Universita' degli Studi di Udine

FACOLTA' DI SCIENZE MATEMATICHE FISICHE E NATURALI

UNA PROPOSTA PER L'INTRODUZIONE DI CAPACITA' DI META-RAPPRESENTAZIONE IN UN LINGUAGGIO DI PROGRAMMAZIONE LOGICA

Relatore: Prof. Gianfranco Rossi

Laureando: Iliano Cervesato

Anno Accademico 1989-90

SOMMARIO

- **Meta-programmazione**
- **'Log**
- **SFOL**

Meta-programmazione, una definizione

"La meta-programmazione e' la scrittura di programmi che trattano altri programmi come dati."

Origini della meta-programmazione

Logica matematica (anni '20)

- distinzione tra un livello oggetto e un meta-livello
- Teoremi di Gödel (1931)

Deduzione automatica (anni '60 - '70)

- maggiore potenza espressiva
- possibilita' di accorciare le dimostrazioni al livello oggetto

IA (anni '60 - '70)

- conoscenza di controllo (heuristiche)
- rappresentazione della conoscenza
- potenziamento espressivo

Linguaggi di meta-programmazione (anni '80)

Strumenti linguistici general purpose per distinguere tra un livello oggetto e un meta-livello

Applicazioni

- **Sistemi di deduzione automatica**
- **Euristica e inferenza del controllo**
- **Sistemi basati sulla conoscenza**
- **Modularizzazione**
- **DBMS**
- **Ambienti di programmazione integrati**

Meta-programmazione nei linguaggi logici

- ✓ • **Prolog: alcuni built-in extalogici**
- ✓ • **1982: Bowen e Kowalski**
- • **1985: MetaProlog (Bowen et al.)**
- • **1987: Reflective Prolog (Costantini, Lanzarone)**
- • **1990: Gödel (Burt, Hill, Lloyd)**
- • **...**

L'approccio di Bowen&Kowalski

Dati due linguaggi logici L ed M, M e' un meta-linguaggio per L sse:

- e' possibile rappresentare tutte le espressioni linguistiche di L mediante termini in M

Φ **in L** $\longrightarrow$ **"**Φ**" in M (nome di** Φ**)**

- e' possibile definire in M un predicato demo/2, per mezzo dell'insieme di clausole Pr, che rappresenti correttamente la relazione di derivabilita' di L $\vdash_L$ **, ossia tale che**

Pr $\vdash_M$ **demo("P","f") sse P** $\vdash_L$ **f**

Pr e' l'assiomatizzazione (in M) della derivabilita' del linguaggio oggetto L

Caso interessante $L = M$

Meta-programmazione in Prolog

- E' semplice scrivere certi meta-programmi (ex. il Vanilla meta-interpreter)

- Per programmi piu' sofisticati, e' necessario ricorrere a primitive extra-logiche (var, arg, functor, name, ecc)

- Non fornisce strumenti per distinguere tra il livello oggetto e il meta-livello

Non e' in generale possibile attribuire una semantica logica ad un meta-programma scritto in Prolog

Si vogliono pertanto definire dei linguaggi di meta-programmazione logica che estendano

dichiarativamente

Prolog con capacita' di meta-rappresentazione

Vanilla

```
solve(true).
solve(A&B) :-  solve(A), solve(B).
solve(A) :-  clause(A,B), solve(B).
```

PRO:

- molto semplice
- utilmente estendibile

CONTRO:

- i programmi oggetto non sono termini
- il meta-interprete risiede fisicamente nello stesso database del programma interpretato
- vi possono essere essere istanziazioni accidentali di variabili oggetto e meta-variabili
- richiede una logica tipata per essere correttamente interpretato
- e' poco efficiente.

Obiettivi di 'Log

- **Trattare al meta-livello qualsiasi entita' sintattica del linguaggio, dai programmi ai simboli**

- **Permettere un'implementazione efficiente, in termini di tempo di esecuzione ed occupazione di memoria**

- **Possedere una semantica logica per tutto il linguaggio, compresi i meta-predicati: e' un linguaggio di meta-programmazione logica**

- **Offrire strumenti di meta-programmazione che siano sufficientemente naturali e semplici da usare**

Caratteristiche di 'Log

- **possiede un doppio schema di auto-rappresentazione per ogni sua entita' sintattica (nomi e rappresentazioni strutturali)**

  Vantaggi:
  - efficienza implementativa

  - differenti visioni della stessa entita' sintattica

- **il livello oggetto e il meta-livello sono** separati

Nomi

Il nome di un'entita' sintattica e' una costante atomica ad essa sintatticamente simile.

		Nomi
Programmi:	**P**	**{P}**
clausole:	**C**	**'C'**
termini:	**t**	**'t'**
simboli:	**s**	**`s`**
caratteri:	**c**	**%c**

Esempi

'nn(X,Y):-nn(Y,X)' e' il clause name di nn(X.Y):-nn(Y,X)

'nn(a,b,X)' e' il term name di nn(a,b,X)

`Alfa` e' il symbol name di Alfa

%l e' il character name di l

Rappresentazioni strutturali

La rappresentazione strutturale di un'entita' sintattica e' un termine ground che ne esprime la struttura in funzione dei nomi delle entita' componenti

L'utente puo' scegliere tra:

- **una notazione esplicita (a liste), oppure**
- **una notazione sintetica**

Esempi

sintetica		esplicita
{{p(a):-q(a),r(a).q(b).}}	**=**	**['p(a):-q(a),r(a)', 'q(b):-']**
"p(a):-q(a),r(a)"	**=**	**clause('p(a)', ['p(a)', 'r(a)']**
"f(a,b,g(c))"	**=**	**[`f`, `a`, `b`, "g(c)"]**
^pippo^	**=**	**[%p, %i, %p, %p, %o]**

Rappresentazioni strutturali incomplete

Sono termini che assomigliano a rappresentazioni strutturali se non per la presenza di (meta-) variabili al loro interno

Esempio **[`f`, X]**

'Log rende disponibile una notazione sintetica anche per esse: si fanno precedere le variabili oggetto da #

Esempio

sintetica		esplicita
[`f`, `X`]	**=**	**"f(#X)"**
[`f`, X]	**=**	**"f(X)"**
[F, `X`]	**=**	**"F(#X)"**
[F, X]	**=**	**"F(X)"**

L'operatore <==>

Permette di passare dal nome di un'entita' sintattica alla sua rappresentazione strutturale, e viceversa

N <==> S

- **ha successo riportando una sostituzione se N e S sono corretti e almeno uno non contiene meta-variabili**
- **fallisce se N o S non sono corretti**
- **Altrimenti viene ritardato ed eventualmente restituito come vincolo (reificatore)**

Esempi

```
?- N <==> "f(a, #X)"
        N --> 'f(a,X)'

?- 'f(a,b)' <==> "F(lArgs)"
        F  --> `f`
        Args --> [`a`,`b`]

?- N <==> f(a,b)
        no

?- N <==> "F(a,b)"
        N <==> "F(a,b)"
```

Proprieta' semantiche

Definendo la semantica di 'Log,il trattamento dell'operatore <==> comporta notevoli cambiamenti rispetto al caso standard:

- **interpretazione di un programma P**
- **risposta corretta ad un goal G in un programma P: e' ora una coppia (R,θ)**
- **derivazione**

La procedura di risoluzione di 'Log e' stata dimostrata

- **corretta**
- **completa**
- **indipendente dalla regola di selezione dei letterali**

Manipolazione di programmi

Vengono fornite in 'Log versioni dichiarative (scritte in 'Log) di clause/2, assert/1, retract/1 di Prolog, nonche' di demo/2

ecall(PgName, GoalName, Constraints, Subs)

eclause(ProgramName, ClauseName, Subs)

eassert(PgName, ClauseName, NewPgName)

eretract(PgName, ClauseName, NewPgName, Subs)

NB. Per motivi di efficienza, conviene implementarli proceduralmente

Conclusioni

Si'

in corso

Si'

da sperimentare

* 'Log e' all'altezza delle migliori proposte nel settore

Sviluppi futuri

- **sperimentazione**
- **implementazione**
- **aggancio con il Constraint Logic Programming (CLP)**

SFOL
(Simplified First Order Logic)

E' un formalismo logico equivalente al calcolo dei predicati del prim'ordine. E' caratterizzato da:

- **assenza di simboli predicativi**
- **rimozione della distinzione tra variabili e altri simboli del linguaggio**

Conseguenze:

- **non vi possono essere formule aperte**
- **riporta nel linguaggio formule (?) della forma**

$$\forall x(x \longrightarrow p(x))$$

accettata da molti interpreti Prolog come

p(X) :- X