Implementazione di un interprete Prolog in C.

di Michele Padovani
Andrea Vicentini

INDICE

1. Struttura del sorgente C

1.1 Moduli

Modulo 'p97'

E' il modulo principale: contiene
- la funzione *main():* inizializza il DataBase e realizza il ciclo di lettura query chiamata della funzione *solve();*
- la funzione *solve():* gestisce le chiamate alla *solve_1st()* e alla *solve_back();*
- le funzioni *mymalloc()* e *myfree()* (che fanno uso della variabile statica *nmall*), per controllare il corretto funzionamento dell'uso della memoria da parte del programma;

Modulo 'clause'

Gestisce il DataBase delle clausole sorgenti; contiene, tra le altre:
- la variabile *src*: e' il puntatore alla testa della coda di memorizzazione del DataBase.
- la funzione *readsource():* legge il sorgente tramite la *read_cl()* e ne colloca le clausole nella apposita struttura dati.

Modulo 'unify'

Realizza l'algoritmo di unificazione; contiene:
- la funzione *unify():* unifica le due strutture passatele come argomenti;
- la funzione *find_cl():* trova, nel DataBase, la prossima clausola che unifica con il sottogoal.

Modulo 'struct'

E' sostanzialmente il modulo che implementa il tipo di dato OBJ; contiene, tra le altre:
- la funzione *dolist():* richiamandosi ricorsivamente, crea un oggetto OBJ a partire dalla stringa corrispondente.
- la funzione *rename_cl():* rinomina le variabili della clausola sorgente. E' importante notare che e' l'unica funzione che elabora stringhe e non oggetti OBJ.
- la funzione *copy_st():* crea una copia di un oggetto OBJ.
- la funzione *destroy_st():* distrugge un OBJ.
- la funzione *scan_st():* ricerca nell'oggetto indicato le variabili, creandole se non gia' esistenti.
- la funzione *spec_st():* crea una copia dell'oggetto in ingresso, sostituendo alle variabili istanziate il loro valore.

- la funzione *free_var_st():* libera le variabili dell'OBJ dal loro valore.

Modulo 'status'

Gestisce lo stack dei sottogoal (stato del programma); contiene:
- la variabile globale *sts_sp:* e' il puntatore alla cima dello stack dei sottogoal.
- la funzione *push_sts():* aggiunge alla cima dello stack il sottogoal indicato, nel caso in cui sia possibile il matching con la testa di una clausola del DataBase. Usa la *spec_st()*.
- la funzione *pop_sts():* distrugge le strutture allocate sulla cima dello stack.
- la funzione *solve_1st():* cerca la prima soluzione del goal, arrestandosi comunque al primo fallimento (non fa backtracking).
- la funzione *solve_back():* cerca la prossima soluzione in backtracking.
- la funzione *redo():* ripristina lo stato delle variabili al momento della creazione dello stato ora in cima allo stack (puntato da *sts_sp*).

Modulo 'var'

Implementa il tipo di dato VAR; contiene, tra le altre:
- la variabile *var_sp*: e' il puntatore alla cima dello stack delle variabili.
- la funzione *getobj():* se l'oggetto in ingresso indica una variabile, ne ritorna il valore. Se questo indica un'altra variabile, si richiama ricorsivamente fino a che non trova un'istanza non variabile.
- la funzione *print_goal_var():* stampa le variabili della query.
- la funzione *free_var():* libera la variabile dal suo valore, distruggendolo.
- la funzione *release_var():* elimina dalla cima dello stack tutte le variabili fino al punto indicato, che diventera' quindi la nuova cima.
- la funzione *inlist_var():* ritorna la variabile, se e' nello stack.
- la funzione *new_var():* genera un nuovo nome di variabile. E' usata da *rename_cl()*.
- la funzione *add_var():* mette la nuova variabile sullo stack, inizializzandola.
- la funzione *link_var():* istanzia la variabile indicata dal primo OBJ al secondo OBJ.

Modulo 'stack'

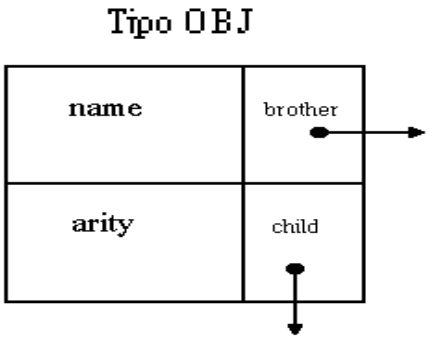
Implementa il tipo di dato astratto STACK, usato dai moduli *status* e *var*.

Modulo 'predef'

Realizza alcuni predicati predefiniti.: cut, fail, nonvar, integer, write, nl, test.

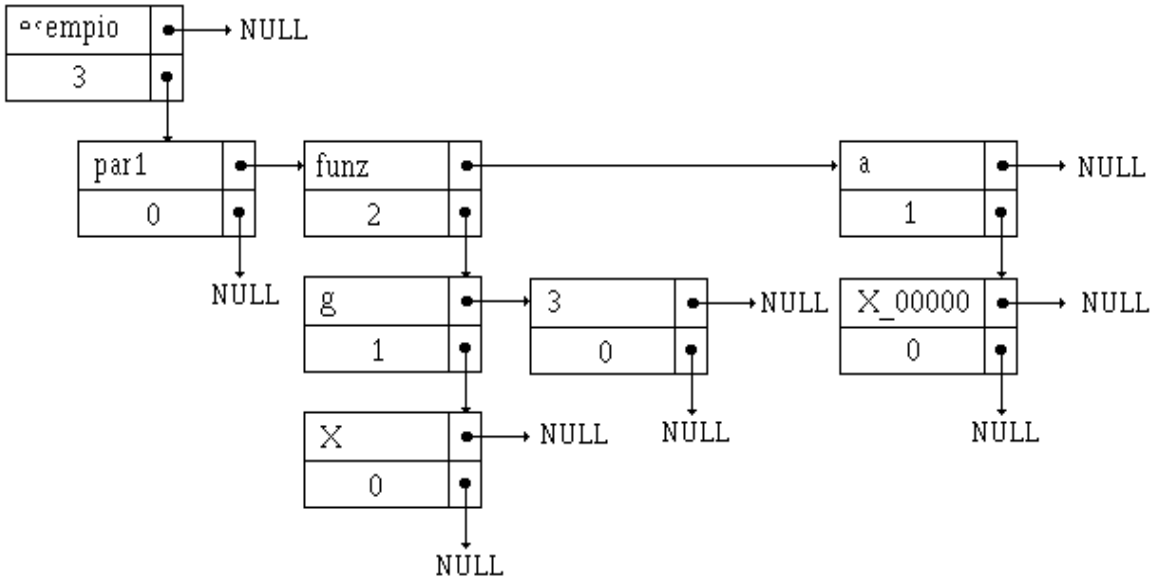
1.1 Strutture dati

Struttura OBJ

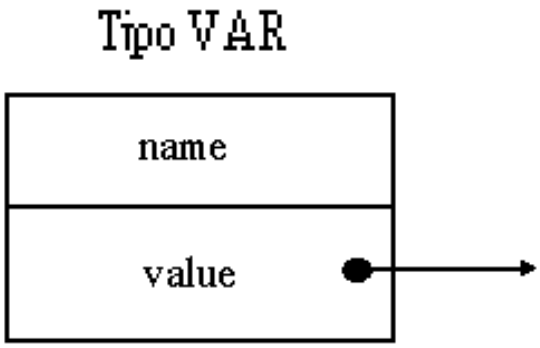


Definita nel modulo struct, e' la struttura ricorsiva che memorizza ogni termine o predicato.
Esempio di utilizzo:

esempio(par1,funz(g(X),3),a(X_00000))



Struttura VAR



Associa un nome di variabile all'OBJ puntato da value; value=NULL se la variabile non e' istanziata.

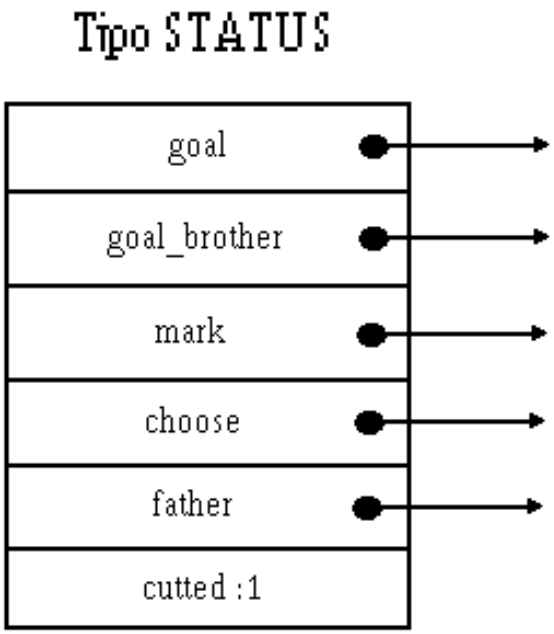
Struttura STACK

Realizza il generico elemento di uno stack, in cui il campo data e' il puntatore all'informazione, di qualunque natura essa sia. Infatti nel programma e' usato sia per la gestione dello stato, che per le variabili.

Struttura DB

Realizza il generico elemento della coda per la memorizzazione delle clausole del DataBase.

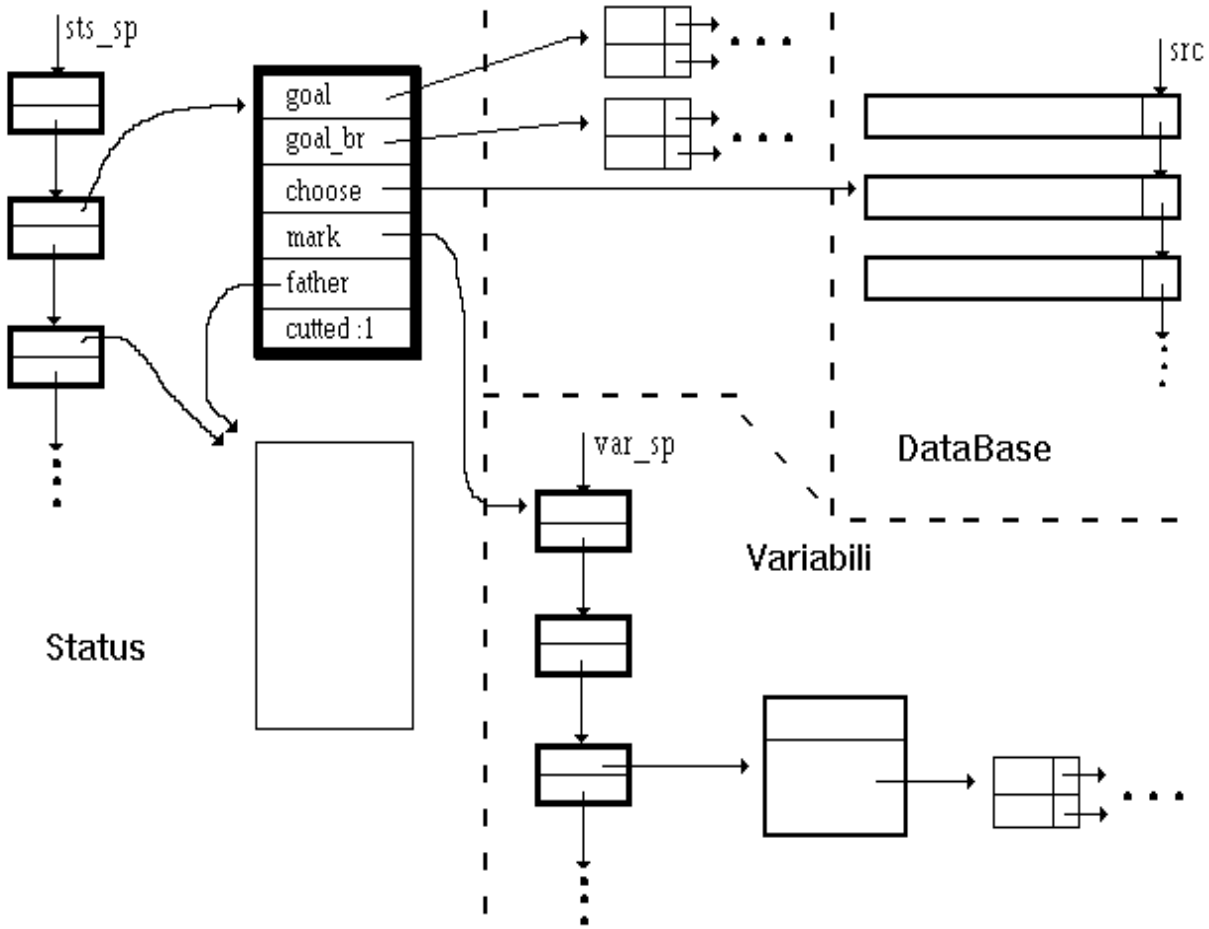
Struttura STATUS



E' la struttura dati che memorizza lo stato dei vari sottogoal. Significato dei vari campi:

- *goal*: punta il sottogoal da eseguire;
- *goal_brother*: punta al primo fratello nella lista dei sottogoal del body a cui appartiene *goal*;
- *mark*: subito prima della risoluzione di *goal*, viene inizializzato alla cima dello stack delle variabili;
- *choose*: punta, nel DataBase, alla clausola con la cui testa *goal* ha unificato;
- *father*: punta allo STATUS padre, nel cui risolvente figurava *goal*;
- *cutted*: e' un flag che e' settato a 1 quando i choose point devono essere ignorati.

1.3 Visualizzazione grafica dei legami tra le varie strutture



2. Limiti e prestazioni dell'interprete

La piu' grave mancanza del nostro interprete e' sicuramente l'assenza di un'analisi sintattica del sorgente, prima di ogni elaborazione; questa e' giustificata dal fatto che e' inessenziale ai nostri scopi, e risulta comunque accettabile nei limiti in cui si consideri il sorgente privo di errori. Comporta vari problemi:

- la gestione sintattica delle liste ne risulta penalizzata: per consentirne l'uso e' comunque possibile utilizzare la forma equivalente:

  [1,3,5,a,b] = -(1,-(3,-(5,-(a,-(b,nil))))).

- la realizzazione dei metainterpreti necessita di reificazione: non avendo realizzato l'analisi sintattica del body, non conosciamo l'*espressione* corrispondente con l'operatore 'virgola' infisso come funtore; la reificazione esplicita l'espressione in forma prefissa, usando il predicato '-' come funtore.

Dai programmi di test risulta evidente la lentezza dell'algoritmo di risoluzione, soprattutto se paragonata alle versioni di Prolog in circolazione. La giustificazione sta nel fatto che si e' ricercata la linearita' e semplicita' delle strutture dati, nonche' la leggibilita' del codice; tutto questo a scapito dell'efficienza degli algoritmi e dell'uso della memoria, che non risulta particolarmente oculato. A questo proposito abbiamo imposto una limitazione sulla lunghezza massima dell'identificatore a 10 caratteri.

Sono stati realizzati alcuni predicati predefiniti, consentendo inoltre la possibilita' di una facile estensione di questi. Ad esempio si e' implementato cut, fail, nonvar /1, ecc.. Il predicato test e' stato scritto ad hoc per il testing di crypto.p00 (aritmetica criptata): e' risultato necessario poiche' non sono stati implementati i predicati is /2, + /2, // /2, mod/2.

Segue una breve lista di particolarita' tecniche di importanza minore:

- abbiamo scelto di procedere comunque al backtracking forzato da una richiesta di nuove soluzioni anche in presenza di una query ground;
- il predicato predefinito call /1 e' implementato implicitamente con l'equivalenza call(X)=X;
- adeguandoci alle versioni di Prolog a nostra disposizione non abbiamo implementato l' 'occur check';
- risulta complicata la conversione del motore di inferenza da noi realizzato al tipo di ricerca breadth first;
- non e' stato realizzato l'operatore ';';
- la stampa delle variabili del goal avviene in ordine inverso per comodita' di programmazione.

3. Programmi di test.

% crypto.p00

```
% aritmetica criptata

lega(X, D, D) :- nonvar(X), !.
lega(X, D, D2) :-
  member(X, D),
  delete1(X,D,D2).

sum2(Rin,A,B,C,Rout,D,Dout) :-
    lega(A, D, D2),
    lega(B, D2, D3),
    lega(C,D3,Dout),
    test(C,Rout,A,B,Rin).

suml(-(T1,nil), -(T2,nil),-(T,nil),Rout,D,Dout) :-
  sum2(0,T1,T2,T,Rout,D,Dout).
suml(-(T1,C1),-(T2,C2),-(T,C),Rout,D,Dout) :-
  suml(C1,C2,C, R1,D,D1),
  sum2(R1, T1, T2, T, Rout,D1,Dout).

sum(A, B, C) :-
  suml(A, B, C, 0, -(0,-(1,-(2,-(3,-(4,-(5,-(6,-(7,-(8,-
(9,nil))))))))),Dout).

member(X,-(X , L)).
member(X,-(_ , L)) :- member(X, L).

delete1(X,nil,nil) :- !.
delete1(X,-(X, T),T) :- !.
delete1(X,-(H , T),-(H , L)) :- delete1(X, T, L).
```

% sum.p01

```
sum(0, X, X).
sum(s(X), Y, s(Z)) :- sum(X, Y, Z).

% proposed querys
% sum(s(0), s(s(s(0))), X).
% sum(s(0), Y, s(s(s(0)))).
% sum(X, Y, s(s(s(0)))).
% sum(X, Y, s(s(s(0)))), sum(X, s(0), Y).
```

% list.p02

```
% cerca elemento
member(El, -(El, _)).
member(El, -(_, T)) :- member(El, T).

% cerca elemento unico
member!(El, -(El, _)) :- !.
member!(El, -(_, T)) :- member(El, T).

% linka due liste
app(nil, L, L).
app(L, nil, L).
app(-(H, T), L, -(H, R)) :- app(T, L, R).

% cancellazione di un elemento
delete(X, -(X, T), T).
delete(X, -(H , T), -(H , L)) :- delete(X, T, L).

% cancellazione di un elemento unico
delete!(_, nil, nil) :- !.
delete!(X, -(X, T), T) :- !.
delete!(X, -(H , T), -(H , L)) :- delete(X, T, L).

% inverte una lista
rev(nil, nil).
rev(-(Head, nil), -(Head, nil)) :- !.
rev(-(Head, Tail), Result) :- rev(Tail, Temp), app(Temp, -(Head,
nil), Result).

% negazione
not(X) :- X, !, fail.
not(_).
```

```
% meta.p03
% meta-interprete vanilla e right-most

normal(X) :- b(X), c(X).
right_m(X) :- c(X), b(X).

b(1).
b(2).
c(2).
c(1).

-(a(X), -(b(X), -(c(X), nil))).
-(b(1), nil).
-(b(2), nil).
-(c(2), nil).
-(c(1), nil).

right(nil).
right(-(A, B)) :- !, right(B), right(A).
right(A) :- -(A, Body), right(Body).

van(nil).
van(-(A, B)) :- !, van(A), van(B).
van(A) :- -(A, Body), van(Body).

query(X) :- write(vanilla), nl, van(a(X)).
query(X) :- write(right), nl, right(a(X)).
```

4. Sorgenti C

```
/*
  p97.c
*/


#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#include "var.h"
#include "unify.h"
#include "struct.h"
#include "clause.h"
#include "status.h"

#define GOAL_L    80


static void solve(OBJ *g);
static int read_goal(OBJ *st);
static int more(void);
static void destroy_all( void );

static int nmall;
     /* nmall: Contatore delle allocazioni di memoria non ancora liberate
*/


void main(int argc, char *argv[])
{
  OBJ goal;

  if (argc && readsource(argv[1])) {
    nmall = 0;
    while (read_goal(&goal)) {
      solve(&goal);
      destroy_st(goal.child);
      if ( nmall ) {
        fprintf(stderr, "\nMemory not deallocated: %d\n", nmall);
        exit(1);
      }
    }
  } else
    fprintf(stderr, "usage: %s <name>\n", argv[0]);
}

static int read_goal(OBJ *st)
{
  char s[LINE_L + 1];

  printf("?- ");
  if (read_cl(s, stdin)) {
    fgetc(stdin);
    create_goal(s, st);
    return 1;
  } else
    return 0;
```

```
}
static int more()
{
  char s[LINE_L + 1];
  int c;

  printf("yes\n");
  print_goal_var(var_sp);
  printf("More? ");
  return (((c = *gets(s)) == ';') || (c == 'y') || (c == 'Y')) ? 1 : 0;
}


static void solve(OBJ *g)
{
  if (solve_1st(g->child, NULL))
    while (more() && solve_back());
  else
    while (solve_back() && more());

  destroy_all();
  printf("no\n");
}


static void destroy_all()
{
  release_var(NULL);
  destroy_sts();
}


void *mymalloc(unsigned n)
{
  void *p = malloc(n);

  if (p) {
    nmall++;
    return p;
  } else {
    fprintf(stderr, "Too many malloc()! (%d)\n", nmall);
    exit(1);
  }
}


void myfree(void *p)
{
  if (nmall == 0) {
    fprintf(stderr, "Too many free()!\n");
    exit(1);
  }

  free(p);
  nmall--;
}
```

```
/*
  p97.h
*/

void *mymalloc(unsigned n);
void myfree(void *p);


/*
  clause.c
*/


#include <stdio.h>
#include <string.h>
#include <ctype.h>

#include "clause.h"
#include "struct.h"
#include "p97.h"


static DB *src = NULL;

static int diff(char *f, char *line);


int readsource(char *fname)
{
  DB *p;
  FILE *f;

  if ((f = fopen(fname, "rt")) != NULL) {
    p = src = (DB *)mymalloc(sizeof(DB));
    while (read_cl(p->line, f)) {
      p->next = (DB *)mymalloc(sizeof(DB));
      p = p->next;
    }
    p->next = NULL;
    fclose(f);
    return 1;
  } else {
    fprintf(stderr, "%s: file not found\n", fname);
    return 0;
  }
}


int read_cl(char *s, FILE *f)
{
  int c;

  while (!feof(f)) {
    switch (c = fgetc(f)) {
      case ',':;
      case '(':;
      case ')':
        *s++ = TOK_SEP;
```

```
        *s++ = c;
        *s++ = TOK_SEP;
        break;

      case ':':
        *s++ = TOK_SEP;
        *s++ = c;
        break;

      case '-':
        *s++ = c;
        *s++ = TOK_SEP;
        break;

      case '%':
        while ((fgetc(f) != '\n') && (!feof(f)));
        break;

      case ' ':
      case '\t':
      case '\n':
        break;

      case '.':
        *s = '\0';
        return 1;
        break;

      default:
        *s++ =c;
        break;
    }
  }
  *s = '\0';
  return 0;
}


DB *find_functor(char *functor, DB *p)
{
  if (p = (p ? p->next : src))
    while (p && diff(functor, p->line))
      p = p->next;

  return p;
}


  /* ritorna se funct e la prima parola di line sono diverse */
static int diff(char *f, char *line)
{
  char s[TOK_L + 1];

  sscanf(line, "%s", s);
  return strcmp(f, s);
}
```

```
/*
  clause.h
*/

#ifndef CLAUSE_H
#define CLAUSE_H

#include <stdio.h>

#define LINE_L       400

struct db_t {
  char line[LINE_L + 1];
  struct db_t *next;
};

typedef struct db_t DB;


int readsource(char *s);
int read_cl(char *s, FILE *f);
DB *find_functor(char *s, DB *p);

#endif


/*
  unify.c
*/

#include <stdio.h>
#include <string.h>

#include "unify.h"
#include "struct.h"
#include "predef.h"
#include "var.h"


int unify(OBJ *goal, OBJ *head)
{
  int n;
  OBJ *argp1, *argp2;

  if ((isconst(goal) && !isatom(head)) || (!isatom(goal) && isconst(head)))
    return 0;

  else if (isconst(goal) && isconst(head))
    return (!strcmp(getobj(goal)->name, getobj(head)->name));

  else if (isvar(goal)) {
    link_var(getobj(goal), getobj(head));
    return 1;

  } else if (isvar(head)) {
    link_var(getobj(head), getobj(goal));
    return 1;

  } else {
    if (!strcmp(getobj(goal)->name, getobj(head)->name) &&
```

```
       (getobj(goal)->arity == getobj(head)->arity)) {
      argp1 = getobj(goal)->child;
      argp2 = getobj(head)->child;

      for (n = getobj(goal)->arity; n; n--) {
        if (!unify(argp1, argp2))
          return 0;
        argp1 = argp1->brother;
        argp2 = argp2->brother;
      }
      return 1;

    } else
      return 0;
  }
}


DB *find_cl(OBJ *head, CLAUSE *cl, DB *p)
{
  char sout[LINE_L + 1];
  STACK *tmp;

  if (ispredef(head->name)) {
    if (p)
      return NULL;
    else {
      cl->body = cl->head = NULL;
      return (DB *)!NULL;
    }
  }

  tmp = mark_var();
  while (p = find_functor(head->name, p)) {
    rename_cl(p->line, sout);
    create_cl(sout, cl);

    if (unify(head, cl->head))
      return p;

    destroy_st(cl->head);
    destroy_st(cl->body);
    free_var_st(head);
    release_var(tmp);
  }
  cl->head = cl->body = NULL;
  return NULL;
}


/*
  unify.h
*/

#ifndef UNIFY_H
#define UNIFY_H

#include "struct.h"
#include "clause.h"
```

```
int unify(OBJ *goal, OBJ *head);
DB *find_cl(OBJ *head, CLAUSE *cl, DB *p);

#endif


/*
  struct.c
*/

#include <stdio.h>
#include <string.h>
#include <assert.h>
#include <ctype.h>

#include "struct.h"
#include "var.h"
#include "p97.h"


#define NVAR         30            /* numero var. diverse in una clausola
*/
#define BUF_L        1024


static struct tab_t {
  char pub[NAME_L + 1], priv[NAME_L + 1];
} tab[NVAR];

typedef struct tab_t TAB;

static char *inputp;        /* contatore per mystrtok */
static TAB *tabp = tab;     /* tabp punta al primo vuoto */


/*

 Esempio di tabella:

          --------------
          |pub   |  priv|
          --------------
  tab  -->| X    | X_000|
          | Y    | X_001|
          | Res  | X_002|
          | Acc  | X_003|
  tabp -->|      |      |
          |      |      |
          |      |      |
          --------------

*/


static int isatomic(void);
static void dolist(OBJ *this);
static void reset_tab( void );
static TAB *in_tab(char *s);
static char *add_tab(char *s);
static char *mystrtok( void );
```

```
void create_cl(char *s, CLAUSE *cl)
{
  char *p, *p2;

  if ((p = strstr(s, " :- ")) != NULL) {
    p2 = p + 4;
    *p = '\0';
    cl->head = (OBJ *) mymalloc(sizeof(OBJ));
    cl->body = (OBJ *) mymalloc(sizeof(OBJ));
    create_head(s, cl->head);
    create_goal(p2, cl->body);
  } else {
    cl->head = (OBJ *) mymalloc(sizeof(OBJ));
    create_head(s, cl->head);
    cl->body = NULL;
  }
}


void create_goal(char *s, OBJ *st)
{
  static char buf[BUF_L + 1];

  strcpy(buf, "dummy ( ");
  strcat(buf, s);
  strcat(buf, " )");

  inputp = buf;
  st->brother = NULL;
  dolist(st);
}


void create_head(char *s, OBJ *st)
{
  inputp = s;
  st->brother = NULL;
  dolist(st);
}


int isconst(OBJ *p)
{
  return (isatom(p) && !isupper(getobj(p)->name[0]));
}


int isatom(OBJ *p)
{
  return (getobj(p)->arity == 0);
}


int isvar(OBJ *p)
{
  return (!isconst(p) && isatom(p));
}
```

```
void print_st(OBJ *st)
{
  OBJ *argp;
  int n;

  if (st) {
    if (isupper(st->name[0]))
      print_var(inlist_var(st->name));
    else {
      printf("%s", st->name);

      if (n = st->arity) {
        printf("(");
        argp = st->child;
        for (; n; n--) {
          print_st(argp);
          printf((argp = argp->brother) ? "," : ")");
        }
      }
    }
  }
}


void rename_cl(char *s, char *sout)
{
  char tok[TOK_L + 1];
  TAB *p;

  reset_tab();
  inputp = s;
  sout[0] = '\0';
  while (*(strcpy(tok, mystrtok()))) {   /*  finisce quando tok = "" */
    if (!strcmp(tok, "_"))               /* "_" == variabile non in lista
*/
      strcat(sout, add_tab(tok));
    else if (isupper(tok[0])) {          /* e' una variabile */
      if ((p = in_tab(tok)) != NULL)
        strcat(sout, p->priv);
      else
        strcat(sout, add_tab(tok));
    } else
      strcat(sout, tok);

    strcat(sout, " ");
  }
}


void copy_st(OBJ **dest, OBJ *src, int flag)
{
  if (src) {
    (*dest) = (OBJ *) mymalloc(sizeof(OBJ));

    strcpy((*dest)->name, src->name);
    (*dest)->arity = src->arity;

    if (src->arity)
      copy_st(&(*dest)->child, src->child, !IGNOREBROTHER);
```

```
    else
      (*dest)->child = NULL;

    if ((flag != IGNOREBROTHER) && src->brother)
      copy_st(&(*dest)->brother, src->brother, !IGNOREBROTHER);
    else
      (*dest)->brother = NULL;
  } else
    *dest = NULL;
}


void scan_st(OBJ *st, int flag)
{
  if (isupper(st->name[0]) && ! inlist_var(st->name))
    add_var(st->name);

  if (st->arity)                                  /* scorro st */
    scan_st(st->child, !IGNOREBROTHER);

  if ((flag != IGNOREBROTHER) && st->brother)   /* scanna anche i fratelli
*/
    scan_st(st->brother, !IGNOREBROTHER);
}

void spec_st(OBJ **dest, OBJ *src, int flag)
{
  OBJ *p = src;

  (*dest) = (OBJ *) mymalloc(sizeof(OBJ));

  if (isupper(src->name[0]))
    src = getobj(src);

  strcpy((*dest)->name, src->name);
  (*dest)->arity = src->arity;

  if (src->arity)
    spec_st(&(*dest)->child, src->child, !IGNOREBROTHER);
  else
    (*dest)->child = NULL;

  src = p;
  if ((flag != IGNOREBROTHER) && src->brother)
    spec_st(&(*dest)->brother, src->brother, !IGNOREBROTHER);
  else
    (*dest)->brother = NULL;
}

void destroy_st(OBJ *st)
{
  if (st) {
    destroy_st(st->child);
    destroy_st(st->brother);
    myfree(st);
  }
}
void free_var_st(OBJ *st)
{
  if (isupper(st->name[0]))
```

```
    free_var(inlist_var(st->name));

  if (st->arity)                    /* libera le var. di st */
    free_var_st(st->child);

  if (st->brother)                  /* libera le var. dei fratelli */
    free_var_st(st->brother);
}


  /* static functions */

static void dolist(OBJ *this)
{
  char c;
  OBJ *argp;

  strcpy(this->name, mystrtok());

  if (!isatomic()) {
    this->arity = 1;

    this->child = (OBJ *)mymalloc(sizeof(OBJ));
    argp = this->child;
    dolist(argp);
    while ((c = *mystrtok()) == ',') {
      this->arity++;
      argp->brother = (OBJ *)mymalloc(sizeof(OBJ));
      argp = argp->brother;
      dolist(argp);
    }
    argp->brother = NULL;

    assert(c == ')');
  } else {
    this->arity = 0;
    this->child = NULL;
  }
}


static void reset_tab()
{
  tabp = tab;
}


static TAB *in_tab(char *s)       /*  se s e' nella tabella, ritorna il
                                      puntatore relativo, altrimenti NULL */
{
  TAB *p;

  for (p = tab; ((p < tabp) && (strcmp(s, p->pub))); p++);
  return (p == tabp) ? NULL : p;
}


static char *add_tab(char *s)     /* aggiunge la var. s  e il suo nuovo
nome
```

```
                                        alla tabella e ritorna il nuovo nome
*/
{
  strcpy(tabp->pub, s);
  strcpy(tabp->priv, new_var());
  tabp++;
  assert((tabp - tab) < NVAR);
  return (tabp - 1)->priv;
}


static int isatomic()
{
  char *tmp = inputp;

  if (*mystrtok() == '(')
    return 0;
  else {
    inputp = tmp;
    return 1;
  }
}

static char *mystrtok()
/*
  Legge, dalla stringa puntata da inputp,
  la prima stringa ( token ) compresa tra i
  caratteri di separazione TOK_SEP, e ritorna
  il puntatore alla stringa letta
*/
{
  static char buf[TOK_L];
  char *p;

  p = buf;
  for (; *inputp && (*inputp == TOK_SEP) ; inputp++);
  for (; (*inputp != TOK_SEP) && (*inputp) ; inputp++,p++)
    *p = *inputp;

  *p = '\0';
  assert(strlen(buf) < NAME_L);
  return buf;
}

/*
  struct.h
*/

#ifndef STRUCT_H
#define STRUCT_H

#define IGNOREBROTHER 0
#define TOK_SEP       ' '
#define NAME_L        10
#define TOK_L         20

struct obj_t {
  char name[NAME_L + 1];
  int arity;
  struct obj_t *brother,
```

```
               *child;
};

typedef struct obj_t OBJ;

struct clause_t {
  OBJ *head, *body;
};

typedef struct clause_t CLAUSE;


void  create_cl(char *s, CLAUSE *cl);
void  rename_cl(char *s, char *sout);
void  create_goal(char *s, OBJ *str);
void  create_head(char *s, OBJ *this);
void  destroy_st(OBJ *st);
void  copy_st(OBJ **dest, OBJ *src, int flag);
void  scan_st(OBJ *st, int flag);
void  spec_st(OBJ **dest, OBJ *src, int flag);
void  free_var_st(OBJ *st);
void  print_st(OBJ *str);
int   isconst(OBJ *p);
int   isvar(OBJ *p);
int   isatom(OBJ *p);


#endif


/*
  status.c
*/

#include <assert.h>
#include <stdio.h>
#include <string.h>

#include "unify.h"
#include "status.h"
#include "clause.h"
#include "var.h"
#include "predef.h"
#include "p97.h"
#include "stack.h"


STACK *sts_sp = NULL;


int push_sts(OBJ *g, STATUS *father, CLAUSE *cl)
{
  STATUS sts;

  spec_st(&sts.goal, g, IGNOREBROTHER);

  if (sts.choose = find_cl(sts.goal, cl, NULL)) {
    sts.mark = mark_var();
    copy_st(&sts.goal_brother, g->brother, !IGNOREBROTHER);
    sts.cutted = 0;
```

```
    sts.father = father;
    push(&sts, sizeof(STATUS), &sts_sp);
    return 1;
  } else {
    destroy_st(sts.goal);
    return 0;
  }
}


void pop_sts()
{
  STATUS *sts = sts_sp->data;

  assert(sts_sp);
  destroy_st(sts->goal_brother);
  destroy_st(sts->goal);
  pop(&sts_sp);
}


void redo()
{
  STATUS *sts = sts_sp->data;

  assert(sts->goal->brother == NULL);
  free_var_st(sts->goal);     /* ripristina stack delle variabili */
  release_var(sts->mark);
}

int solve_back()
{
  CLAUSE cl;
  STATUS *sts;
  STATUS * p;
  int r;

  while (sts_sp) {
    cl.head = cl.body = NULL;
    sts = sts_sp->data;
    redo();

    if (!sts->cutted && (sts->choose = find_cl(sts->goal, &cl, sts-
>choose)))
      if (!cl.body || solve_1st(cl.body->child, sts))
        if (solve_1st(sts->goal_brother, sts->father)) {
          p = sts->father;
          r = 1;
          while (p && (r = solve_1st(p->goal_brother, p->father)))
            p = p->father;

          if (r) {
            destroy_st(cl.head);
            destroy_st(cl.body);
            return r;
          }
        }

    destroy_st(cl.head);
    destroy_st(cl.body);
```

```
    pop_sts();
  }
  return 0;
}


int solve_1st(OBJ *g, STATUS *father)
{
  STATUS *sts = (sts_sp ? sts_sp->data : NULL);
  STACK *tmp;
  CLAUSE cl;
  int r;

  cl.head = cl.body = NULL;
  if (g) {
    if (push_sts(g, father, &cl)) {
      sts = sts_sp->data;
      tmp = sts_sp;

      if (cl.body) {
        if (r = solve_1st(cl.body->child, tmp->data))
          r = solve_1st(sts->goal_brother, ((STATUS *)tmp->data)->father);

      } else {
        if (r = predef(g)) /* r == 0 solo se predefinito e fallito */
          r = solve_1st(sts->goal_brother, ((STATUS *)tmp->data)->father);
      }
    } else
      r = 0;

  } else
    r = 1;

  destroy_st(cl.head);
  destroy_st(cl.body);
  return r;
}


void destroy_sts()
{
  while (sts_sp)
    pop_sts();
}


/*
  status.h
*/

#ifndef STATUS_H
#define STATUS_H

#include "struct.h"
#include "stack.h"
#include "clause.h"
#include "var.h"

struct status_t {
  OBJ *goal;
```

```
  DB *choose;
  STACK *mark;
  OBJ *goal_brother;
  struct status_t *father;
  unsigned cutted:1;
};

typedef struct status_t STATUS;

extern STACK *sts_sp;

int push_sts(OBJ *g, STATUS *father, CLAUSE *cl);
void pop_sts( void );
void redo( void );
int solve_1st(OBJ *p, STATUS *f);
int solve_back( void );
void destroy_sts( void );

#endif


/*
  var.c
*/

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <assert.h>

#include "var.h"
#include "p97.h"

STACK  *var_sp = NULL;


OBJ *getobj(OBJ *st)

  /* ritorna il ptr all'ultimo OBJ della lista di istanziazioni di st

     variabile    |   struttura

         X  ----------->  X1
         ^
         |
         v
         X1 ----------->  a( .... )

    In questo caso getobj( X ) ritorna il ptr a a( .... )
  */
{
  VAR *p;
  OBJ *p2;

  if (st && isupper(st->name[0]) && (p = inlist_var(st->name)) && p->value)
    return ( p2 = getobj(p->value) ) ? p2 : p->value;
  else
    return st;
  /* se st non e' NULL, la struttura st riferisce una variabile e la
    variabile esiste gia' nella lista, allora la funzione si richiama
```

```
    ricorsivamente sul valore della variabile in lista.
  */
}


void print_goal_var(STACK *p)
{
  while (p) {
    if (strncmp(((VAR *)p->data)->name, "X_", 2)) {
      printf("%s = ", ((VAR *)p->data)->name);
      print_var(((VAR *)p->data));
      printf("\n");
    }
    p = p->prev;
  }
}


void print_var(VAR *p)
{
  if (p) {
    if (p->value) {
      if (isupper(p->value->name[0]))
        print_var(inlist_var(p->value->name));
      else
        print_st(p->value);
    } else
      printf("%s", p->name);
  } else
    printf("<var>");
}


STACK *mark_var()
{
  return var_sp;
}

void release_var(STACK *marked)
{
  while (var_sp != marked) {
    destroy_var((VAR *)var_sp->data);
    pop(&var_sp);
  }
}

void destroy_var(VAR *p)
{
  if (p)
    if (p->value) {
      assert(p->value->brother == NULL);
      destroy_st(p->value);
    }
}
VAR *inlist_var(char *varname)
{
  STACK *p = var_sp;

  while ((p != NULL) && (strcmp(((VAR *)p->data)->name, varname)))
    p = p->prev;
```

```
  return ( p ? (VAR *)p->data : NULL );
}


char *new_var()
{
  static char buf[NAME_L];
  static long var_count = 0;

  assert(var_count < 10000000L);
  sprintf(buf, "X_%07ld", var_count++);
  return buf;
}


void link_var(OBJ *stv, OBJ *st)
{
  VAR *v;

  if (!( v = inlist_var(stv->name)) )
    v = add_var( stv->name );

  copy_st( &v->value, st, IGNOREBROTHER ); /* copia ma ignora i fratelli */
  scan_st(st, IGNOREBROTHER);              /* scanna ma ignora i fratelli
*/
}


VAR *add_var(char *s)
{
  VAR v;

  assert(strlen(s) < NAME_L);
  strcpy(v.name, s);
  v.value = NULL;
  push(&v, sizeof(VAR), &var_sp);
  return ((VAR *)var_sp->data);
}


void free_var(VAR *p)
{
  if (p) {
    destroy_st(p->value);
    p->value = NULL;
  }
}




/*
  var.h
*/

#ifndef VAR_H
#define VAR_H
```

```
#include "struct.h"
#include "stack.h"

struct var_t {
  char name[NAME_L + 1];
  OBJ *value;
};

typedef struct var_t VAR;

extern STACK *var_sp;

VAR  *inlist_var(char *varname);
OBJ  *getobj(OBJ *st);
char *new_var(void);
void  link_var(OBJ *st, OBJ *st2);
STACK *mark_var( void );
void  release_var(STACK *p);
void  free_var(VAR *p);
void  print_var(VAR *p);
void  print_goal_var(STACK *p);
void  destroy_var(VAR *p);
VAR  *add_var(char *s);


#endif


/*
  stack.c
*/

#include "mem.h"
#include "stack.h"
#include "p97.h"


void push(void *data, unsigned size, STACK **this)
{
  STACK *tmp = *this;

  (*this) = (STACK *) mymalloc(sizeof(STACK));
  (*this)->data = mymalloc(size);
  memcpy((*this)->data, data, size);
  (*this)->prev = tmp;
}




void pop(STACK **this)
{
  STACK *tmp = *this;

  if (*this) {
    myfree((*this)->data);
    (*this) = (*this)->prev;
```

```
    myfree(tmp);
  }
}


/*
  stack.h
*/

#ifndef STACK_H
#define STACK_H

struct stack_t {
  void *data;
  struct stack_t *prev;
};

typedef struct stack_t STACK;


void push(void *data, unsigned size, STACK **this);
void pop(STACK **this);

#endif


/*
  predef.c
*/


#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <assert.h>

#include "struct.h"
#include "status.h"
#include "predef.h"
#include "p97.h"

#define PREDEFC  7

static int cut(OBJ *st);
static int fail(OBJ *st);
static int integer(OBJ *st);
static int integer2(OBJ *st);
static int write(OBJ *st);
static int nl(OBJ *st);
static int nonvar(OBJ *st);
static int test(OBJ *st);


struct {
  char name[TOK_L];
  int (*func)(OBJ *st);
} predefv[] = {
  { "!", cut },
  { "fail", fail },
  { "nonvar", nonvar },
```

```
  { "integer", integer },
  { "write", write },
  { "nl", nl },
  { "test", test }
};


int ispredef(char *s)
{
  int n;

  for (n = 0; n < PREDEFC; n++)
    if (!strcmp(predefv[n].name, s))
      return n + 1;

  return 0;
}


int predef(OBJ *st)
{
  int n;

  st = getobj(st);
  if (n = ispredef(st->name))
    return (*predefv[n - 1].func)(st);
  else
    return NOPREDEF;
}


static int cut(OBJ *st)
{
  STACK *tmp;

  for (tmp = sts_sp; tmp->data != ((STATUS *)sts_sp->data)->father; ) {
    ((STATUS *)tmp->data)->cutted = 1;
    tmp = tmp->prev;
  }

  ((STATUS *)sts_sp->data)->father->cutted = 1;
  return 1;
}


static int fail(OBJ *st)
{
  return 0;
}


static int nonvar(OBJ *st)
{
  return isvar(st->child) ? 0 : 1;
}


static int test(OBJ *st)
{
  int sum;
```

```
  OBJ
    tmp,
    *C    = st->child,
    *Rout = st->child->brother,
    *A    = st->child->brother->brother,
    *B    = st->child->brother->brother->brother,
    *Rin  = st->child->brother->brother->brother->brother;

  if ((st->arity != 5) || !integer2(A) || !integer2(B) || !integer2(Rin))
    return 0;

  sum = atoi(getobj(A)->name)+atoi(getobj(B)->name)+atoi(getobj(Rin)-
>name);

  tmp.arity = 0;
  tmp.brother = 0;
  tmp.child = 0;

  assert(!isvar(C));

  if (!(atoi(getobj(C)->name) == (sum % 10)))
    return 0;

  if (isvar(Rout)) {
    itoa(sum / 10, tmp.name, 10);
    link_var(getobj(Rout), &tmp);
  } else
    if (!(atoi(getobj(Rout)->name) == (sum / 10)))
      return 0;

  return 1;
}


static int integer(OBJ *st)
{
  return integer2(st->child);
}


static int integer2(OBJ *st)
{
  char *p;

  if (isconst(st)) {
    for (p = getobj(st)->name; *p && isdigit(*p); p++);
    return *p ? 0 : 1;
  } else
    return 0;
}

static int write(OBJ *st)
{
  print_st(st->child);
  return 1;
}

static int nl(OBJ *st)
{
  printf("\n");
```

```
  return 1;
}


/*
  predef.h
*/

#ifndef PREDEF_H
#define PREDEF_H

#include "struct.h"

#define NOPREDEF        (-1)

int predef(OBJ *st);
int ispredef(char *s);

#endif
```