

# PROLOG E SISTEMI ESPERTI

---

- Prolog può essere utilizzato in almeno due modi differenti nella costruzione di sistemi esperti:
  - come semplice linguaggio di realizzazione.
  - sfruttando le caratteristiche del Prolog e definendo e costruendo dei sistemi integrati nel Prolog.
- Il Prolog può essere visto come un sistema a regole di produzione in cui:
  - clausole (regole) Prolog=regole di produzione;
  - asserzioni Prolog = fatti;
  - data-base Prolog = memoria di lungo termine + memoria di lavoro;
  - interprete Prolog=motore inferenziale (basato su una strategia backward, con ricerca in profondità con backtracking).

# USO DIRETTO DEL PROLOG

---

- USO DIRETTO DEL PROLOG NELLA COSTRUZIONE DI SISTEMI ESPERTI A REGOLE
- Un sistema a regole di produzione quale Prolog è solo il nucleo di un sistema esperto.
- Per ottenere tutte le funzionalità di un sistema esperto (quali la possibilità di effettuare ragionamento approssimato, la capacità di interazione con l'utente, la capacità di spiegazione) è necessario estendere tale nucleo di base.

# POSSIBILE FORMATO DELLE REGOLE

---

If precondizione P then conclusione C

If situazione S then azione A

If condizioni C1 and C2 sono vere  
then la condizione C è falsa.

- Le regole sono una forma abbastanza naturale per esprimere la conoscenza e godono delle seguenti proprietà:
  - Modularità;
  - Incrementalità;
  - Modificabilità;
  - Trasparenza, cioè capacità di spiegare il proprio comportamento:
    - "How", ovvero come sei arrivato a questa conclusione?
    - "Why", ovvero perchè sei interessato a questa informazione?

# ESEMPIO

---

- Si consideri una semplice regola di produzione (parte della base di conoscenza di un sistema per la diagnosi di guasti a una automobile):

if luci spente and

motorino di avviamento muto

then probabile (0.8) guasto alla batteria

- Il formato della regola è quello del sistema MYCIN. Il valore 0.8 associato alla regola rappresenta il “grado di certezza” con cui può essere raggiunto il conseguente se l’antecedente della regola è vero.

# ESEMPIO

---

- La regola può essere tradotta in una clausola Prolog:

```
batteria (guasta) :-  
    luci (spente) ,  
    motorino_avviamento (muto) .
```

- Il passo successivo è quello di definire dei meccanismi per il **ragionamento approssimato**.
- La verità di un fatto non viene più valutata utilizzando una logica a due valori {vero, falso}, ma piuttosto utilizzando una logica a più valori (anche infiniti).

# RAGIONAMENTO APPROSSIMATO (1)

---

- A ogni fatto  $F$  viene associato un **grado di evidenza** rappresentato mediante un numero reale nell'intervallo  $[0,1]$ :
  - se il grado di evidenza di  $F$  è 0, allora  $F$  è falso;
  - se il grado di evidenza di  $F$  è 1, allora  $F$  è vero;
  - valori intermedi corrispondono a casi di incertezza sulla falsità o verità di  $F$ .

- Grado di evidenza **EV** di un fatto **F**

**F with EV**

(in cui **with** è un operatore).

# RAGIONAMENTO APPROSSIMATO (2)

---

- Relazioni per trattare conoscenza approssimata:

**valuta\_antecedente (LISTA\_EV, EV)**

- data la lista **LISTA\_EV** dei gradi di evidenza delle condizioni nell'antecedente di una regola, **EV** è il grado di evidenza globale dell'antecedente della regola

**valuta\_consequente (EV\_ANT, CF, EV\_CONS)**

- data una regola con grado di certezza **CF** e il grado di evidenza **EV\_ANT** dell'antecedente, allora **EV\_CONS** è il grado di evidenza del conseguente

# ESEMPIO

---

- La regola può allora essere rappresentata mediante la seguente clausola Prolog:

```
batteria(guasta) with EV :-  
    luci(spente) with EV1,  
    motorino_avviamento(muto) with EV2,  
    valuta_antecedente([EV1, EV2], EV_ANT),  
    valuta_consequente(EV_ANT, 0.8, EV) .
```

- Può essere significativo ricercare tutte le soluzioni per un dato goal.



# MECCANISMI DI SPIEGAZIONE

---

- Una semplice spiegazione può essere ottenuta mediante argomenti aggiuntivi nelle asserzioni e nelle clausole.
- Più in particolare:
  - Ogni fatto **F** viene rappresentato mediante una asserzione del tipo
  - **F with EV explain EXPL.**
  - dove **explain** e **with** sono operatori e **EXPL** è la spiegazione associata al fatto **F**.
- Nella clausola corrispondente a una regola viene aggiunta una relazione per sintetizzare la spiegazione.

# MECCANISMI DI SPIEGAZIONE

---

- Ad esempio, la regola precedente può essere rappresentata mediante la clausola:

```
batteria(guasta) with EV explain
  dimostrato(batteria(guasta) with EV,
  a_partire_da([EXPL1,EXPL2])) :-
  luci(spente) with EV1 explain EXPL1,
  motorino_avviamento(muto) with EV2 explain EXPL2,
  valuta_antecedente([EV1, EV2], EV_ANT),
  valuta_consequente(EV_ANT, 0.8, EV) .
```

# PROLOG E SISTEMI ESPERTI

---

- **Vantaggi:**

- possibilità di sfruttare a fondo le caratteristiche del Prolog;
- efficienza;
- facilità di realizzazione.

- **Svantaggi:**

- approccio limitato all'uso di regole di produzione con strategia di inferenza backward;
- scarsa leggibilità e modificabilità dei programmi.

# META-INTERPRETAZIONE

---

- Il Prolog può essere utilizzato per definire linguaggi di rappresentazione della conoscenza e per la definizione di interpreti (motori inferenziali) per tali linguaggi.
- Si supponga che le regole di produzione siano rappresentate mediante asserzioni Prolog del tipo:

**CONSEG cert\_fact CF if ANTEC**

in cui **cert\_fact** e **if** sono operatori, **CONSEG** è un termine Prolog e **ANTEC** è una congiunzione di termini.

- Ogni fatto nella memoria di lavoro sia rappresentato mediante una asserzione del tipo:

**F with EV**

# ESEMPIO

---

- Definire un meta-interprete che realizza un motore inferenziale **backward** su tali regole. `solve (GOAL, EVID)`
- "GOAL può essere dimostrato con evidenza EVID utilizzando le regole contenute nel data base"

```
solve (true, 1) .
solve ( (A, B) , EVID) :-
    solve (A, EV1) ,
    solve (B, EV2) ,
    valuta_antecedente ([EV1, EV2] , EVID) .

solve (A, EV) :-
    A with EV.

solve (A, EV) :-
    A cert_fact CF if B,
    solve (B, EVID_ANTEC) ,
    valuta_consequente (EV_ANTEC, CF, EV) .
```

- Può essere facilmente esteso per fornire spiegazioni e per interagire con l'utente. Non ci si deve limitare alla strategia di ragionamento backward.

# ESEMPIO

---

- Meta-interprete che utilizza una strategia di inferenza **forward**.
- Consideriamo, in primo luogo, il caso di un interprete di base per regole rappresentate come asserzioni Prolog del tipo: **CONSEG if ANTEC**
- Un interprete forward per le regole è definito dal seguente programma:

```
interpreta:- <verifica se si è raggiunto un obiettivo>
```

```
interpreta:-      CONSEG if ANTEC,  
                  verifica_antec(ANTEC),  
                  not (CONSEG),  
                  assert (CONSEG),  
                  interpreta.
```

```
interpreta :- <riporta fallimento della dimostrazione>.
```

```
verifica_antec(ANTEC) "la congiunzione ANTEC è  
                      soddisfatta"
```

```
verifica_antec((A,B)) :-A,
```

```
verifica_antec(B) .
```

```
verifica_antec(A) :- A.
```

# REGOLE APPLICABILI

---

- Affinché una regola sia applicabile devono essere soddisfatte due condizioni:
  - L'antecedente della regola deve essere soddisfatto.
  - Il conseguente della regola non deve essere già vero. Ciò permette di evitare che una regola venga applicata più volte sugli stessi dati e che l'interprete vada in ciclo.
- L'interprete non ha una vera e propria fase di risoluzione di conflitti: viene semplicemente attivata la prima regola applicabile.
- Non è difficile realizzare un meta-interprete in cui le fasi di MATCH e CONFLICT-RESOLUTION sono separate.
- Supponiamo che le regole siano rappresentate da asserzioni del tipo:  
**regola(NOME, CONSEG if ANTEC)**  
in cui **NOME** è un nome che identifica univocamente ogni regola.

# INTERPRETE FORWARD:

## risoluzione di conflitti esplicita

---

```
interpretal :- <verifica se si è raggiunto un obiettivo>
```

```
interpretal :-
```

```
    match(REG_APPLICABILI),
```

```
    conflict_res(REG_APPLICABILI,REGOLA),
```

```
    applica(REGOLA),
```

```
    interpretal.
```

```
interpretal :- <riporta fallimento della dimostrazione>.
```

```
match(REG_APPLICABILI)
```

```
"REG_APPLICABILI: l'insieme di regole applicabili dato il  
    contenuto della memoria di lavoro (data base)"
```

```
match(REG_APPLICABILI) :-
```

```
    setof([REG,CONSEG],applic(REG,CONSEG),REG_APPLICABILI).
```



# INTERPRETE FORWARD:

## risoluzione di conflitti esplicita

---

`applic (REG, CONSEG)`

"la regola REG con conseguente CONSEG è applicabile"

```
applic (REG, CONSEG) :-  
    regola (REG, CONSEG if ANTEC),  
    verifica_antec (ANTEC),  
    not (CONSEG) .
```

`conflict_res (REG_APPLIC, [REG, CONSEG])`

"la regola REG con conseguente CONSEG è la regola selezionata all'interno della lista REG\_APPLIC di regole"

```
conflict_res (REG_APPLIC, [REG, CONSEG]) :-  
    <selezione della regola da applicare>  
    applica ([REG, CONSEG])  
    "applicazione regola REG con conseguente CONSEG"
```

```
applica ([REG, CONSEG]) :- assert (CONSEG) .
```

# MECCANISMO DI SPIEGAZIONE

---

- È facile aggiungere un meccanismo di spiegazione.
- È sufficiente modificare la definizione della relazione "applica" nel modo seguente:

```
applica ([REG, CONSEG]) :-  
    assert (CONSEG) ,  
    assert (dimostrato (CONSEG, REG)) .
```

- Una spiegazione può quindi essere fornita mediante il predicato:

```
spiega (GOAL) :-  
    dimostrato (GOAL, REGOLA) , ! ,  
    regola (REGOLA, GOAL if ANTEC) ,  
    write ('dimostrato ') , write (GOAL) ,  
    write ('utilizzando la regola ') ,  
    write (GOAL if ANTEC) .  
  
spiega (GOAL) :-  
    write ('fatto ') , write (GOAL) .
```

# COSTRUZIONE DI SISTEMI ESPERTI CON META-INTERPRETAZIONE

---

- **Vantaggi**

- elevata flessibilità;
- facilità di realizzazione dei meta-interpreti;
- leggibilità e modificabilità (almeno per i meta-interpreti semplici);
- portabilità;
- possibilità di definire meta-interpreti per diversi linguaggi di rappresentazione della conoscenza e diverse strategie di controllo.

- **Svantaggi**

- i meta-interpreti possono diventare difficili da mantenere se il linguaggio di rappresentazione e le strategie di controllo diventano molto complesse;
- elevata inefficienza dovuta alla sovrapposizione di uno o più livelli di interpretazione al di sopra di quello del Prolog;
- Per ovviare al problema di inefficienza è stato proposto di utilizzare tecniche di **valutazione parziale**.