# Rappresentazione della conoscenza

# Lezione 3

# Clausole

♢ I *letterali* sono atomi o atomi negati.

♢ Una *formula* è detta in *forma normale negativa* se il segno di negazione compare solo davanti agli atomi.

♢ Una *clausola* è una disgiunzione di letterali $L_1 \vee L_2 \vee \cdots \vee L_n$.

♢ Una *formula* è detta in *forma normale congiuntiva* o in *forma clausale* oppure si dice che essa è un *insieme di clausole* se è in forma normale negativa e inoltre ha la forma $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ dove le $C_i$ sono clausole.

# Clausole

♢ Poiché la disgiunzione è commutativa, una clausola generica può essere scritta come:

$$A_1 \vee A_2 \vee \cdots \vee A_n \vee \neg B_1 \vee \neg B_2 \vee \cdots \vee \neg B_m$$

dove gli $A_i$ e $B_j$ sono atomi.

♢ Una clausola verrà nel seguito anche indicata come l'insieme dei suoi letterali e cioè $\{L_1, L_2, \ldots, L_p\}$.

♢ Se $n = m = 0$ si ha la *clausola vuota* e si scrive $\{\}$, oppure $\diamond$, oppure [].

# Clausole di Horn

♢ Una clausola si dice **di Horn** se $n \leq 1$, cioè contiene al più un letterale positivo:

♢ Una clausola si dice **definita** se $n = 1$, cioè contiene esattamente un letterale positivo:

♢ Una clausola si dice **negativa** se $n = 0$, cioè non contiene letterali positivi:

♢ Una formula CNF si dice in clausole di Horn/definite se tutte le sue clausole sono clausole di Horn/definite

# Regole

Le clausole di Horn definite sono le **regole**

$P \vee \neg Q \vee \neg R$

che corrisponde a $Q \wedge R \rightarrow P$

```
P:-Q,R.
```

Un programma PROLOG (senza negazione) è un insieme di regole (clausole definite).

# Esistenziali nel corpo della regola

```
nonno(X,Z) :- padre(X,Y), padre(Y,Z).
nonno(X,Z) :- padre(X,Y), madre(Y,Z).
```

$\forall x \forall y \forall z (padre(x,y) \wedge padre(y,z) \rightarrow nonno(x,z))$
$\forall x \forall z (\exists y \ (padre(x,y) \wedge padre(y,z)) \rightarrow nonno(x,z))$
$\forall x \forall y \forall z (padre(x,y) \wedge madre(y,z) \rightarrow nonno(x,z))$
$\forall x \forall z (\exists y \ (padre(x,y) \wedge madre(y,z)) \rightarrow nonno(x,z))$

Nota: il goal negato è l'unica clausola negativa in un programma a clausole)

# Risoluzione SLD

Costruisce la derivazione della clausola vuota partendo con il goal negato e risolvendo ad ogni passo con una clausola ($c$) dell'insieme di partenza $S$). Derivazione-SLD

$c_1, \ldots, c_n$
tale che $c_n = c$, $c_1 \in S$,$c_{i+1}$ è un risolvente di $c_i$ ed una clausola di $S$.

Teorema per le clausole di Horn:

$S \vdash_{SLD}$ [] iff $S \vdash$ []

# Albero di dimostrazione

In generale:
ogni passo di risoluzione (binaria) è rappresentato come un albero (disegnato al contrario), la cui radice è il **risolvente** ed i successori della radice sono le due clausole **genitrici**

Nella risoluzione SLD:
il goal è la radice dell'albero ed ogni passo della dimostrazione si applica al risultato del passo precedente e ad una clausola della base di conoscenza (pettine).

# Esempio

$T$

$T \rightarrow C$

$C \wedge M \rightarrow B$

$I \rightarrow B$

$C \wedge F \rightarrow G$

$F$

Goal: $G$

Controesempio non-Horn: $[p, q], [p, \neg q], [q, \neg p], [\neg p, \neg q]$

# Albero di ricerca della dimostrazione

Nodi sono insiemi di atomi (da dimostrare). I successori si un nodo corrispondono alle possibili alternative di dimostrazione generate da uno degli atomi del nodo padre.

La ricerca della dimostrazione viene fatta:

- considerando le clausole nell'ordine in cui compaiono nel programma
- dimostrando gli atomi nel corpo delle clausole da sinistra a destra.

# Albero di ricerca della dimostrazione

Aggiungiamo:

$R \rightarrow G$
$F \rightarrow R$
$A \rightarrow C$
$A$

# Rappresentazione della conoscenza “a regole”

- **Programmazione logica**
- **Basi di dati deduttive**
- **Regole di produzione**

# Logica

- linguaggio: clausole (con simboli di funzione)
- semantica: in logica classica $\models$,
- inferenza: risoluzione

- clausole non-horn **disjunctive logic programs**
- Clausole Horn programmazione logica (+ negazione)

# Programmazione Logica

- linguaggio: clausole Horn (con simboli di funzione)
- semantica: in logica classica $\models$, (oppure puntofisso)
- inferenza: SLD risoluzione

Estensioni:

- negazione
- programmazione logica con vincoli (CLP), dove l'unificazione viene fatta in presenza di vincoli più deboli dell'uguaglianza

# PROLOG

Modello operazionale per la programmazione logica, incompleto per la dipendenza dall'ordine di scrittura delle clausole e degli atomi nel corpo delle clausole.

♢ Negazione come fallimento

♢ Predicati e funzionalità extra-logiche

# Basi di dati deduttive

- linguaggio: clausole senza simboli di funzione
- semantica: logica classica $\models$
- inferenza: metodi deduttivi per il calcolo proposizionale e metodi deduttivi ad hoc (computazione punto fisso etc.)

$\diamondsuit$ Ipotesi di **dominio finito** (costanti nella KB): BDD corrispondono a teorie proposizionali.

$\diamondsuit$ Horn DB (HDB) e Disjunctive DB (DDB)

# Basi di dati Horn (HDB)

Basi di dati Horn (HDB) o vivide, includono solo clausole Horn:

1. $\top \rightarrow A(x)$
2. $A_1(x) \wedge \ldots \wedge A_n(x) \rightarrow B(x)$
3. $A_1(x) \wedge \ldots \wedge A_n(x) \rightarrow \bot$

1. Fatti 2. Regole 3. Interrogazioni

3. possono anche essere letti come vincoli di integrità

# Inferenza nelle basi di dati Horn

MP è una regola di inferenza corretta e completa per HDB

Per HDB vale la proprietà dell'intersezione di modelli: quindi HDB hanno un *modello minimo*.

complessità dell'inferenza (senza simboli di funzione):
$n \times d$
dove $n$ è il numero massimo di premesse delle regole e $d$ è la dimensione del HDB (data complexity)

HDB è un esempio di come, restringendo il linguaggio si abbatte la complessità della deduzione.

# Backward chaining

$P_1 \rightarrow P_2$
$P_0 \rightarrow P_1$
$Q_1 \rightarrow P_2$
$Q_0 \rightarrow P_1$
$P_1 \rightarrow Q_2$
$P_0 \rightarrow Q_1$
$Q_1 \rightarrow Q_2$
$Q_0 \rightarrow Q_1$

Goal: $P_2$ oppure $Q_2$

La KB è lineare con $n$. La dimostrazione di fallimento richiede $2^n$ passi.

# DATALOG

DATALOG: linguaggio usato per le Basi di Dati Deduttive che corrisponde a HDB.

DATALOG: metodi di inferenza efficienti in avanti e all'indietro.

Per un HDB è facile costruire la chiusura deduttiva: applicando MP in tutti i modi possibili e aggiungendo le conclusioni, se non ci sono già.

DATALOG esteso per trattare la negazione nel corpo delle clausole secondo la semantica dei modelli stabili.

# Basi di dati disgiuntive (DDB)

Non tutte le formule della logica del primo ordine possono essere messe in forma di clausole Horn, e.g.

$piove \rightarrow (ombrello \vee giaccavento)$

Basi di dati disgiuntive (DDB):

$A(x)$
$A_1(x) \wedge \ldots \wedge A_n(x) \rightarrow B_1(x) \vee \ldots \vee B_n(x)$

Tutte le formule della logica del primo ordine possono essere messe in questa forma.

DLP è un sistema per DDB con la negazione.

# Regole

- linguaggio: regole IF-THEN (strutturalmente simile HDB)
- semantica: ($\models$ per la parte Horn pura, altrimenti procedurale)
- inferenza: concatenazione in avanti per la parte Horn pura, più modello procedurale **match-select-execute** (risoluzione e tableau)

$KB$ ha una rappresentazione del tipo:

$A(x)$
$IF \quad A_1(x) \wedge \ldots \wedge A_n(x) \quad THEN \quad B(x)$

Sistemi: OPS5, SOAR
Applicazioni: MYCIN, XCON,

# Regole di produzione

$IF \;\; condizione \;\; THEN \;\; azioni$

**condizione**:

`(type att:spec ...)`

dove `spec` = valore, variabile, espressione

**azioni**:

`ADD, REMOVE, MODIFY`

# Esempi

```
IF (student name:x) THEN ADD (person name:x)

IF (person age:x name:n) (birthday who:n)
THEN MODIFY 1 (age [x+1])
     REMOVE 2

(student name:daniele age:18)
(birthday name:daniele)
```

# Sistemi a regole di produzione

Ma nei sistemi a regole di produzione
(esempio più famoso OPS5, sviluppato a CMU per applicazioni Digital)

- ricerca di soluzione forward (data driven)

- decisione se regola applicabile: semiunificazione, cioè pattern matching su memoria di lavoro

- decisione su quale regola applicare: algoritmo di scheduling delle regole

# Sistemi a regole (in sintesi)

Struttura:

- ♢ Data Base di Regole
- ♢ Memoria di lavoro
- ♢ Ciclo: Riconosci-Seleziona-Agisci

Memoria di lavoro: memorizza stato iniziale, risultati parziali, conclusione

Riconosci: pattern matching tra parte sinistra regole e memoria di lavoro, creazione dell' "insieme dei conflitti"

Seleziona: individua tra le regole applicabili ("insieme dei conflitti") quella da utilizzare

Agisci: side effect nella memoria di lavoro, o sul mondo esterno

# Memoria di lavoro

```
(type att:val ...)
```

$\exists x \wedge att(x) = val \wedge \ldots$

```
(student name:daniele age:18)
(basicFact relation:olderThan firstArg:daniele secondArg:Jacopo)
```

Oggetti rappresentati implicitamente
Reificazione delle relazioni

# Rappresentazione RETE

RETE (Forgy '83): le regole vengono compilate in una "rete", cioè una struttura in cui

- viene condivisa quanto più possibile la parte sinistra:
  $if\ \ felino \wedge carnivoro \wedge maculato\ then\ giaguaro$
  $if\ \ felino \wedge carnivoro \wedge rigato\ then\ tigre$
- vengono collegate le regole in base alle dipendenze dalle premesse:
  $if\ carnivoro\ then\ aggressivo$

# Selezione

Insieme dei conflitti: lista di regole applicabili, con relativi legami per le variabili

- ordinamento
- specificità
- utilizzo recente
- non ripetizione

# Vantaggi dei sistemi a regole

♢ PATTERN DIRECTED PROCEDURE INVOCATION
il controllo passa dal programmatore all'algoritmo di scheduling, quindi programmazione più dichiarativa, più incrementale

♢ Spiegabilità delle conclusioni e delle linee di ragionamento seguite

SVANTAGGI:
♢ inefficienza
♢ si possono costruire DB di regole errate, contraddittorie
...

e.g. sia $KB$:
$if\ piovuto\ then\ pratobagnato$
$if\ pratobagnato\ then\ impiantoirrigazioneacceso$
$piovuto$

# Dimostratori di teoremi

Non sono veri e propri sistemi di rappresentazione della conoscenza, spesso consentono di fare le dimostrazioni, usando diversi metodi di ragionamento, a volte interattivamente.

♢ specifica del controllo
♢ verifica/deduzione automatica
♢ OTTER, PTTP, Boyer-Moore, ...

Applicazioni:

- dimostrazione di teoremi matematici
- sintesi automatica di programmi
- verifica e sintesi di circuiti

Nel corso di Ragionamento Automatico