Implementazione dell'algoritmo di Martelli-Montanari

Corso di Laurea in Matematica - Corso di Paradigmi di Programmazione

Emanuele Bigiarini

Sommario

Si descrive una realizzazione dell'algoritmo di unificazione di Martelli e Montanari segendo l'articolo, *A.Martelli, U. Montanari: An Efficient Unification Algorithm, in ACM TOPLAS, 1982.*

1 Strutture dati

Lo strumento fondamentale dell'algoritmo di M-M è la *multiequazione*, una generalizzazione di un'equazione, usata per raggruppare termini da unificare. La notazione usata è $\{S = M\}$ dove S è un insieme non vuoto di variabili e M è un *multitermine*. Nel seguito dell'algoritmo viene assegnato un contatore ad ogni multiequazione, in modo che, scegliendo quella con contatore 0, sia possibile ridurre il numero di passi dell'algoritmo. Pertanto useremo anche la struttura $\{C\,,\, S = M\}$, dove C è il contatore.

Un multitermine non è altro che un termine della forma $f\,(P_1, \ldots, P_n)$, dove f è un simbolo di funzione di arietà n e P_i sono coppie $\{S_i, M_i\}$, con S_i un insieme di variabili e M_i un multitermine.

Dovendo lavorare su un sistema di multiequazioni la scelta naturale sarebbe stata rappresentare tale insieme come una lista, così il numero di equazioni che contiene inizialmente il sistema è pari al numero di variabili. Durante l'esecuzione dell'algoritmo si richiede necessario, data una variabile V, trovare la multiequazione del sistema che contiene nella sua parte S la variabile V. Rappresentando il sistema come una lista ciò porterebbe a un costo pari a $O(n)$, dove n è il numero di variabli. Si è scelto di utilizzare una rappresentazione ad albero, i cui nodi fossero:

$$V\ -\ Meq$$

dove V è una variabile e Meq è la multiequazione contenente la variabile V nella sua parte S.

Costruendo l'albero perfettamente bilanciato, sfruttando l'ordinamento totale tra variabili definito nel Prolog, e modificando la multiequazione associata ad

ogni variabile nei nodi dell'albero (quando richiesto dall'algoritmo), possiamo usare l'albero per la ricerca della multiequazione con complessità $O(\log_2 n)$, dove n è il numero di variabili iniziali. Ad esempio, su un sistema con inizialmente 30 variabili, la ricerca mediante albero bilanciato significa $[\log_2 30]+1 = 5$ chiamate ricorsive del predicato di ricerca nell'albero.

Si rappresentano gli alberi mediante liste, ovvero `[Node, Left, Right]`.

Inoltre si fa uso di liste differenza dove utile, per la loro efficiente concatenazione.

2 Predicati dell'algoritmo di unificazione

I predicati che riguardano l'unificazione lavorano sull'albero di multiequazioni, sulle equazioni stesse e sui termini.

Il predicato `unify_sys/5` rappresenta un passo dell'algoritmo:

`unify_sys(U_tree,T_temp_sys,T_sys,Zero_meq_list,Counter)`

T_temp_sys è la parte temporaneamente triangolarizzata del sistema individuato da U_tree, dove Zero_meq_list è la lista di multiequazioni con contatore 0 e Counter è il numero di equazioni ancora da eliminare. `unify_sys` realizza una ricorsione tail che si interrompe nel caso in cui la Zero_meq_list sia vuota e ci siano ancora equazioni da risolvere (in caso di ciclo, ad es. f(X)=X) oppure ha come base della ricorsione il caso in cui non vi siano più equazioni da risovere (Counter=0). `unify_sys` "chiama" `cpf/3`, che calcola parte comune e frontiera di un multitermine (in questo caso quello contenuto nella multiequazione con contatore 0), e `compact/7`, che compatta la frontiera nell'albero, al fine di realizzare la successiva chiamata ricorsiva.

Il predicato `cpf/3`:

`cpf(T,C,F)`, C è la parte comune e F la frontiera del multitermine T. Dovendo scendere in profondità nel termine produce una forma di ricorsione tree. Lo stesso vale per `merge_mt(T1,T2,T3)` dove T3 è il multitermine merge di T1 e T2, come descritto nell'articolo.

L'altro predicato fondamentale è

`compact([Frontier,Temp_tree,Tree,Temp_z_meq,Z_meq,Temp_el_var,El_var)`

che compatta ricorsivamente ogni equazione della frontiera nell'albero, aggiornando la lista delle equazioni con contatore nullo. `compact` ha il predicato ausiliario `compact_iter` tail ricorsivo sulla lista di variabili dell'equazione da compattare che sfrutta il predicato `m_tree` per la ricerca e modifica all'interno dell'albero:

`m_tree(Tree_old,V,M1,M2,Tree_new)`, dove Tree_new è l'albero Tree_old a cui è stato sostituito il nodo V-M1 con V-M2.

3 Predicati di preparazione

Se si vuole unificare una lista di termini attraverso l'algoritmo si deve trasformare la lista in un multitermine. Il predicato `trasf_begin/3` si occupa di questo. Fa di più:

`trasf_begin(List,Meq,Vars)`

Meq è la multiequazione `{S=Multiterm}`, dove S è la lista di variabili in List (eventualmente vuota). Vars è la lista differenza di variabili (tante quante sono le occorrenze) nel multitermine, servirà per la costruzione del sistema di partenza.

Ad esempio:

```
?- L=[f(X1,g(X2,c)),f(X2,g(h,c)),Y], trasf_begin(L,Meq,Vars).
L = [f(X1, g(X2, c)), f(X2, g(h, c))],
Meq = {[Y]=f({[X1, X2]=[]}, {[]=g({[X2]=h}, {[]=c})})},
Vars = [X2, X1, X2|_G480]-_G480.
```

Ordinate le variabili, il predicato `build_sys/7` costruisce il sistema iniziale delle equazioni delle variabili del multitermine trovate in precedenza, tendendo conto del numero di occorrenze di ciascuna. Così ad esempio:

```
?- build_sys([X1,X2,X2,X3],nil,0,Nil-Nil,Sys-[],0,N).
Nil = [{1, [X1]=[]}, {2, [X2]=[]}, {1, [X3]=[]}],
Sys = [{1, [X1]=[]}, {2, [X2]=[]}, {1, [X3]=[]}],
N = 3.
```

Se la multiequazione ottenuta con `trasf_begin` è della forma `{[]=Multiterm}`, allora si può creare l'albero a partire dal sistema appena ottenuto (questo compito è affidato al predicato `crea_albero/2` che crea un albero bilanciato) e inserire la multiequazione `{0, [New_var]=Multiterm}` nella lista di mutiequazioni con contatore 0.

Se invece la multiequazione ha la forma `{S=Multiterm}`, come prima si crea l'albero e inoltre se esistono variabili in S che compaiono nel Multiterm (e quindi nell'albero) si deve aggiornare l'albero e ricalcolare il contatore della multiequazione. Questi compiti sono affidati ai predicati `sys_tree/4` e `counter_me/5`.

4 Esempi

Il predicato `unify_mm/2` è quello utile per eseguire il calcolo dell'unificazione:

unify_mm(List, System)

`List` è la lista di termini da unificare, *`System`* è il sistema *triangolare*.

Grazie all'ordinamento parziale sull'insieme di equazioni in testa a tale ordinamento c'è l'equazione con contatore 0. Che a un certo passo dell'algoritmo la lista Zero_meq_list sia vuota è sintomo della presenza di un ciclo, ovvero si cerca di unificare una certa variabile con un termine che la contiene. Si realizza così l'*occur check*. Un esempio:

```
?- unify_mm([X, f(X)], T_sys).
Error: cycle
false.
```

Un altro esempio tratto dall'articolo di Martelli e Montanari è il seguente, si vuole unificare i termini

$$f(x_1, g(x_2, x_3), x_2, b) = f(g(h(a, x_5), x_2), x_1, h(a, x_4), x_4)$$

```
?- unify_mm([f(X1,g(X2,X3),X2,b),f(g(h(a,X5),X2),X1,h(a,X4),X4)], T_sys).
T_sys = [[X5, X4]=b,
         [X3, X2]=h(a, X4),
         [X1]=g(X2, X2),
         [_G1497]=f(X1, X1, X2, X4)].
```

Un altro esempio:

```
?- unify_mm([X1, f(X2,X3), f(X3,X2)], T_sys).
T_sys = [[X3, X2]=nil, [X1]=f(X2, X3)].
```

Quando nel sistema triangolare abbiamo `List=t` significa che le variabili in `List` vengono tutte unificate a `t`. Se `t` è `nil`, significa che non vengono unificate con nessun altro termine. Usando questi due predicati, possiamo definitivamente realizzare l'unificazione:

```
risolvi([]).
risolvi([[V|L]=nil|T])   :-   unif_lista(V,L), !, risolvi(T).
risolvi([[V|L]=M|T])     :-   unif_lista(V,L), V=M, risolvi(T).

unif_lista(_,[]).
unif_lista(V,[H|T])      :-   V=H, unif_lista(V,T).
```

E applicato a uno dei precedenti esempi, avremmo:

```
X1 = g(h(a, b), h(a, b)),
X2 = h(a, b),
X3 = h(a, b),
X5 = b,
X4 = b
```