Programmazione logica e linguaggi formali

Lezione 19 di Fondamenti di informatica

Docente: Giuseppe Scollo

Università di Catania
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica, I livello, AA 2008-09

Indice

clausole Horn strettamente positive

base della programmazione logica è una forma ristretta della logica predicativa, che limita le formule alle **clausole Horn strettamente positive:**

> clausole disgiuntive (con implicita quantificazione universale), in cui esattamente un letterale è positivo

il letterale positivo è detto **testa** della clausola

è facile mostrare l'equivalenza logica della clausola Horn $\{\overline{\gamma_1}, \ldots, \overline{\gamma_n}, \alpha\}$, di testa α, con la formula implicativa $(\gamma_1 \wedge \ldots \wedge \gamma_n) \rightarrow \alpha$, di letterali positivi

per qualche buona ragione, nella sintassi della programmazione logica è invalso l'uso di invertire l'ordine di antecedente e conseguente nella presentazione di tali implicazioni

> ad es., la sintassi Prolog di una clausola Horn è: $\alpha :- \gamma_1, \ldots, \gamma_n$

le teorie Horn non richiedono Skolemizzazione per la trasformazione in forma clausale

elementi della programmazione logica

primitive:

- **fatti:** clausole Horn strettamente positive costituite solo dalla testa
- **regole:** clausole Horn strettamente positive con almeno due atomi
- **programma:** insieme di fatti e regole
- ***query*:** predicato, di solito con variabili
- **soluzione:** sostituzione di termini chiusi per variabili della *query* che ne rende la corrispondente istanza deducibile dal programma

l'**ordine** di fatti e regole ***dovrebbe*** essere **irrilevante**

> nell'ideale della programmazione logica «pura»...

un programma determina implicitamente una segnatura algebrica:

> il suo universo di Herbrand costituisce lo spazio di valutazione delle variabili

con un dato programma, una *query* può avere una o più soluzioni, o nessuna

un esempio di programma Prolog

convenzioni sintattiche *Prolog*:

variabili: identificatori con iniziale maiuscola

simboli della segnatura: identificatori con iniziale minuscola

fatti e regole: come già indicato

"=" è soddisfatto dall'unificabilità dei due termini argomento, mentre il predicato binario infisso "**is**", predefinito, è soddisfatto dall'identità di valutazione aritmetica

ecco ad es. un programma Prolog per il calcolo del fattoriale:

```
fact(0,1)
fact(N,M) :- N1 is N-1, fact(N1,Z), M is N*Z
```

compilazione del programma ed esecuzione di una *query*:

```
?- [fact].
% fact compiled 0.00 sec, 1,004 bytes
?- fact(3,X).
X = 6
```

calcolare = dedurre

l'esecuzione di una *query* inizia con il tentativo di sua unificazione con la testa di una clausola del programma

- se l'unificazione ha successo con un fatto, l'unificatore determina una soluzione
- se l'unificazione ha successo con una regola, l'istanza del suo antecedente determinata dall'unificatore costituisce il nuovo ***goal*** da soddisfare
- il processo termina quando il *goal* è vuoto, nel qual caso la soluzione è ottenuta per composizione degli unificatori via via determinati, o altrimenti quando nessuna regola è applicabile a (ovvero ha testa unificabile con) alcun atomo del goal non vuoto

non è difficile riconoscere nel processo così delineato l'applicazione del principio di risoluzione quale meccanismo deduttivo

N.B. una *query* può ben essere priva di variabili

che risposta ci si può attendere dalla computazione in tal caso?

è ben possibile che il processo non termini...

regole DCG in Prolog

le **liste** sono una struttura dati predefinita in Prolog:

sintassi: `[], [Head | List]` e $[t_1,...,t_n]$

si possono inoltre rappresentare regole di una grammatica libera, dette ***Definite Clause Grammar (DCG) rules***, o semplicemente ***grammar rules***, automaticamente tradotte in clausole Horn positive secondo una tecnica detta delle **liste-differenza**

sintassi: t `-->` $t_1,...,t_n$ `.` (inoltre ";" separa alternative)

esempio 1: (i lessemi terminali sono in parentesi quadre)

```
frase --> soggetto, predicato ['.'].
soggetto --> articolo, sostantivo.
predicato --> verbo_intrans ; verbo_trans, compl_oggetto.
compl_oggetto --> articolo, sostantivo.
sostantivo --> [nutrice] ; [bimbo].
articolo --> [il] ; [la] ; [un] ; [una].
verbo_intrans --> [dorme].
verbo_trans --> [nutre].
```

vincoli sintattici con regole DCG

si possono imporre **vincoli di concordanza** usando **variabili** e struttura di **termini** per simboli non terminali

esempio 2: così, rispetto all'esempio 1, si possono modificare le regole per `soggetto`, `compl_oggetto`, `articolo` e `sostantivo` come segue:

```
soggetto --> articolo(Gen), sostantivo(Gen).
soggetto --> articolo(Gen), sostantivo(Gen).
sostantivo(f) --> [nutrice].
sostantivo(m) --> [bimbo].
articolo(f) --> [la] ; [una].
articolo(m) --> [il] ; [un].
```

compilazione del programma ed esecuzione di *query*:

```
?- [nbg2].
% nbg2 compiled 0.00 sec, 3,096 bytes
?- phrase(frase, [il, bimbo, X, .]).
X = dorme ;
No
?- phrase(frase, [X, Y, nutre, il, bimbo, .]).
X = la
Y = nutrice
Yes
```

analisi sintattica con regole DCG

variabili e struttura di **termini** per simboli non terminali possono essere usati anche per la **costruzione dell'albero sintattico** di una data frase
a tal fine le regole DCG della grammatica vanno modificate così che la *query*
`phrase(frase(X), [ lista_terminali ]).`
abbia come soluzione `X = t`, dove `t` sia l'albero sintattico di `lista_terminali`
esempio 3: ecco una modifica in tal senso rispetto all'esempio precedente:

```
frase(fs(S,P)) --> soggetto(S), predicato(P) ['.'].
soggetto(sg(A,S)) --> articolo(Gen,A), sostantivo(Gen,S).
predicato(pi(V)) --> verbo_intrans(v).
predicato(pt(V,O)) --> verbo_trans(V), compl_oggetto(O).
compl_oggetto(co(A,S)) --> articolo(Gen,A), sostantivo(Gen,S).
sostantivo(f,s(nutrice)) --> [nutrice].
sostantivo(m,s(bimbo)) --> [bimbo].
articolo(f,a(la)) --> [la].
articolo(f,a(una)) --> [una].
articolo(m,a(il)) --> [il].
articolo(m,a(un)) --> [un].
verbo_intrans(vi(dorme)) --> [dorme].
verbo_trans(vt(nutre)) --> [nutre].
```

automazione dell'analisi sintattica

la tecnica esemplificata sopra trasforma in modo relativamente meccanico una grammatica libera in una DCG Prolog che può essere compilata ed eseguita per ottenere l'analisi sintattica delle espressioni del linguaggio che essa genera
esempio: compilazione del programma ed esecuzione di *query*:

```
?- [nbg2p].
% nbg2p compiled 0.00 sec, 4,136 bytes
?- phrase(frase(X), [la, nutrice, nutre, il, bimbo, .]).
X = fs(sg(a(la), s(nutrice)), pt(vt(nutre), co(a(il), s(bimbo)))) ;
No
?- phrase(frase(X), [la, nutrice, Y, .]).
X = fs(sg(a(la), s(nutrice)), pi(vi(dorme)))
Y = dorme ;
No
?- phrase(frase(X), [Y, Z, W .]).
X = fs(sg(a(la), s(nutrice)), pi(vi(dorme)))
Y = la
Z = nutrice
W = dorme
Yes
```