

```
stato = STATO_INIZIALE;
for (;;)
{
     do
     {
          loop_function (stato);
          symbol = GET (BUFFER, timeout);
     } while (symbol == T_OUT);
     esegui_transizione (symbol);
}
```

L'attività di stato è realizzata dalla periodica invocazione della `loop_function()` garantita almeno ogni `timeout` unità di tempo, anche in assenza di eventi.

11.1.3 IMPLEMENTAZIONE A STRUTTURA DI PROGRAMMA

Un'implementazione che mappa il comportamento dell'automa sulla **struttura** del programma assume in genere una forma come quella esemplificata nel seguito.

```
stato = STATO_INIZIALE;
for (;;)
switch (stato)
{
// gestione dello stato 0
     case S0:
```

```
          loop_function (0);
          symbol = getsymbol (0);
          switch (symbol)
          {
               // gestione delle transizioni
               ................
               case Ik:
                    azione_transizione (n);
                    stato = nuovo_stato;
                    break;
               ................
          }
          break;
// gestione dello stato 1
     case S1:
     ............
}
```

Questa tecnica implementativa presenta le seguenti caratteristiche.

Facilità di realizzazioni di comportamenti **non regolari**, con eccezioni, stati ricchi e poveri di attività o di transizioni, ecc.

Facilità di debug selettivo di particolari comportamenti, in particolare con inserimento di punti di arresto (*break point*) nelle porzioni relative del programma.

Difficoltà di modifica e in generale manutenzione, soprattutto se il programmatore sfrutta i gradi di libertà in modo irregolare e disomogeneo.

Maggiore occupazione di memoria con numerose ripetizioni di porzioni di codice eguali o simili.

11.1.4 IMPLEMENTAZIONE A STRUTTURA DI DATI

La tecnica implementativa che basa su strutture di dati la descrizione del comportamento di un automa, cioè del suo grafo delle transizioni, è decisamente più compatta, regolare e strutturata rispetto a quella vista al punto precedente, ed è quindi quella **generalmente adottata**.
La struttura dati di base è costituita da un **array di stati**. Ogni elemento di questo array contiene indicazioni su:

azione da eseguire in ingresso allo stato
attività ciclica da eseguire durante lo stato (*loop-function*)
azione da eseguire in uscita dallo stato
descrizione delle transizioni uscenti dallo stato.

Per la descrizione delle transizioni si possono adottare soluzioni diverse, in dipendenza del grado di regolarità dell'automa. Ad esempio le seguenti soluzioni.

Automi regolari.

Sono automi per cui in ogni stato sono significativi (quasi) tutti i simboli in ingresso.
Le transizioni sono descritte associando ad ogni stato un **array** con tanti elementi (transizione) **quanti sono tutti i simboli di ingresso** (indice). Ad ognuno dei possibili simboli di ingresso viene associata una transizione.

NOTA - In molti casi può essere significativo distinguere tra i simboli che non provocano transizioni e quelli che provocano transizioni verso lo stato attuale (autoanelli). La distinzione è sostanziale se si adotta la possibilità di eseguire azioni all'ingresso e all'uscita degli stati: queste azioni dovranno essere eseguite in caso di **transizione autoanello** e non eseguite in caso di **non transizione**.

Con questa struttura si ricade nella tipica tabella bidimensionale **stati - ingressi,** con tante righe quanti sono gli stati e tante colonne quanti sono i simboli di ingresso.
I pregi principali di questa soluzione sono la regolarità della struttura e la facilità di verifica della completezza, dato che si devono riempire tutte le caselle in numero predeterminato. Esiste talora il rischio di occupazione di memoria in quantità inutilmente elevata se l'automa è piuttosto irregolare, cioè con molti simboli che, in determinati stati, non provocano transizioni.

Automi irregolari.

Si tratta di automi per i quali gli ingressi significativi sono molto diversi da stato a stato e spesso per il singolo stato costituiscono un ridotto sottoinsieme di tutti gli ingressi.
Ad ogni stato si associa una **lista che contiene solo le transizioni effettive** ed eventualmente solo quegli autoanelli che intenzionalmente rappresentino l'uscita e il rientro nello stesso stato con esecuzione delle corrispondenti azioni di ingresso e uscita.

In entrambi i casi (array o liste) ogni **transizione** è costituita da un **record con i seguenti campi**.

Valore del simbolo di ingresso. (o funzione combinatoria se automa sincrono)
Prossimo stato
Puntatore a funzione che esegue l'azione di (Mealy) transizione.

Si noti che la struttura dati che descrive il comportamento può essere vista come un particolare linguaggio di programmazione, di cui il motore dell'automa è l'interprete.

11.1.4.1 Esempio di Automa asincrono con time-out

Struttura dati

```
// --- descrittore di transizione ---
typedef struct desc_trans
{
      char       symbol;
      int    s_prossimo;
      void       (*f_trans)();
      struct desc_trans  *next;    // punt. elem. successivo lista
} TRANS;

// --- descrittore di stato ---
typedef struct desc_stato
{
      void       (*f_in)();     // funzione azione di ingresso
      void       (*f_ciclica)();   // funzione attivita' ciclica
      void       (*f_out)();    // funzione azione di uscita
      TRANS      *transiz;          // punt. lista transizioni uscenti
} STATO;
```

Motore

```
/*********************************************************************
Processo di esecuzione di automa asincrono
Riceve nella coda in_buf i simboli rappresentanti gli eventi in
ingresso, mediante la primitiva GET() con time-out.
GET (B, TO) riporta il primo simbolo dalla coda B
      se non riceve simboli entro il tempo TO riporta il simbolo
      particolare T_OUT
*********************************************************************/

STATO d_automa [N_STATI]; // array descrittore dell'automa

void automa (void)
{
int   st_corr;      // stato corrente
int   symbol_in;    // simbolo in ingresso
TRANS *p_trans;     // puntatore di servizio a transizioni

// ----- inizializzazioni -----
      stato = STATO_INIZIALE;

// ----- ciclo permanente -----
      for (;;)
      {
          do     // --- ciclo ripetuto in attesa di simboli ---
          {
              d_automa[st_corr].(*f_ciclica)();
              symbol_in = GET (in_buf, TIMEOUT);
          } while (symbol_in == T_OUT);

          // ----- qui automa ha ricevuto un simbolo -----
          p_trans = d_automa[st_corr].transiz;   //testa lista
          while ((p_trans != NULL)&&(p_trans->symbol != symbol_in))
              //cerca simbolo in lista di transizioni uscenti da stato
              p_trans = p_trans->next;

          if (p_trans != NULL)
          {   // --- trovato simbolo - esegue transizione ---
              d_automa[st_corr].(*f_out)();   //azione uscita
              p_trans->(*f_trans)();          //azione transiz
              st_corr = p_trans->s_prossimo;  //nuovo stato
              d_automa[st_corr].(*f_in)();    //azione ingresso
          }
      }
}
```

11.2 SISTEMI A REGOLE

I sistemi a regole (RBS = *Rule Based Systems*) presentano un'interessante forma di rappresentazione del comportamento del sistema, e possono essere considerati per certi aspetti un'estensione degli automi a stati.

Le regole sono costrutti della forma:

se <premessa> allora <conclusione>.

premessa è una proposizione che può essere vera o falsa, e rappresenta una condizione;
conclusione può essere un fatto dedotto o un'azione da compiere.
Negli automi la rappresentazione potrebbe essere basata su regole tutte della particolare forma:

se [(stato=Si) and (ingresso=Ik)] allora [esegui transizione Tn]

Nei sistemi a regole in generale, sia le premesse che le conclusioni possono assumere forme diverse e varie.

Lo scenario che consideriamo in un sistema a regole è composto da:

- Un insieme di fatti primari costituiti da stati ed eventi del mondo esterno
- Un insieme di fatti derivati volatili che costituiscono derivazioni dei fatti primari al solo scopo di estrarne informazioni più comode.
- Un insieme di fatti derivati persistenti (Base di conoscenza di fatti) che costituiscono una rappresentazione storica, analoga agli stati degli automi.
- Un insieme di regole (Base di conoscenza di procedimenti).
- Un insieme di procedure che implementano le conclusioni.
- Un motore inferenziale costituito da un processo software che si fa carico di rendere operative le regole.

Si noti che il comportamento di un sistema a regole è tipicamente **combinatorio**. Per ottenere il comportamento **sequenziale** richiesto in molte applicazioni, occorre introdurre un insieme di "fatti derivati **persistenti**", come sopra accennato, che sopravvivano ad ogni singola "tornata" del motore inferenziale, in modo da propagare alla tornata successiva la memoria storica dell'evoluzione passata e influenzare di conseguenza il comportamento in corso.

Per la modellazione di sistemi reattivi ci interessano i motori inferenziali che realizzano il comportamento detto concatenamento progressivo (*forward chaining*) che consiste nella seguente attività ciclica:

acquisire i fatti in ingresso
valutare le premesse delle regole
per ogni premessa vera eseguire la procedura associata alla conclusione

La base di regole può essere costruita in diversi modi.

1. Si genera l'elenco delle regole mediante **interviste ad esperti** del comportamento reattivo voluto. Durante il funzionamento la base di regole sarà statica
2. Si prevedono **meccanismi di apprendimento** che partendo da zero o da una base di regole iniziale, producono variazioni nell'insieme di regole, cercando di ottimizzare una opportuna funzione di valutazione della bontà del comportamento.

I sistemi a regole trovano interessanti applicazioni anche per l'autodiagnosi dei sistemi di automazione e nella diagnostica in generale. In queste applicazioni le azioni derivanti dall'applicazione delle regole sono costituite da messaggi diagnostici all'operatore o da comandi per saggiare specifiche funzionalità da diagnosticare.

Problemi tipici dei sistemi a regole

Consistenza delle regole. Evitare che vi siano regole contradditorie, cioè con premesse che possono verificarsi contemporaneamente e con conclusioni incompatibili.

Completezza delle regole. Tutte le combinazioni di fatti (In ingresso e derivati) devono verificare almeno una premessa, in modo che il sistema sappia sempre cosa fare. In alcuni casi possono essere accettabili delle azioni di *default*.

Produzione dei fatti derivati. Il concatenamento progressivo (*forward*) deve portare ad un insieme di fatti stabile con un numero limitato di iterazioni del motore inferenziale, o alternativamente le regole vanno ordinate in modo da minimizzare i riferimenti in avanti (v. ritardi nelle azioni cicliche par. 10.3.1).

11.3 TAVOLE DI DECISIONE

Le tavole (o tabelle) di decisione sono un modo particolare di rappresentare delle regole che ne facilita la verifica di consistenza e completezza.
Gli antecedenti delle regole sono espressi come congiunzione (*and*) di condizioni.
Ogni condizione ha la forma

soggetto = valore

che in termini inglesi suona:

stub = entry

I valori sono generalmente TRUE o FALSE, ma per alcuni soggetti si possono avere diversi valori enumerativi.
Le singole condizioni risultano vere se il loro soggetto assume il valore citato.
Ogni soggetto (*stub*) è corredato di una funzione di valutazione che ne riporta il valore (*entry*).
Ad esempio il soggetto **livello_carburante** potrà assumere i valori [**pieno, medio, riserva, vuoto**]

Per garantire la completezza proponiamoci inizialmente di rappresentare tutti i possibili antecedenti di regole, cioè tutte le combinazioni di valori di tutti i soggetti condizione **Ci**, che costituiscono gli ingressi del sistema a tabella.
Ogni antecedente può essere associato ad una colonna di una tabella, le cui righe **superiori** corrispondono ai soggetti **condizione di ingresso**.
Il numero di colonne sarà dato dalla produttoria delle cardinalità dei valori dei soggetti, in modo da avere tutte le possibili combinazioni.

Ad es. consideriamo i soggetti

C1 - livello_carburante cardinalità 4 (Pieno, Medio, Riserva, Vuoto)
C2 - pressione_olio cardinalità 2 (Normale, Bassa)

Soggetto	Cardinal.	COLONNE DEGLI ANTECEDENTI							
C1	4	P	M	R	V	P	M	R	V
C2	2	N	N	N	N	B	B	B	B

Naturalmente possono esservi condizioni di indifferenza, per cui il numero totale di colonne si riduce anche notevolmente rispetto alle combinazioni totali.

La porzione **inferiore** della tabella è invece dedicata alle **azioni** e sarà composta di tante righe quanti sono i soggetti di azione previsti, e dello stesso numero di colonne della parte condizione.
Gli elementi di questa parte della tabella sono i valori associati ai soggetti azione, cioè i valori associati alle uscite.
Il caso più semplice è quello binario (azione da eseguire o no), ma si possono avere azioni con cardinalità maggiore.
Es. soggetti azione

A1 - fare_rifornimento cardinalità 3 Si, Urgente, No
A2 - fermare motore cardinalità 2 Si, No

Una **tabella completa** assume quindi la forma seguente.

Soggetto condizione	Cardinalità	REGOLE							
C1	4	P	M	R	V	P	M	R	V
C2	2	N	N	N	N	B	B	B	B
				↓					
Soggetto azione									
A1	3	N	N	U	S	N	N	U	S
A2	2	N	N	N	N	S	S	S	S

Il meccanismo operativo di gestione della tabella è il seguente.

1. Si acquisiscono i valori degli ingressi Ci.
2. Si valutano i corrispondenti valori di condizione
3. Si individua la colonna corrispondente alla regola attivata
4. Si attivano le uscite con i valori del vettore colonna della regola.

Ad esempio nella tabella è evidenziata in neretto la situazione in cui valgono le condizioni C1 = R (livello Riserva) e C2 = N (olio Normale) che individuano la colonna n.3, che nella parte inferiore della tabella riporta le azioni da eseguire A1 = U (rifornimento urgente) e A2 = N (fermare motore No).
Si noti la possibilità di adottare una memoria EPROM per realizzare a livello HW la funzionalità della tabella.

Si noti che il comportamento descritto da tabelle di decisione è tipicamente **combinatorio**. Il comportamento **sequenziale**, necessario in molti casi, si ottiene quindi dedicando alcune uscite e alcuni ingressi a variabili di stato nella classica **retroazione**.

11.4 SISTEMI A REGOLE FUZZY

La logica fuzzy è stata introdotta da Lofti Zadeh alla fine degli anni sessanta per superare la rigidità della logica classica nel descrivere le modalità di ragionamento della mente umana.
Questo proposito prende le mosse dalla considerazione che la separazione netta tra verità e falsità, tipica della logica classica e del mondo dei calcolatori digitali, raramente rispecchia l'impostazione umana di molti problemi, anche ingegneristici. Molte applicazioni ingegneristiche hanno infatti a che fare con fenomeni continui i cui modelli non sono noti con precisione assoluta e le cui misure sono affette da incertezze. In sintesi i concetti di **incertezza** e di **tolleranza** sono fondamentali in molte situazioni.

Molte **classificazioni**, che costituiscono una utile riduzione di cardinalità di insiemi (discreti o continui), sono basate su una **discretizzazione**, cioè su una corrispondenza (generalmente univoca) di sottoinsiemi di valori con un numero ridotto di valori enumerativi (classi o categorie). Ad esempio possiamo **semplificare l'insieme delle velocità** di un automobile digitalizzandole in **basse**, **medie** e **alte**.
Se utilizziamo in modo classico un calcolatore per fare ciò, introduciamo 2 soglie, ad esempio rispettivamente di 10 m/s (=36 Km/h) e di 50 m/s (=180 Km/h), così da suddividere il campo di valori della grandezza che ci interessa in tre fasce, ad una sola delle quali apparterrà ogni possibile velocità. E' immediato accorgersi che l'esito prodotto da un normale algoritmo di digitalizzazione non sarà generalmente soddisfacente rispetto alle nostre aspettative, dato che assegnerà la qualifica di **velocità media** con pari titolo a velocità di 40 Km/h e velocità di 178 Km/h.
Così Zadeh ha avuto l'idea di introdurre un **grado di appartenenza** di un valore numerico a **diverse** categorie, catturando così il vero significato di molte espressioni imprecise del linguaggio naturale. Questo particolare modo di effettuare una discretizzazione è detto **granulazione**.
I campi di applicabilità della logica fuzzy sono tipicamente i sistemi basati sulla conoscenza (KBS = *Knowledge Based Systems*) e i sistemi di controllo.

11.4.1 LA LOGICA FUZZY

La logica **fuzzy** (= sfumata) tratta insiemi, detti *fuzzy set*, per i quali il valore di appartenenza di un elemento non è necessariamente o *vero* o *falso* come per i sistemi *crisp* (precisi), ma può assumere tutta una gamma di valori intermedi.
Differenze rispetto alla logica classica

Verità

I valori di verità linguistica (proprio vero, abbastanza vero, non molto vero, ecc.) sono etichette di sottoinsiemi fuzzy dell'intervallo 0...1, dove gli estremi rappresentano rispettivamente 0 = falso, 1 = vero.

Predicati

Sono significativi anche predicati sfumati, come: *veloce*, *molto maggiore*, *efficace*, ecc.

Modificatori dei predicati

Nella logica classica l'unico modificatore dei predicati è la negazione *not*. Nella logica fuzzy sono invece possibili numerosi modificatori di predicati, come: *molto, abbastanza, più o meno*, ecc. Questi modificatori costituiscono la base per generare i valori di variabili linguistiche.

Quantificatori

Oltre ai quantificatori esistenziale (∃) e universale (∀) della logica classica, sono possibili altri quantificatori come: *pochi*, *molti*, *quasi tutti*, *generalmente*, ecc.

Analogamente alla logica classica, la logica fuzzy si propone di fornire le regole che consentono di calcolare il valore di verità di un enunciato composito, a partire dai valori di verità delle sue componenti. Come c'è da attendersi, con questo approccio anche il valore di verità ottenuto sarà sfumato.

Per consentire ciò la logica fuzzy propone delle tecniche per svolgere i seguenti passi.

Fuzzificazione.

Regole che consentono di valutare il valore di verità associato all'attribuzione di un valore ad ognuna delle categorie previste. Cioè, ad es. valutare quanto è vero che Paolo, alto m 1,75, appartiene alle categorie *statura alta, statura media, statura bassa*.

Composizione.

Regole di composizione dei valori di verità sfumati, mediante i classici operatori logici (and, or, not) per valutare espressioni logiche.

Defuzzificazione.

Regole da utilizzare se alla fine ci interessa ricavare dei valori numerici.

Diversamente dalla logica booleana, le regole di fuzzificazione, composizione e defuzzificazione non sono uniche ma, coerentemente con l'approccio fuzzy, si prestano a varianti.

11.4.2 VARIABILI E FUNZIONI DI APPARTENENZA

Una **variabile numerica** (classica) assume valori numerici. Es. V = 50.
Una **variabile linguistica** è una variabile che può assumere valori linguistici, cioè parole, come velocità elevata, alto, basso, ecc.
I valori linguistici costituiscono etichette (*label*) di altrettanti insiemi fuzzy associati alla variabile.
Ogni insieme fuzzy è definito da una **funzione di appartenenza** (MF = *membership function*) che determina il **grado di appartenenza** (compreso tra 0 e 1) di ogni valore numerico all'insieme stesso.

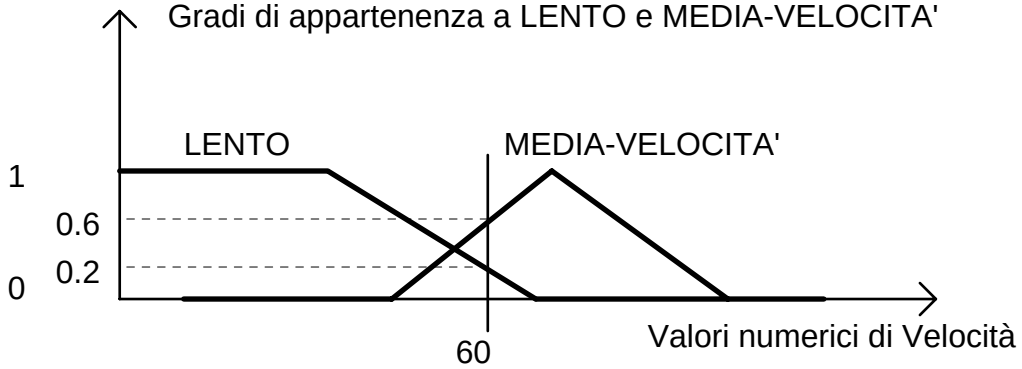

Fig. 11.1 - Funzioni MF per insiemi fuzzy LENTO e MEDIA-VELOCITA' e gradi di appartenenza ad essi della velocità 60.

Una variabile linguistica assume contemporaneamente, e ognuno con un suo grado, diversi valori linguistici.
Ad esempio con gli insiemi rappresentati nella figura 11.1, una velocità numericamente pari a 60 assume con grado 0.2 il valore linguistico LENTO e con grado 0.6 il valore linguistico MEDIA-VELOCITA'.

L'operazione di fuzzificazione opera quindi la trasformazione del valore numerico di una variabile in un insieme di valori linguistici ognuno associato ad un grado di appartenenza di valore compreso tra 0 e 1.
Le funzioni di appartenenza più utilizzate sono quelle triangolari e trapezoidali, e sono collocate in modo da avere sovrapposizioni.
Come casi particolari si possono esprimere anche appartenenze *crisp*, mediante funzioni di appartenenza rettangolari o a *singleton*.
Nella figura 11.2 sono esemplificati i casi di insiemi fuzzy "degeneri", che prevedono solo i gradi di appartenenza 1 (*vero)* e 0 (*falso).*

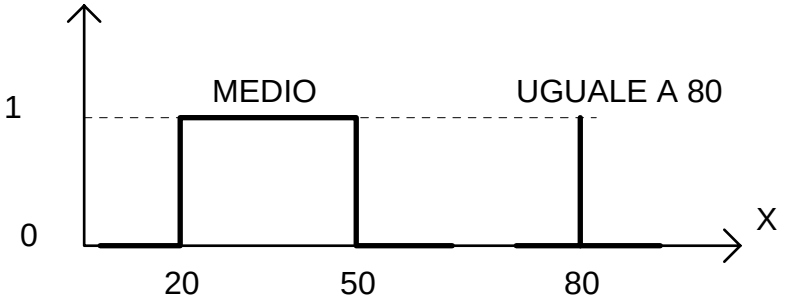

Fig. 11.2 - Funzioni MF per casi particolari di insiemi fuzzy.

Nella figura l'insieme MEDIO comprende i valori dell'intervallo $20 < X < 50$, mentre UGUALE_A_80 è un insieme *singleton* a cui appartiene solo il valore 80.

11.4.3 OPERATORI LOGICI E ARITMETICI

Nella figura 11.1 la proposizione

(V=60) è una media velocità

ha valore di verità pari a 0.6 e questo è anche il grado di appartenenza del valore V=60 all'insieme MEDIA-VELOCITA'.
Il calcolo dei valori di verità di **proposizioni composte con i connettivi logici** si basa sulle seguenti regole.

Siano P, P1 e P2 delle proposizioni
val (P) grado di verità della proposizione P.

operatore	**proposizione**		**regola di composizione**
AND	se $P = P1 \wedge P2$	allora	val(P) = min(val(P1), val(P2))
OR	se $P = P1 \vee P2$	allora	val(P) = max(val(P1), val(P2))
NOT	val($\neg$P)	=	1-val(P)

11.4.4 REGOLE FUZZY

Un sistema basato sulla conoscenza prevede i seguenti elementi.

- **Fatti** - Espressi con asserzioni o con valori numerici di variabili.
- **Base di Regole** - Relazioni del tipo
  IF <antecedente> THEN <conseguente>
- **Conclusioni** - Asserzioni risultanti dal processo inferenziale.

Viene detto **motore inferenziale** il processo che individua le regole i cui antecedenti sono resi veri dai fatti e aggiunge alle conclusioni i relativi conseguenti. In generale le

conclusioni costituiscono fatti aggiuntivi che a loro volta, col meccanismo detto di *forward chaining*, possono attivare altre regole, e così via, fino a quando l'inferenza non produce più fatti nuovi.
Nei sistemi a regole fuzzy la forma tipica dell'antecedente è la congiunzione (AND) di variabili linguistiche.
Il grado di verità risultante per l'antecedente viene attribuito al conseguente.
Esempio. - Una delle regole che descrivono l'intensità di frenata richiesta per un veicolo in base a velocità e distanza da un ostacolo potrebbe essere:

> **IF (velocità è media) $\wedge$ (distanza è breve) THEN (frenata è moderata)**
> Se (velocità è media) ha grado 0.8 e (distanza è breve) ha grado 0.5, il grado complessivo dell'antecedente è 0.5 (si ricordi la regola dell'operatore *and*). In tal caso si assegna un grado di appartenenza di 0.5 di frenata all'insieme fuzzy moderata.

E' interessante notare che le conclusioni possono essere non solo **nuovi fatti** semplicemente da aggiungere alla base dei fatti, ma anche **azioni** da eseguire.
Con un sistema a regole è quindi possibile descrivere anche un **comportamento reattivo** che agli stimoli in ingresso (fatti) reagisce con l'esecuzione di azioni.
Un aspetto interessante è che un tale comportamento deriva da regole che sono molto simili, grazie alla logica fuzzy, a quelle che potrebbe fornire un esperto per spiegare le modalità di comportamento volute.

11.4.5 DEFUZZIFICAZIONE

I valori fuzzy ottenuti dall'applicazione delle regole, come accennato sopra, sono costituiti da un insieme di variabili linguistiche e relativi gradi. Se tali valori sono destinati ad essere forniti come risultati numerici, devono subire l'operazione detta di **defuzzificazione** che dai valori linguistici di una variabile (linguistica) calcola il valore numerico. Sono state proposte diverse tecniche di defuzzificazione, ma qui presentiamo solo quella detta del **baricentro**.
Osserviamo innanzitutto che le regole della base di conoscenza vanno considerate implicitamente in OR tra loro, e che una stessa variabile linguistica può apparire nei conseguenti di più regole, con lo stesso o con diversi valori linguistici, i cui insiemi di appartenenza col relativo grado vanno quindi composti con l'operazione di unione OR.
Nella fig. 11.3 è riportato un esempio in cui da una regola risulta che la variabile Frenata assume valore linguistico MEDIA con grado 0.6, e da un'altra regola assume valore linguistico BRUSCA con grado 0.2.
Il baricentro dell'area di unione corrisponde al valore numerico da attribuire alla variabile Frenata ottenuto per defuzzificazione.

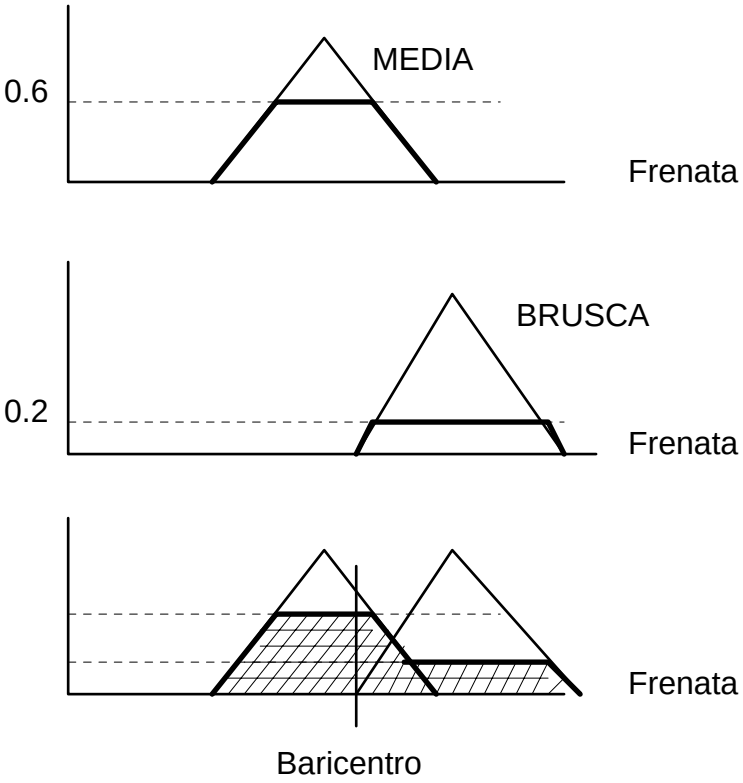

Fig. 11.3 - Defuzzificazione con la tecnica del baricentro.

Con i sistemi fuzzy si possono realizzare:

- Comportamenti descritti da regole "umane"
- Funzioni non lineari
- Funzioni a risultato numerico con ingressi numerici ed enumerativi
- Funzioni con risultato enumerativo (decisori)

Per ogni ingresso si decidono gli insiemi fuzzy. Spesso 3, 5 o 7
Ad es.
NL (*Negative Large*), NM (*neg. medium*), NS (*Neg. Small*), ZR (*Zero*), PS, PM, PL
E' opportuno, per completezza, che si abbiano sovrapposizioni tra gli insiemi contigui.

Ogni insieme è definito da 3 parametri se si adottano le funzioni MF triangolari.
La scelta dei valori dei parametri costituisce uno dei punti lasciati a tecniche empiriche, a meccanismi di apprendimento o ad algoritmi di ricerca di ottimo.

Si può rappresentare con una matrice bidimensionale il caso di due ingressi e un'uscita.
Ogni colonna corrisponde ad un valore linguistico di una variabile di ingresso.
Ogni riga corrisponde ad una valore dell'altra variabile di ingresso.
Gli elementi riportano i valori linguistici della variabile di uscita.

Esempio.
Un controllore fuzzy riceva in ingresso una variabile pari all'errore (misura - setpoint) e una che rappresenta la derivata dell'errore, e fornisca in uscita l'incremento da dare alla variabile manipolata. Si suppone di adottare per le variabili in ingresso e per quella in uscita una classificazione a 5 valori linguistici (NL, NS, ZR, PS, PL). Una possibile descrizione delle regole può essere quella rappresentata dalla tabella seguente.

		ERR				
		NL	NS	ZR	PS	PL
dE/dt	NL	**PL**	**PL**	**PS**	**PS**	**PS**
	NS	**PL**	**PS**	**PS**	**ZR**	**ZR**
	ZR	**PS**	**PS**	**ZR**	**NS**	**NS**
	PS	**ZR**	**ZR**	**NS**	**NS**	**NL**
	PL	**ZR**	**NS**	**NS**	**NL**	**NL**

Nei casi più generali si può adottare una rappresentazione come quella vista precedentemente per le tabelle di decisione, salvo naturalmente tenere conto che le regole non sono mutuamente esclusive nell'attivazione ma che anzi ogni colonna (regola) può venire attivata con un suo grado di attivazione che propaga alle azioni di uscita della parte inferiore della tabella.

11.4.6 CONTROLLORI FUZZY

Una classe di interessanti applicazioni dei sistemi a regole fuzzy è costituita dai controllori.
I controllori fuzzy possono prevedere come variabili di ingresso:

- set point
- valore della misura della grandezza controllata
- valori di misure di altre grandezze
- derivate di grandezze
- integrali di errori
- modi di funzionamento

Si noti che si può avere una grande varietà nel numero e nel tipo delle variabili in ingresso. Ciò conferisce potenzialità molto interessanti ai controllori fuzzy, che però possono comportare la necessità di un elevato numero di funzioni di appartenenza e di regole.
Un insieme di regole determina il comportamento del controllore. Queste regole vedono negli antecedenti congiunzioni di valori linguistici degli ingressi e nei conseguenti valori linguistici di una o più uscite.
Le uscite (una o più) sono generalmente valori numerici di comando di attuatori, e quindi sono ottenute mediante una defuzzificazione.

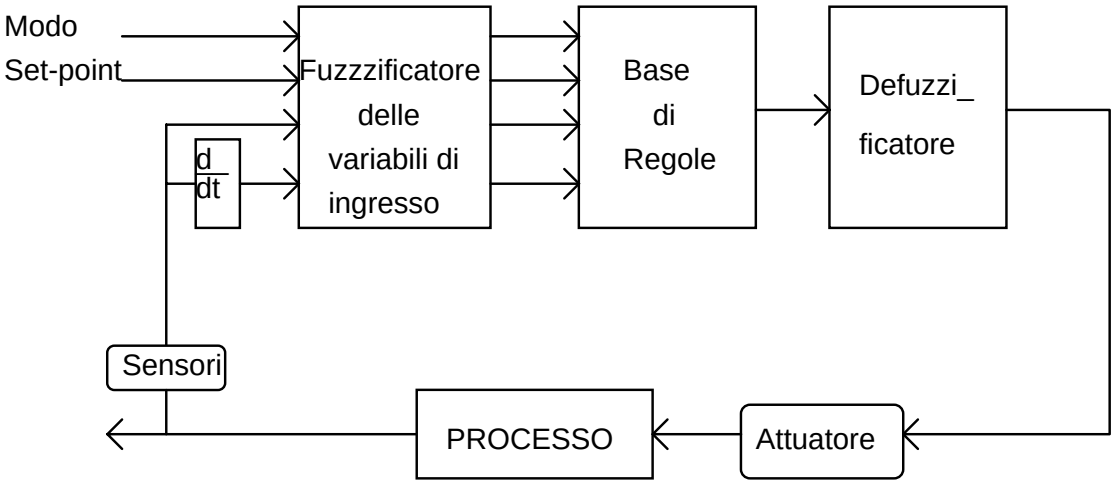

Fig. 11.4 - Schema di controllore fuzzy.

I controllori fuzzy sono da considerare complementari e non sostitutivi dei controllori classici, ed assumono particolare interesse quando il sistema da controllare è non lineare e noto solo approssimativamente, e si abbiano **diverse variabili in ingresso**.

> Si noti a questo proposito che ad un controllore fuzzy non si fornirà in ingresso il valore dell'errore, ma la **misura** e il ***set-point*** costituiranno due variabili distinte, consentendo così, ad esempio, di trattare diversamente errori della stessa entità ma in corrispondenza di piccoli o grandi valori della misura (comportamento non lineare).

Possono essere realizzate interessanti forme di **simbiosi** tra controllori classici e sistemi fuzzy (SF).

- SF produce i valori dei parametri di un PID
- SF è posto come filtro a valle del controllore classico
- SF in alternativa ad un controllore classico, comandata da un decisore in base alla situazione.

Un controllore fuzzy adattativo, o meglio ad apprendimento, può essere realizzato mediante un meccanismo di modifica della base di regole. In questo caso è necessario disporre di una funzione di valutazione della bontà del comportamento, che guidi l'algoritmo di modifica delle regole e delle funzioni di appartenenza MF.

11.4.7 PROCESSORI FUZZY

Le operazioni di fuzzificazione, valutazione delle regole e defuzzificazione si prestano ad una esecuzione SW. I tempi di calcolo su un processore *general-purpose* non sono però brevi come quelli che si possono ottenere con processori appositi.
Sono presenti in commercio interessanti coprocessori specializzati per svolgere le operazioni tipiche di una elaborazione fuzzy basata su regole.
Tipici parametri di valutazione di questi processori sono i seguenti.

- Numero massimo di regole su cui può operare l'inferenza.
- Numero massimo di termini degli antecedenti e dei conseguenti.
- Numero massimo di valori linguistici gestibili.
- Forme e parametri delle funzioni di appartenenza (MF).
- Metodi di defuzzificazione.

Un esempio è costituito dal processore Omron FP-3000.
Questo coprocessore si appoggia ad un normale processore per le operazioni di:

- caricamento della base di regole
- acquisizione dei valori delle variabili
- sincronizzazioni con il mondo esterno
- emissione dei comandi di controllo.

Particolarmente interessante è l'interfacciamento HW con il processore *host*, che vede FP-3000 come se fosse una normale memoria RAM, le cui celle costituiscono dei registri in cui scrivere regole, parametri delle MF e valori delle variabili, e registri da cui leggere i valori dei risultati.
Alcune caratteristiche di FP-3000 sono:

8 ingressi e 4 uscite
regole con 8 antecedenti e 2 conseguenti
128 regole
max 7 label per ogni ingresso con MF a 4 vertici (trapezi)
singleton e max 7 label per uscite
risoluzione numerica 12 bit

11.5 RETI NEURALI

Costituiscono un'interessante tecnica per realizzare funzioni di classificazione (ad es. riconoscimento di caratteri o forme in immagini), ispirata a modelli neurali biologici.
Si prestano a processi di apprendimento e trovano applicazione in problemi specialistici, in particolare legati alla robotica e all'Intelligenza Artificiale. Qui non vengono ulteriormente approfondite, ma si riportano alcuni riferimenti in bibliografia per i lettori interessati.

11.6 Esercizi

- Discutere il problema del non determinismo nel caso degli automi sincroni, così come presentati in 11.1.1.

- Discutere pregi e difetti dell'implementazione di automi a struttura di programma.

- Un ascensore debba servire 3 piani. Progettarne il controllore, realizzato a SW come un automa nelle versioni sincrona e asincrona.

- Descrivere con una tabella di decisione la gestione di un tergicristallo, avendo come ingressi la posizione della chiave (ON, OFF) e la posizione della levetta di comando (OFF, Lento, Veloce).

- Un ascensore debba servire 3 piani. Progettarne il controllore, realizzato a SW con una tabella di decisione. Si noti che essendo il comportamento della tabella di tipo combinatorio, si dovranno introdurre degli ingressi ulteriori in retroazione, costituiti da variabili di stato (fermo, in moto, porta aperta, prenotato, ecc.).

- Si progetti il controllo di un gruppo elettrogeno motorizzato, per alimentazione ausiliaria in caso di caduta della rete. Indicativamente le fasi di funzionamento siano:
fermo, motorino avviamento, transitorio di accelerazione, chiusura contatto alimentazione (a tensione e frequenza nominali), spegnimento. Si utilizzino temporizzatori per evitare l'accensione in caso di brevi cadute di rete e per evitare spegnimenti in caso di ritorni temporanei di rete.

11.7 BIBLIOGRAFIA

David Tweed
Designing Real-Time Embedded Software Using State-Machine Cocepts.
The Computer Application Journal - #53 - Dec. 1994
Semplice e applicativo.

Gill
Introduction to the Theory of Finite-State Machines.
McGraw-Hill

J. McCarthy
Decision Tables and Logic Processing.
Computer Language. Nov. 1986

Riccardo Scattolini, Nicola Schiavoni
Introduzione al controllo fuzzy.
Automazione e Strumentazione - Apr. 1993

Roberto Maietti, Roberto Marini
Sistemi di sviluppo per applicazioni fuzzy.
Automazione e Strumentazione - Apr. 1993

B. Fringuelli
Come ragionare in contesti imprecisi.
Automazione Oggi - Febb. 1995 - Ed. Jackson

Autori vari
Sistemi fuzzy nell'automazione e nel controllo dei processi industriali.
Giornata di studio ANIPLA - Milano 11-11-92

D.P. Kwok, P. Wang, C.K. Li
A Combined Fuzzy and Classical PID Controller.
Microprocessing and Microprogramming N.32 pag. 701
North Holland - 1991

L.A. Zadeh
Making Computers Think Like People.
IEEE Spectrum - Aug. 1984 pag.26

L.A. Zadeh
Knowledge Representation in Fuzzy Logic.
IEEE Transactions on Knowledge and Data Engineering - Vol.1 - March 1989 pag.89

D. Hammerstrom
Neural Networks at Work
IEEE Spectrum - June 1993

D. Hammerstrom

Working with Neural Networks.
IEEE Spectrum - July 1993

T. Khanna
Foundations of Neural Networks
Addison Wesley - 1990

J. Hertz, A. Kogh, R. Palmer
Introduction to the Theory of Neural Computation.
Addison Wesley - 1991